Perfect Zero-Knowledge in Constant Rounds

Mihir Bellare* Silvio Micali† Rafail Ostrovsky‡

MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

Quadratic residuosity and graph isomorphism are classic problems and the canonical examples of zero-knowledge languages. However, despite much research effort, all previous zero-knowledge proofs for them required either unproven complexity assumptions or an unbounded number of rounds of message exchange.

For both (and similar) languages, we exhibit zero-knowledge proofs that require 5 rounds and no unproven assumptions. Our solution is essentially optimal, in this setting, due to a recent lower bound argument of Goldreich and Krawczyk.

1 Introduction

Interactive proofs and especially zero-knowledge ones have found many applications, most notably in the field of secure protocols. In all such proofs, interaction is the crucial resource, as prover and verifier exchange messages in rounds. The fundamental problem here is whether the number of rounds induces a hierarchy. That is, can we prove more languages in zero knowledge given more rounds?

In a cryptographic setting the answer is no. But little is known for perfect zero-knowledge proofs. These are the ones that can be proven to be zero-knowledge without making use of unproven complexity assumptions. For this reason, perfect zero-knowledge is the right context for studying the *intrinsic* properties of this notion.

There is a gap in what we are able to prove about perfect Zero-Knowledge (ZK): Goldreich and Krawczyk [3] show that a language outside BPP requires more than 3 rounds from any perfect ZK proof. On the other hand, the classic examples for perfect zero-knowledge, the languages of quadratic residuosity and graph isomorphism, required an unbounded number of rounds. In this paper we show

Theorem *The languages of graph isomorphism and quadratic residuosity have* 5 *round perfect zero knowledge interactive proofs.*

More generally, we show that any *random self-reducible* language [7] has a 5 round perfect zero knowledge interactive proof.

Perfect zero-knowledge works by carefully exploiting the structure of the problem at hand. It is important not only in practice (where we do not know which functions are one-way), but also in a theoretical setting. For example, our best way to prove that a language L, for which no efficient algorithm is known, is not NP-complete involves exhibiting a perfect zero-knowledge proof for it [2].

Let us now see, at a very high level, why achieving perfect ZK in constant rounds is hard. Essentially, in a ZK proof, the confidence that a theorem is true is transferred by discrete amounts, or *tokens*. Each token consists of an elementary protocol that decreases the probability of error by a factor of, say, 2. Thus, after k tokens have been exchanged, this probability will be reduced to 2^{-k}. Tokens can be exchanged sequentially or in parallel (that is concurrently). The ZK constraint implies that the messages exchanged for a single token can be simulated in expected polynomial time. That is, a probabilistic, efficient simulator can output a token in expected, say, 2 trials without any intervention of the prover, and knowing nothing about the proof. Thus, if the proof transfers tokens one at a time, it is easy for the simulator to generate the view relative to k of them: in expected two trials the simulator will output a "good" first token; after that, in expected two more trials, will output a second "good" token, and so on. Thus, overall, the entire view of the protocol can be

* Supported in part by NSF grant CCR-87-19689.

† Supported in part by NSF grant DCR-84-13577 and ARO grant DAALO3-86-K-0171.

‡ Part of this work was done at Boston University, Department of Computer Science, partially supported by NSF grant DCR-86-07492.

simulated in expected $2k$ trials. If, however, the prover transfers tokens "all together," the job of the simulator is much harder: the expected number of trials so that all k of them are *simultaneously* "good," is 2^k.

This phenomenon – which will be made more precise in the context of graph isomorphism – is what defeated researchers ever since ZK came about.

The main result of this paper is a technique for squeezing the number of rounds in a interactive proof, while preserving simulatability.

The proof of our main theorem involves several new ideas; in particular, a non-cryptographic committal scheme and a novel method of simulation in modes. In fact, although the protocol only requires 5 rounds and is surprisingly simple, the simulation argument is complex.

Moreover, our result is essentially optimal due to the general 4-round lower bound of Goldreich and Krawczyk [3].

In this extended abstract we confine ourselves to a proof that graph isomorphism has a 5 round perfect zero knowledge interactive proof.

2 Definitions

2.1 Probability Spaces and Algorithms

These notations and conventions for probabilistic algorithms are derived from [6] and further extended.

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$"; if it receives two we write "$A(\cdot,\cdot)$", and so on.

If A is a probabilistic algorithm then, for any input i the notation $A(i)$ refers to the probability space which to the string σ assigns the probability that A, on input i, outputs σ.

If S is a probability space we denote by $\mathbf{P}_S(e)$ the probability that S associates with element e. We denote by $[S]$ the set of elements to which S assigns positive probability.

If $f(\cdot)$ and $g(\cdot,\cdots)$ are probabilistic algorithms then $f(g(\cdot,\cdots))$ is the probabilistic algorithm obtained by composing f and g (i.e. running f on g's output). For any inputs $x, y, \ldots$ the associated probability space is denoted $f(g(x,y,\cdots))$.

If S is a probability space then $x \leftarrow S$ denotes the algorithm which assigns to x an element randomly selected according to S (that is, x is assigned the value e with probability $\mathbf{P}_S(e)$) (in the case that $[S]$ consists of only one element e we write $x \leftarrow e$ rather than $x \leftarrow \{e\}$).

For probability spaces $S, T, \ldots$, the notation

$$\mathbf{P}(p(x,y,\cdots) : x \leftarrow S; y \leftarrow T; \cdots)$$

denotes the probability that the predicate $p(x,y,\cdots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc. The notation

$$\{ f(x,y,\cdots) : x \leftarrow S; y \leftarrow T; \cdots \}$$

denotes the probability space which to the string σ assigns the probability

$$\mathbf{P}(\sigma = f(x,y,\cdots) : x \leftarrow S; y \leftarrow T; \cdots) \, ,$$

f being some function.

If S is a finite set we will identify it with the probability space which assigns to each element of S the uniform probability $U(S) = \frac{1}{|S|}$. (Then $x \leftarrow S$ denotes the operation of selecting an element of S uniformly at random, and $\mathbf{P}_S(e) = U(S)$ for any $e \in S$).

We let PPT denote the set of probabilistic (expected) polynomial time algorithms.

2.2 Interactive TMs and Protocols

Full descriptions of interactive Turing Machines and protocols appear in [5]; we will only summarize the notation we use.

The probability that (A, B) accepts the common input x is denoted

$$\mathbf{P}(\ (A,B) \text{ accepts } x\ ) \, ,$$

and the probability space of all conversations between A and B on input x is denoted $(A \leftrightarrow B)(x)$ (the probability in both cases is taken over the random tapes of both A and B).

Sometimes we want to make the coin tosses of B explicit. For any $R \in \{0,1\}^*$ we write $B(R)$ for the (deterministic) machine B with R as its random tape. Then $(A \leftrightarrow B(R))(x)$ denotes the probability space of conversations between A and $B(R)$ on input x (the probability is over the random tapes of A).

We let $B(R; x, \alpha_1\beta_1 \ldots \alpha_{i-1}\beta_{i-1})$ denote the next message that $B(R)$ sends when the conversation up to this point was $\alpha_1\beta_1 \ldots \alpha_{i-1}\beta_{i-1}$.

Finally, it is convenient to define the *state* of the ITM B at any point in its computation as consisting of the contents of its tapes, the positions of the heads on these tapes, and the internal state of the machine.

2.3 Interactive Proof Systems and Zero Knowledge

Definition 2.1 An interactive protocol (P, V) is an *interactive proof system* for the language L if the following conditions hold:

- *Completeness*: For every $x \in L$,

$$\mathbf{P}(\ (P,V) \text{ accepts } x) \geq 1 - 2^{-|x|} \, .$$

- *Soundness*: For every ITM $\widehat{P}$ and every $x \notin L$,
$$\mathbf{P}(\,(\widehat{P}, V) \text{ accepts } x) \le 2^{-|x|}\,.$$

P and V are referred to as the *prover* and the *verifier* respectively.

The view of the verifier during an interaction with the prover is everything he sees: that is, his own coin tosses and the conversation between himself and the prover. Accordingly we define

Definition 2.2 Let $(P, \widehat{V})$ be an interactive protocol and let $x \in \{0,1\}^*$. The *view* of $\widehat{V}$ on input x is the probability space

$$View_{(P,\widehat{V})}(x) = \{\, (R, C) : R \leftarrow \{0,1\}^{p(|x|)}\,;$$
$$C \leftarrow (P \leftrightarrow \widehat{V}(R))(x)\,\}\,,$$

where p is a polynomial bounding the running time of $\widehat{V}$.

Definition 2.3 An interactive proof system (P, V) for L is a *perfect zero knowledge* interactive proof system for L if for every polynomial time ITM $\widehat{V}$ there exists a PPT algorithm $S_{\widehat{V}}(\cdot)$ such that for all $x \in L$ it is the case that $S_{\widehat{V}}(x) = View_{(P,\widehat{V})}(x)$ (this $S_{\widehat{V}}$ is called the *simulator*).

Perfect zero knowledge represents the strongest notion of zero knowledge in that we require the distribution produced by the simulator to exactly equal the view of the verifier. Weaker versions (namely *computational* and *statistical* zero knowledge) are defined in [5], but we will not be concerned with those here, and when we say "zero knowledge" we always mean perfect.

3 Graph Isomorphism: Background

3.1 Preliminaries

We consider undirected graphs $G = (V, E)$ in which we always take V to be $[n] = \{1, 2, \ldots, n\}$ when G has n nodes. We represent G by its adjacency matrix $[a_{ij}]$ defined by

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Since the graph is undirected its adjacency matrix is symmetric.

Let S_n denote the set of permutations on $[n]$. Permuting the nodes of a graph $G = ([n], E)$ according to a $\pi \in S_n$ yields a graph $\pi(G) = ([n], \pi(E))$ defined by

$$\pi(E) = \{\, (\pi(i), \pi(j)) : (i,j) \in E\,\}\,.$$

The graphs $G_0 = ([n], E_0)$ and $G_1 = ([n], E_1)$ are *isomorphic* (written $G_0 \cong G_1$) if there is a $\pi \in S_n$ such that $G_1 = \pi(G_0)$. The language of graph isomorphism is

$$GI = \{\, (G_0, G_1) : G_0 \cong G_1 \,\}$$

where the graphs are represented by their adjacency matrices.

Isomorphism defines an equivalence relation on the set of n node graphs, and we let

$$[G] = \{\, \pi(G) : \pi \in S_n \,\}$$

denote the isomorphism class of the n node graph G. For an n node graph H we further let

$$[G \rightarrow H] = \{\, \pi \in S_n : \pi(G) = H \,\}$$

denote the set of all permutations mapping G to H. An important fact about this set is

Lemma 3.1 Let G be an n node graph. The collection of sets $\{\, [G \rightarrow H] \,\}_{H \in [G]}$ form a partition of S_n and moreover all the sets in this partition are of the same size.

A basic component of graph isomorphism protocols is the creation of a random isomorphic copy of a given graph G. That is, given the adjacency matrix of G we wish to create the adjacency matrix of a random element of $[G]$. The following lemma is useful in this regard.

Lemma 3.2 If π, H are chosen as $\pi \leftarrow S_n$; $H \leftarrow \pi(G)$ then

(1) H is randomly and uniformly distributed over $[G]$
(2) π is randomly and uniformly distributed over $[G \rightarrow H]$.

3.2 The Goldreich-Micali-Wigderson Protocol

The first zero knowledge proof for graph isomorphism was given by Goldreich, Micali and Wigderson [4] and we use many of their ideas in our protocol. It will be helpful to briefly review their protocol here.

The protocol in [4] consists of serial repetitions of a small *atomic* protocol. This atomic protocol is as follows. On input a pair of isomorphic graphs (G_0, G_1) the prover sends the verifier a random isomorphic copy H of G_0, and the verifier responds by sending a bit q selected at random. The prover must now provide an isomorphism ϕ between G_q and H. If G_0 and G_1 are really isomorphic he can always do this, but if not then the probability that he could do it is $\frac{1}{2}$, regardless of how he might attempt to select H.

The simulator for this protocol begins by picking a bit i and a permutation φ at random, and then constructing the random isomorphic copy $H = \varphi(G_i)$ of G_i. He now *runs* the verifier on input H. With probability $\frac{1}{2}$ the verifier's request q is equal to i, and the simulator can provide φ as the isomorphism. In expected two tries, he has a simulation of the atomic protocol.

Serial repetitions of the atomic proof are easily simulatable in the same manner – each run can be simulated in expected two tries and thus the whole in expected polynomial time.

3.3 Why Does Straightforward Parallelization Fail?

Could we run the above atomic protocol k times in parallel and maintain zero knowledge? Difficulties crop up when we are dealing with cheating verifiers.

For example, suppose we run k versions of the atomic protocol in parallel, and the prover is dealing with a cheating verifier $\widehat{V}$ who acts as follows. When the prover sends randomly selected graphs $H_1, \ldots, H_k$ each isomorphic to G_0, the verifier $\widehat{V}$ sends back bits $q_1, \ldots, q_k$ which are computed, say, by hashing in a very complicated way the inputs and the prover's message: that is,

$$(q_1, \ldots, q_k) = \textit{complex-hash}(G_0, G_1, H_1, \ldots, H_k) .$$

Although $q_1, \ldots, q_k$ have not been picked by the prescribed protocol, the prover at this stage will, for each i, provide an isomorphism $\phi_i \in [G_{q_i} \rightarrow H_i]$. The simulator must generate this same view of the cheating verifier. It is not at all clear how he can do this. There is no obvious way in which he can "anticipate" the verifier's queries. In essence, he must be able to simultaneously satisfy many conditions: namely, he must come up with randomly distributed graphs $H_1, \ldots, H_k$ for which he *also* knows isomorphisms $\phi_i \in [G_{q_j} \rightarrow H_i]$ where $(q_1, \ldots, q_k) = \textit{complex-hash}(G_0, G_1, H_1, \ldots, H_k)$. This may not be possible in polynomial time unless graph isomorphism is easy. Thus the naive parallel execution of the original atomic proof yields a proof system which may not be a zero knowledge one.

Furthermore Goldreich and Krawczyk [3] provide evidence that these difficulties are intrinsic: their results imply that the protocol of [4] would be parallelizable only if the language *GI* was in BPP, and the latter seems unlikely.

4 A Constant Round Protocol for Graph Isomorphism

4.1 The Protocol

The common input is a pair of n node graphs (G_0, G_1), and k denotes the length of this input.

Program for the Prover P:

Auxiliary Input: $\pi \in [G_1 \rightarrow G_0]$

(P1) P selects $\gamma_0, \gamma_1 \leftarrow S_n$ and sets $A_0 = \gamma_0(G_0)$, $A_1 = \gamma_1(G_0)$. His message to V is $\alpha_1 = (A_0, A_1)$.

(P2) P receives a message β_1. He now selects $\varphi_1, \ldots, \varphi_k \leftarrow S_n$ and sets $H_i = \varphi_i(G_0)$ for all $i = 1, \ldots, k$. His message to V is $\alpha_2 = (H_1, \ldots, H_k)$.

(P3) P receives a message β_2. He checks that

- β_1 is of the form $(Q_1, \ldots, Q_k)$ where each Q_i is an n node graph
- β_2 is of the form $((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$ where $q_i \in \{0,1\}$ and $\mu_i \in S_n$ for each i
- $Q_i = \mu_i(A_{q_i})$ for each i.

If any of these checks fail, P aborts the protocol here. Otherwise, P sets

$$\phi_i = \begin{cases} \varphi_i & \text{if } q_i = 0 \\ \varphi_i \pi & \text{if } q_i = 1. \end{cases}$$

for $i = 1, \ldots, k$. His message to V is then $\alpha_3 = ((\gamma_0, \gamma_1), (\phi_1, \ldots, \phi_k))$.

end of Program for P

Remark: Note that given an isomorphism between G_1 and G_0, P is actually PPT.

Program for the Verifier V:

(V1) V receives a message α_1. He checks that α_1 is of the form (A_0, A_1) where each A_i is an n node graph, and if this is not the case then **rejects**. Otherwise he selects $\mu_1, \ldots, \mu_k \leftarrow S_n$; $q_1, \ldots, q_k \leftarrow \{0,1\}$, and sets $Q_i = \mu_i(A_{q_i})$ for all $i = 1, \ldots, k$. His message to P is $\beta_1 = (Q_1, \ldots, Q_k)$.

(V2) V receives a message α_2. He sends to P the message $\beta_2 = ((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$.

(V3) V receives a message α_3. He checks that

- α_2 is of the form $(H_1, \ldots, H_k)$ where each H_i is an n node graph
- α_3 is of the form $((\gamma_0, \gamma_1), (\phi_1, \ldots, \phi_k))$ where each γ_i and each ϕ_i is in S_n
- $\gamma_0(G_0) = A_0$ and $\gamma_1(G_0) = A_1$ and $\phi_i(G_{q_i}) = H_i$ for all $i = 1, \ldots, k$.

If this is the case then V **accepts**, else V **rejects**.

end of Program for V

We will prove that (P, V) constitutes a zero knowledge interactive proof for GI.

4.2 A Brief Intuitive Look at the Protocol

Let us try to explain the main ideas behind this protocol, building on the discussion of §3.3.

In order to prevent the verifier from picking his questions as

$$\textit{complex-hash}(G_0, G_1, H_1, \ldots, H_k)$$

or any other bizarre function of the message he receives we will have him *commit* to his questions *before* the prover sends his test graphs $H_1, \ldots, H_k$. After receiving the test graphs from the prover he decommits these questions which the prover then proceeds to answer. The hope, intuitively, is that if the verifier is in some sense committed to his questions before he sees the test graphs then he cannot keep changing these questions from run to run in the simulation.

It must certainly be the case that the committals give no information about the actual questions, since if the prover knew the questions beforehand he could cheat. How can such a committal, secure against an infinitely powerful prover, be accomplished? The usual method of implementing committals is through a cryptographic assumption, which would destroy perfect zero knowledgeness. We will instead use the input graphs themselves to implement the committal. The naive approach of encoding a bit j as a random isomorphic copy of G_j does not work. To see this, consider what happens if the two input graphs are *not* isomorphic: a cheating prover sees the committals in the clear and prepares $H_1, \ldots, H_k$ accordingly.

We instead begin by having the prover send the verifier a pair of graphs (A_0, A_1) both isomorphic to G_0. If the verifier wishes to commit to a bit q he sends the prover a random isomorphic copy B of A_q. If the graphs A_0 and A_1 are indeed isomorphic, the prover will have no information as to the value of q. During the last step of the protocol, P must exhibit the isomorphisms between A_0 and G_0 and between A_1 and G_0. If he does not do so, the verifier rejects. If he does do so, the verifier is convinced that the prover could not predict his committed questions: both a committal of a 0 and a committal of a 1 are just random isomorphic copies of G_0.

We also must argue, though, that the verifier is indeed committed to his questions. That is, he cannot change his questions after seeing the graphs $H_1, \ldots, H_k$ from the prover. This is *intuitively* so because otherwise the (cheating) verifier could also find the isomorphism between the graphs G_0 and G_1, making our proof system vacuously zero knowledge. In other words, we design the simulator so that if the verifier changes his committals the simulator will extract an isomorphism between the original pair of input graphs G_0 and G_1 and thus be able to answer any questions whatsoever. This idea of using changing decommittals to extract some secret first appeared in in [1].

As this discussion might indicate, the simulator is in fact complex and contains the real heart of the argument. The protocol itself is surprisingly simple, and moreover, of just five rounds.

In discussing the protocol we will continue to use the informal terminology used here. We will refer to the graphs $(Q_1, \ldots, Q_k)$ sent by V in step (V1) as his *committals*. The $((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$ of step (V2) are his *decommittals* and $(q_1, \ldots, q_k)$ are the *questions*. The isomorphisms $(\phi_1, \ldots, \phi_k)$ sent by the prover in step (P3) are his *answers*.

We omit the formal proof that the protocol is an interactive proof system and proceed to the zero-knowledge.

5 The Simulator

We describe the simulator $S_{\widehat{V}}$ for a polynomial time ITM $\widehat{V}$.

Central to our simulation is an idea that comes from the recent protocol of Brassard, Crépeau, and Yung [1]. Very informally, they get a cheating verifier to twice decommit a single set of committals and then complete the simulation by (1) having "learned" the decommittals if they don't change, or (2) extracting some *secret* if they do change.

The basic idea of running a cheating verifier twice in order to "learn" something first appeared in the quadratic non-residuosity protocol of [5] and then in many later works; the novelty of the [1] idea is in considering changing or unchanging decommittals and in one case extracting a secret.

Unlike [1], our protocol is non-cryptographic. We approach the simulation by first describing a [1] type "double running" process and seeing how this motivates the novel and crucial idea of *modes* of operation of the simulator.

5.1 Intuition: The "Double Running" Process

On input a pair of isomorphic n node graphs (G_0, G_1), the simulator's first action is to fill the random tape of $\widehat{V}$ with coin tosses R from its own random tape. It is now prepared to run $\widehat{V}(R)$ as part of its program.

The simulator's next step is to select a pair of graphs (A_0, A_1) to correspond to the pair of graphs that the prover would send in his first message.

Let us for the moment suppose it selects them exactly as the prover would, and see where this leads. Thus, the simulator picks $\gamma_0, \gamma_1 \leftarrow S_n$ and sets $A_0 = \gamma_0(G_0)$ and $A_1 = \gamma_1(G_0)$.

Setting $\alpha_1 = (A_0, A_1)$, the simulator now writes $\alpha_1 = (A_0, A_1)$ on $\widehat{V}(R)$'s communication tape, and then runs $\widehat{V}(R)$ to get his committals $(Q_1, \ldots, Q_k)$. It then picks $\varphi'_1, \ldots, \varphi'_k \leftarrow S_n$ and sets $H'_i = \varphi'_i(G_0)$ for all $i = 1, \ldots, k$. It feeds $\widehat{V}(R)$ the message $\alpha'_2 = (H'_1, \ldots, H'_k)$ and runs it to obtain its decommittals $\beta'_2 = ((q'_1, \ldots, q'_k), (\mu'_1, \ldots, \mu'_k))$.

The message from the prover at this point would contain isomorphisms between $G_{q'_i}$ and H'_i for each i, and thus the simulator should provide such isomorphisms as well. But it is not in a position to do so (unless, of course, all the q'_i are 0). However, something has been accomplished: the simulator now knows the decommittals of $(Q_1, \ldots, Q_k)$.

The simulator takes advantage of this by selecting *new* graphs based on this information: it selects $\varphi_1, \ldots, \varphi_k \leftarrow S_n$ and set $H_i = \varphi_i(G_{q'_i})$. That is, it selects $(H_1, \ldots, H_k)$ so that it can answer the questions $(q'_1, \ldots, q'_k)$. Note that since $G_0 \cong G_1$ the graphs $(H_1, \ldots, H_k)$ have the same distribution as when the prover chooses them in his step (P2). The simulator can now certainly supply isomorphisms between H_i and $G_{q'_i}$ for each i, and, further, it can supply γ_0 mapping G_0 to A_0 and γ_1 mapping G_1 to A_1. Moreover these are again distributed as they would be for the prover. So are we done? No, because $((q'_1, \ldots, q'_k), (\mu'_1, \ldots, \mu'_k))$ might not have been $\widehat{V}(R)$'s response if it had been given $(H_1, \ldots, H_k)$ as the second message (recall this is a possibly *cheating* verifier).

However the simulator might have been lucky, so it restores $\widehat{V}(R)$ to the state where it is ready to receive the second message and feeds it $\alpha_2 = (H_1, \ldots, H_k)$. It then runs $\widehat{V}(R)$ to get $\beta_2 = ((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$. If $(q_1, \ldots, q_k) = (q'_1, \ldots, q'_k)$ then the simulator can output $(R, (\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3))$ and halt, where $\alpha_3 = ((\gamma_0, \gamma_1), (\varphi_1, \ldots, \varphi_k))$; note that this is a perfectly correct simulation. But what if there was a j such that $q_j \neq q'_j$? It is not clear what the simulator could do.

We now make the following crucial observation. Suppose that the simulator had begun by selecting (A_0, A_1) differently. Namely, it would pick $\gamma_0, \gamma_1 \leftarrow S_n$ and set $A_0 = \gamma_0(G_0)$ and $A_1 = \gamma_1(G_1)$. (Notice that since $G_0 \cong G_1$, both A_0 and A_1 are still just randomly distributed isomorphic copies of G_0 and from $\widehat{V}(R)$'s point of view there is nothing to differentiate this choice of the simulator's from the original one described above). The simulator now proceeds exactly as before. Suppose the result of the two runs again resulted in there being a j such that $q_j \neq q'_j$ (say $q_j = 0$ and $q'_j = 1$). Then the simulator is now in a good position: it can *compute* an isomorphism π between the original pair of input graphs G_0 and G_1. Namely, it sets $\pi = \gamma_0^{-1}\mu_j^{-1}\mu'_j\gamma_1$. It is then in the same position as the prover: it can compute isomorphisms ϕ_i between H_i and G_{q_i} to answer the questions[1].

Notice however that this new way of picking (A_0, A_1) is not good for the case when $\widehat{V}(R)$'s decommittals remain the same. This is because if the simulator chooses $A_0 = \gamma_0(G_0)$ and $A_1 = \gamma_1(G_1)$ he does not know an isomorphism between G_0 and A_1 which the prover's last step would require him to provide.

Section 5.2 will summarize and discuss what exactly these ideas have achieved; let us conclude here by emphasizing the nature of the "double running" process itself. This is the name we give to the above described procedure of running $\widehat{V}(R)$ once on dummy graphs $(H'_1, \ldots, H'_k)$ to get some decommittals, preparing appropriate new graphs $(H_1, \ldots, H_k)$, and then running $\widehat{V}(R)$ again and seeing whether or not his decommittals change. Note that this process is independent of the *manner* in which (A_0, A_1) were chosen and depends only on the graphs (A_0, A_1) themselves.

5.2 Overview of the Simulator Algorithm

Let us summarize what we have achieved while introducing some terminology, and sketch a rough outline of the complete simulator program.

We consider two *modes* of operation of the simulator:

Mode 0: $S_{\widehat{V}}$ selects $\gamma_0, \gamma_1 \leftarrow S_n$ and sets $A_0 = \gamma_0(G_0)$, $A_1 = \gamma_1(G_0)$.
Mode 1: $S_{\widehat{V}}$ selects $\gamma_0, \gamma_1 \leftarrow S_n$ and sets $A_0 = \gamma_0(G_0)$, $A_1 = \gamma_1(G_1)$.

Whatever the mode may be, the simulator goes through the "double running" process. This process has one of two outcomes

- The decommittals remain the same (i.e. $(q_1, \ldots, q_k) = (q'_1, \ldots, q'_k)$)
- The decommittals change (i.e. $(q_1, \ldots, q_k) \neq (q'_1, \ldots, q'_k)$)

and these are independent of the mode. As we have just seen, the outcome from the point of view of the simulator is summarized by the following table of Figure 1.

[1] A subtle point arises: although the simulator knows an isomorphism between G_1 and G_0, it is not quite in a position to terminate because it does not know a *random* isomorphism between G_0 and A_1 such as the prover would provide in his last message. We will deal with this difficulty later.

	Decommittals remain the same	Decommittals change
mode = 0	Output a simulation and halt	unsuccessful
mode = 1	unsuccessful	Extract an isomorphism $\pi \in [G_1 \to G_0]$

Figure 1: Summary of result of double running

The simulator's goal will be to reach either the top left or the bottom right points of this table. To do this it will operate in runs, where a run consists of

- Picking a mode (and (A_0, A_1) correspondingly)
- Executing the double running process.

If the outcome falls in one of the categories marked "unsuccessful" in the table, the simulator will try another run, picking the mode in some way that reflects the failure of the previous run. The obvious choice would be to switch modes every time, but for technical reasons the following turns out to be better: the simulator

- Picks the initial mode to be 1
- Switches to mode 0 for the second run if the first run was unsuccessful
- Thereafter never changes the mode (i.e. if future runs are unsuccessful, begins a new run in mode 0).

Notice that this leads to a potentially infinite number of runs; we will have to show not only that the simulator produces the right distribution but that it halts in expected polynomial time.

This is the global outline; many details remain. For example: how should the simulator terminate in the case that it extracts an isomorphism between the input graphs (see the footnote on the previous page)? What about messages of $\widehat{V}(R)$ that are "garbage" (for example, $\widehat{V}(R)$'s first message might not even be a sequence of n node graphs as the prover expects)? These issues are dealt with in the following code.

5.3 The Program for the Simulator

Let p be the polynomial bounding the computation time of $\widehat{V}$. It will be convenient for us to describe the algorithm for the simulator $S_{\widehat{V}}(\cdot)$ via another algorithm $M_{\widehat{V}}(\cdot,\cdot)$ which takes as input, in addition to a pair of isomorphic graphs (G_0, G_1), a string $R \in \{0,1\}^{p(|(G_0,G_1)|)}$ which will used as the random tape of $\widehat{V}$. Thus

Program for $S_{\widehat{V}}$:

Input: $(G_0, G_1) \in GI$

```
(1) $S_{\widehat{V}}$ sets $R$ to be the first $p(k)$ bits of its random
    tape, were $k$ is the length of $(G_0, G_1)$
    Comment This string will be used by $M_{\widehat{V}}$ as
    the random tape of $\widehat{V}$.
(2) $S_{\widehat{V}}$ runs $M_{\widehat{V}}$ on inputs $(G_0, G_1)$ and $R$.
end of Program for $S_{\widehat{V}}$
```

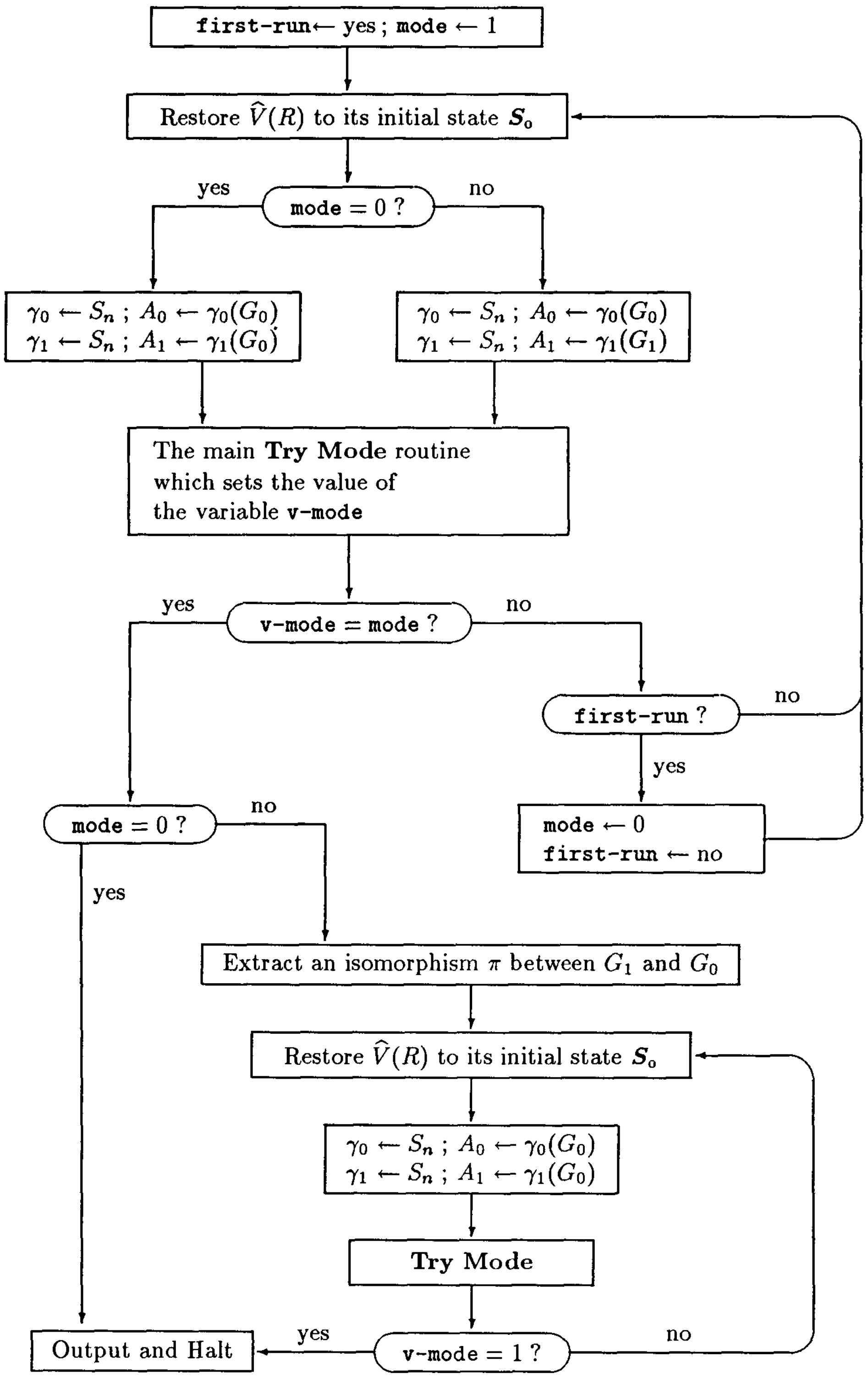
and we can concentrate on $M_{\widehat{V}}$. With a little abuse of language, we will continue to refer to $M_{\widehat{V}}$ as the "simulator".

5.4 The Machine $M_{\widehat{V}}$

The basic framework and flow of control of the algorithm $M_{\widehat{V}}$ is diagrammed in Figure 2. A detailed description follows below. **Program for $M_{\widehat{V}}$:**

Input: $(G_0, G_1) \in GI$ and $R \in \{0,1\}^{p(k)}$ where $k = |(G_0, G_1)|$

```
(M1) $M_{\widehat{V}}$ sets first-run ← yes and mode ← 1.
     Comment Note the initial mode is 1.
(M2) $M_{\widehat{V}}$ records the state $S_0$ of $\widehat{V}(R)$ at this point.
     Comment This is the initial state of $\widehat{V}(R)$.
(M3) $M_{\widehat{V}}$ restores $\widehat{V}(R)$ to the state $S_0$ recorded in
     step (M2).
     Comment Start a run.
(M4) $M_{\widehat{V}}$ picks $(A_0, A_1)$ according to the value of
     mode:
       • If mode = 0 then $\gamma_0 \leftarrow S_n$ ; $A_0 \leftarrow \gamma_0(G_0)$ ; $\gamma_1 \leftarrow S_n$ ; $A_1 \leftarrow \gamma_1(G_0)$
       • If mode = 1 then $\gamma_0 \leftarrow S_n$ ; $A_0 \leftarrow \gamma_0(G_0)$ ; $\gamma_1 \leftarrow S_n$ ; $A_1 \leftarrow \gamma_1(G_1)$
     Comment Regardless of the mode, both
     $A_0$ and $A_1$ are randomly distributed over
     $[G_0] = [G_1]$.
(M5) $M_{\widehat{V}}$ invokes the Try Mode routine on inputs
     $(A_0, A_1)$, $R$.
     Comment The Try Mode routine returns
     various variables; $M_{\widehat{V}}$ will act on the values
     of these variables.
(M6)   • If garbage = yes then
           • If first-run = yes then $M_{\widehat{V}}$ outputs $(R, (\alpha_1, \beta_1, \alpha_2', \beta_2'))$ and halts
           • Otherwise it returns to step (M3)
       • Otherwise $M_{\widehat{V}}$ proceeds to step (M7).
     Comment If $\widehat{V}(R)$'s responses were of an
     inappropriate form then the simulator halts,
     as the prover would, but only if this is the
```



first-run ← yes ; mode ← 1
Restore $\widehat{V}(R)$ to its initial state S_0
mode = 0 ?
yes
no
$\gamma_0 \leftarrow S_n$; $A_0 \leftarrow \gamma_0(G_0)$
$\gamma_1 \leftarrow S_n$; $A_1 \leftarrow \gamma_1(G_0)$
$\gamma_0 \leftarrow S_n$; $A_0 \leftarrow \gamma_0(G_0)$
$\gamma_1 \leftarrow S_n$; $A_1 \leftarrow \gamma_1(G_1)$
The main Try Mode routine which sets the value of the variable v-mode
v-mode = mode ?
yes
no
first-run ?
no
yes
mode ← 0
first-run ← no
mode = 0 ?
no
yes
Extract an isomorphism π between G_1 and G_0
Restore $\widehat{V}(R)$ to its initial state S_0
$\gamma_0 \leftarrow S_n$; $A_0 \leftarrow \gamma_0(G_0)$
$\gamma_1 \leftarrow S_n$; $A_1 \leftarrow \gamma_1(G_0)$
Try Mode
Output and Halt
yes
v-mode = 1 ?
no

Figure 2: Outline of the Program for $M_{\widehat{V}}$ on input (G_0, G_1) and R

first run. If this is not the first run then it begins an entire new run.

```
(M7)   $M_{\widehat{V}}$ branches according to v-mode:
       • If v-mode = mode then $M_{\widehat{V}}$ proceeds to step (M8)
       • Otherwise, $M_{\widehat{V}}$ branches according to
           • If first-run = yes then $M_{\widehat{V}}$ sets mode $\leftarrow$ 0 and first-run $\leftarrow$ no and goes to step (M3)
           • Otherwise it goes straight to step (M3)
         Comment $M_{\widehat{V}}$ begins a new run. If this was the first run then it switches modes; otherwise it stays in the same mode.
(M8)   $M_{\widehat{V}}$ now branches according to the mode
       • If (mode = 0) then $M_{\widehat{V}}$ sets $\alpha_3 = ((\gamma_0, \gamma_1), (\varphi_1, \ldots, \varphi_k))$, outputs
             $(R, (\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3))$
         and halts.
         Comment We are in the case where $\widehat{V}(R)$'s decommittals did not change (that is, $\vec{q} = \vec{q'}$) and the mode was 0. So $M_{\widehat{V}}$ can output a simulation.
       • Otherwise $M_{\widehat{V}}$ proceeds to step (M9)
(M9)   $M_{\widehat{V}}$ fixes a $j$ such that $q_j \neq q'_j$ and sets
       $$\pi = \begin{cases} \gamma_0^{-1} \mu_j^{-1} \mu'_j \gamma_1 & \text{if } q_j = 0 \text{ and } q'_j = 1 \\ \gamma_0^{-1} (\mu'_j)^{-1} \mu_j \gamma_1 & \text{if } q_j = 1 \text{ and } q'_j = 0. \end{cases}$$
       Comment We are in the case where $\widehat{V}(R)$'s decommittals change (that is, $\vec{q} \neq \vec{q'}$) and the mode was 1. So $M_{\widehat{V}}$ can extract an isomorphism $\pi \in [G_1 \to G_0]$ as shown. This is still not enough to terminate, however, since $M_{\widehat{V}}$ does not have a *random* isomorphism in $[G_0 \to A_1]$. So $M_{\widehat{V}}$ will start from scratch again.
(M10)  $M_{\widehat{V}}$ restores $\widehat{V}(R)$ to its initial state $S_0$
       Comment Start a run
(M11)  $M_{\widehat{V}}$ picks $(A_0, A_1)$ as $\gamma_0 \leftarrow S_n$ ; $A_0 \leftarrow \gamma_0(G_0)$ ; $\gamma_1 \leftarrow S_n$ ; $A_1 \leftarrow \gamma_1(G_0)$.
(M12)  $M_{\widehat{V}}$ invokes the Try Mode routine on inputs $(A_0, A_1)$, $R$.
       Comment The Try Mode routine returns various variables on whose values $M_{\widehat{V}}$ will act below.
(M13)  $M_{\widehat{V}}$ checks
       • If garbage = yes then $M_{\widehat{V}}$ returns to step (M10)
       • Otherwise it branches according to the value of v-mode:
           • If v-mode = 0 then $M_{\widehat{V}}$ returns to step (M10)
           • Otherwise for $i = 1, \ldots, k$ it sets
             $$\phi_i = \begin{cases} \varphi_i & \text{if } q_i = 0 \\ \varphi_i \pi & \text{if } q_i = 1. \end{cases}$$
             It sets $\alpha_3 = ((\gamma_0, \gamma_1), (\phi_1, \ldots, \phi_k))$, outputs $(R, (\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3))$ and halts.
         Comment $M_{\widehat{V}}$ is looping, waiting for v-mode to again have the value 1. The first time it detects this, it outputs a simulation and halts. Otherwise it tries Try Mode again.
```

end of Program for $M_{\widehat{V}}$

This completes the description of the main program for the simulator. We now describe the **Try Mode** routine which executes the double running process.

The **Try Mode** Routine:
Input: (A_0, A_1), R.

```
(1)  Set garbage $\leftarrow$ no.
(2)  Set $\alpha_1 = (A_0, A_1)$.
(3)  Write the message $\alpha_1$ on $\widehat{V}(R)$'s communication tape and then run $\widehat{V}(R)$ in order to obtain a message $\beta_1$.
(4)  Record the state $S_1$ of $\widehat{V}$ at this point.
(5)  Select $\varphi'_1, \ldots, \varphi'_k \leftarrow S_n$ and set $H'_i = \varphi'_i(G_0)$ for all $i = 1, \ldots, k$.
(6)  Write the message $\alpha'_2 = (H'_1, \ldots, H'_k)$ on $\widehat{V}(R)$'s communication tape and run $\widehat{V}(R)$ to get his next message $\beta'_2$.
(7)  Check that
     • $\beta_1$ is of the form $(Q_1, \ldots, Q_k)$ where each $Q_i$ is an $n$ node graph
     • $\beta'_2$ is of the form $(\vec{q'}, \vec{\mu'})$ where $\vec{q'} = (q'_1, \ldots, q'_k)$ with each $q'_i \in \{0,1\}$, $\vec{\mu'} = (\mu'_1, \ldots, \mu'_k)$ with each $\mu'_i \in S_n$
     • $Q_i = \mu'_i(A_{q'_i})$ for each $i$.
     and
     • If this is indeed the case go to step 8
     • Otherwise, set garbage $\leftarrow$ yes and go to step 13.
(8)  Restore $\widehat{V}$ to the state $S_1$ recorded in step 4.
(9)  Select $\varphi_1, \ldots, \varphi_k \leftarrow S_n$ and set $H_i = \varphi_i(G_{q'_i})$ for all $i = 1, \ldots, k$.
     Comment The graphs $H_1, \ldots, H_k$ have been selected so that the simulator knows an isomorphism between $H_i$ and $G_{q'_i}$ for each $i$ and can thus answer the questions $q'_1, \ldots, q'_k$.
```

(10) Write the message $\alpha_2 = (H_1, \ldots, H_k)$ on $\widehat{V}(R)$'s communication tape and run $\widehat{V}(R)$ to get his next message β_2.

(11) Check that

- β_2 is of the form $((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$ where $q_i \in \{0,1\}$ and $\mu_i \in S_n$ for each i
- $Q_i = \mu_i(A_{q_i})$ for each i.

and

- If this is indeed the case, set $\vec{q} = (q_1, \ldots, q_k), \vec{\mu} = (\mu_1, \ldots, \mu_k)$, and go to step 13
- Otherwise return to step 8.

Comment Again, the simulator is checking that $\widehat{V}(R)$'s message is of an appropriate form. The action taken if it is not is, however, different this time: the simulator simply loops, picking new $H_1, \ldots, H_k$ until it finds ones in response to which $\widehat{V}(R)$'s response has the appropriate form.

(12) Set

$$\texttt{v-mode} = \begin{cases} 0 & \text{if } \vec{q} = \vec{q'} \\ 1 & \text{if } \vec{q} \neq \vec{q'} \end{cases}.$$

Comment The variable v-mode records whether or not $\widehat{V}(R)$'s decommittals change.

(13) Return the following: **garbage**, **v-mode**, α_1, β_1, α_2', β_2', α_2, β_2, $(\varphi_1, \ldots, \varphi_k)$.

Comment This is the information **Try Mode** returns to the main algorithm.

end of **Try Mode** Routine

5.5 Remarks

In the case that the simulator extracts an isomorphism between G_1 and G_0 it cannot terminate directly because, as we have pointed out earlier, it cannot provide a *random* isomorphism in $[G_0 \to A_1]$ (although having π it could of course provide a particular element of this set, namely $\gamma_1\pi^{-1}$). At first glance it would appear that the solution is simply to start a new run as though in mode 0 and then use π to answer $\widehat{V}(R)$'s questions exactly as the prover would. This however would not lead to a correct distribution. What we do instead is try the "double running" until we are once again in the same situation (i.e. the bottom right corner of the table of §5.2) and *then* use π to construct appropriate respones and halt. The reason this works will be clearer when we prove the correctness of the simulator.

Figure 2 is a simplification: it does not show how "garbage" is handled.

6 Why is this Perfect Zero Knowledge?

Two things have to be shown: that the simulator generates $(P \leftrightarrow \widehat{V})(G_0, G_1)$, and that it halts in expected polynomial time.

6.1 Preliminaries

We begin with terminology, notation, and a general analysis of the algorithm $M_{\widehat{V}}$. Fix a pair of n node graphs $(G_0, G_1) \in GI$ and let k be the length of (G_0, G_1). Let $\mathcal{G} = [G_0] = [G_1]$.

A *run* of $M_{\widehat{V}}$ consists of

- Choosing (A_0, A_1)
- Executing **Try Mode**

Thus the passage from step (M3) to step (M5) constitutes a run, as does the passage from step (M10) to step (M12). We call a run of the former kind a *primary* run and a run of the latter kind a *secondary* run. There are three possibilities for any given run:

- The run is aborted via "garbage" being encountered at step 7 of **Try Mode**
- The run is good for mode 0
- The run is good for mode 1

where a run is *good for mode i* if **Try Mode** outputs $\texttt{v-mode} = i$. We denote the probabilities of these events by g, h_0, h_1 respectively. Note that $g + h_0 + h_1 = 1$.

Next, some terminology:

Definition 6.1 Suppose $\alpha_1 = (A_0, A_1)$ and $\beta_1 = (Q_1, \ldots, Q_k)$ where each A_i and each Q_i is an n node graph. We say that a string β_2 is (α_1, β_1)-*good* if it is of the form

$$(\vec{q}, \vec{\mu}) = ((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$$

and

- $q_i \in \{0,1\}$ and $\mu_i \in S_n$ for each i
- $Q_i = \mu_i(A_{q_i})$ for each i.

Definition 6.2 We say that C is *complete* if it is of the form $(\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3)$ where

- $\alpha_1 = (A_0, A_1) \in \mathcal{G} \times \mathcal{G}$
- $\beta_1 = (Q_1, \ldots, Q_k)$ where each Q_i is an n-node graph
- $\alpha_2 = (H_1, \ldots, H_k) \in \mathcal{G}^k$
- $\beta_2 = ((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$ is (α_1, β_1)-good
- $\alpha_3 = ((\gamma_0, \gamma_1), (\phi_1, \ldots, \phi_k))$ where $\gamma_0(G_0) = A_0$, $\gamma_1(G_0) = A_1$, and $\phi_i(G_{q_i}) = H_i$ for each i.

Definition 6.3 We say that C is *incomplete* if it is of the form $(\alpha_1, \beta_1, \alpha_2, \beta_2)$ where

- $\alpha_1 \in \mathcal{G} \times \mathcal{G}$

- $\alpha_2 \in \mathcal{G}^k$
- Either β_1 is not of the form $(Q_1, \ldots, Q_k)$ with each Q_i an n node graph, or β_2 is not (α_1, β_1)-good

It is easy to see that

Lemma 6.1 Any $C \in [(P \leftrightarrow \widehat{V}(R))(G_0, G_1)]$ is either complete or incomplete.

Proof: Omitted □

Definition 6.4 Suppose $\beta_2' = (\vec{q'}, \vec{\mu'})$ and $\beta_2 = (\vec{q}, \vec{\mu})$ are (α_1, β_1)-good. We say that β_2' is $(\alpha_1, \beta_1, \beta_2, 0)$-*good* if $\vec{q} = \vec{q'}$. Otherwise we say that it is $(\alpha_1, \beta_1, \beta_2, 1)$-*good.*

6.2 Correctness

Theorem 6.1 Suppose $(G_0, G_1) \in GI$. Then

$$S_{\widehat{V}}(G_0, G_1) = (P \leftrightarrow \widehat{V})(G_0, G_1) .$$

Proof: Follows from the program for the simulator (§5.3) and Lemma 6.2. □

Lemma 6.2 Suppose $(G_0, G_1) \in GI$ and $R \in \{0,1\}^{p(k)}$, where $k = |(G_0, G_1)|$). Then

$$M_{\widehat{V}}((G_0, G_1), R) = (P \leftrightarrow \widehat{V}(R))(G_0, G_1) .$$

Proof: It suffices to show that

$$\mathbf{P}_{M_{\widehat{V}}((G_0,G_1),R)}(C) = \mathbf{P}_{(P \leftrightarrow \widehat{V}(R))(G_0,G_1)}(C)$$

for any $C \in [(P \leftrightarrow \widehat{V}(R))(G_0, G_1)]$. We consider separately the cases of C being complete and C being incomplete.

First suppose C is complete. It is thus of the form $(\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3)$ where

- $\alpha_1 = (A_0, A_1)$
- $\beta_1 = (Q_1, \ldots, Q_k)$
- $\alpha_2 = (H_1, \ldots, H_k)$
- $\beta_2 = ((q_1, \ldots, q_k), (\mu_1, \ldots, \mu_k))$
- $\alpha_3 = ((\gamma_0, \gamma_1), (\phi_1, \ldots, \phi_k))$

are as described in Definition 6.2 ($\mathcal{G} = [G_0] = [G_1]$). It is easy to see that $\mathbf{P}_{(P \leftrightarrow \widehat{V}(R))(G_0,G_1)}(C)$ equals

$$\mathbf{P}_{\mathcal{G}\times\mathcal{G}}(A_0, A_1)\mathbf{P}_{\mathcal{G}^k}(H_1, \ldots, H_k)\mathbf{P}_{[G_0 \to A_0]\times[G_0 \to A_1]}(\gamma_0, \gamma_1) \cdot$$
$$\mathbf{P}_{[G_{q_1} \to H_1]\times\cdots\times[G_{q_k} \to H_k]}(\phi_1, \ldots, \phi_k)$$
$$= U(\mathcal{G}^2) \cdot U(\mathcal{G}^k) \cdot U([G_0 \to A_0] \times [G_0 \to A_1]) \cdot$$
$$U([G_{q_1} \to H_1] \times \cdots \times [G_{q_k} \to H_k]) ;$$

this follows from the way the prover selects his messages (§4.1) and Lemmas 3.1 and 3.2. We will compute $\mathbf{P}_{M_{\widehat{V}}((G_0,G_1),R)}(C)$ and show that it has this same value. Let

$$s_i = \mathbf{P}(\widehat{V}(R; (G_0, G_1), \alpha_1\beta_1\overline{\alpha}_2) \text{ is } (\alpha_1, \beta_1, \beta_2, i)\text{-good} : \overline{\alpha}_2 \leftarrow \mathcal{G}^k)$$
$$f = \mathbf{P}(\widehat{V}(R; (G_0, G_1), \alpha_1\beta_1\overline{\alpha}_2) \text{ is not } (\alpha_1, \beta_1)\text{-good} : \overline{\alpha}_2 \leftarrow \mathcal{G}^k) ,$$

($i \in \{0,1\}$). Observe that

$$f + s_0 + s_1 = 1 . \quad (1)$$

Consider a primary run of $M_{\widehat{V}}$ when it is in mode 0. This run will result in the output $C = (\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3)$ if

- $M_{\widehat{V}}$ gets $\gamma_0, \gamma_1, A_0, A_1$ in step (M4)
- The message β_2' that $M_{\widehat{V}}$ gets in step 6 of **Try Mode** is $(\alpha_1, \beta_1, \beta_2, 0)$-good.
- $M_{\widehat{V}}$ gets $\varphi_1 = \phi_1, \ldots, \varphi_k = \phi_k$ and $H_1, \ldots, H_k$ in step 9 of **Try Mode.**

The probability of this (which we will denote by p) is

$$\mathbf{P}_{\mathcal{G}\times\mathcal{G}}(\alpha_1) \cdot s_0 \cdot \mathbf{P}_{\mathcal{G}^k}(\alpha_2) \cdot \mathbf{P}_{[G_0 \to A_0]\times[G_0 \to A_1]}(\gamma_0, \gamma_1)$$
$$\cdot \mathbf{P}_{[G_{q_1} \to H_1]\times\cdots\times[G_{q_k} \to H_k]}(\varphi_1, \ldots, \varphi_k) \cdot \sum_{j\geq 0} f^j$$
$$= U(\mathcal{G}^2) \cdot s_0 \cdot U(\mathcal{G}^k) \cdot U([G_0 \to A_0] \times [G_0 \to A_1])$$
$$\cdot U([G_{q_1} \to H_1] \times \cdots \times [G_{q_k} \to H_k]) \cdot \sum_{j\geq 0} f^j .$$

(Note that $f \neq 1$ since we are assuming that C is complete and $C \in [(P \leftrightarrow \widehat{V}(R))(G_0, G_1)]$).

On the other hand consider a secondary run of $M_{\widehat{V}}$. Let π be the isomorphism that was obtained in step (M9). This run will result in the output $(\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3)$ if

- $M_{\widehat{V}}$ gets $\gamma_0, \gamma_1, A_0, A_1$ in step (M11)
- The message β_2' that $M_{\widehat{V}}$ gets in step 6 of **Try Mode** is $(\alpha_1, \beta_1, \beta_2, 1)$-good.
- $M_{\widehat{V}}$ gets
$$\varphi_i = \begin{cases} \phi_i & \text{if } q_i = 0 \\ \phi_i\pi^{-1} & \text{if } q_i = 1. \end{cases}$$
($i = 1, \ldots, k$) and $H_1, \ldots, H_k$ in step 9 of **Try Mode.**

The probability of this (which we will denote by q) is

$$\mathbf{P}_{\mathcal{G}\times\mathcal{G}}(\alpha_1) \cdot s_1 \cdot \mathbf{P}_{\mathcal{G}^k}(\alpha_2) \cdot \mathbf{P}_{[G_0 \to A_0]\times[G_0 \to A_1]}(\gamma_0, \gamma_1)$$
$$\cdot \mathbf{P}_{[G_{q_1} \to H_1]\times\cdots\times[G_{q_k} \to H_k]}(\varphi_1, \ldots, \varphi_k) \cdot \sum_{j\geq 0} f^j$$
$$= U(\mathcal{G}^2) \cdot s_1 \cdot U(\mathcal{G}^k) \cdot U([G_0 \to A_0] \times [G_0 \to A_1])$$
$$\cdot U([G_{q_1} \to H_1] \times \cdots \times [G_{q_k} \to H_k]) \cdot \sum_{j\geq 0} f^j .$$

How could $M_{\widehat{V}}$ output C? It could do so either through a primary run or through a secondary run. There are two alternatives:

- The first (primary) run is good for mode 0 (which happens with probability h_0): In this case, $M_{\hat{V}}$ initiates new runs while staying in mode 0. Each time it has a probability p of outputting C and a probability $1 - h_0$ of beginning a new run

- The first (primary) run is good for mode 1 (which happens with probability h_1): In this case, $M_{\hat{V}}$ initiates new runs until it again finds a run good for mode 1. Each time it has a probability q of outputting C and a probability $1 - h_1$ of beginning a new run.

The probability $\mathbf{P}_{M_{\hat{V}}((G_0,G_1),R)}(C)$ that $M_{\hat{V}}$ outputs C can thus be written as

$$\begin{cases} h_0 \sum_{j\geq 0}(1-h_0)^j p & \text{if } h_0 \neq 0 \text{ and } h_1 = 0 \\ h_1 \sum_{j\geq 0}(1-h_1)^j q & \text{if } h_0 = 0 \text{ and } h_1 \neq 0 \\ h_0 \sum_{j\geq 0}(1-h_0)^j p + h_1 \sum_{j\geq 0}(1-h_1)^j q & \text{otherwise} \end{cases}$$

$$= \begin{cases} p & \text{if } h_0 \neq 0 \text{ and } h_1 = 0 \\ q & \text{if } h_0 = 0 \text{ and } h_1 \neq 0 \\ p+q & \text{otherwise} \end{cases}$$

If $h_i = 0$ then $s_i = 0$ ($i \in \{0,1\}$) so this is always $p+q$. Substituting the expressions for p and q derived above we get

$$U(\mathcal{G}^2)\cdot(s_0+s_1)\cdot U(\mathcal{G}^k)\cdot U([G_0 \rightarrow A_0]\times[G_0 \rightarrow A_1]) \cdot U([G_{q_1} \rightarrow H_1]\times\cdots\times[G_{q_k} \rightarrow H_k])\cdot \sum_{j\geq 0} f^j \ .$$

Now using equation 1 we get

$$U(\mathcal{G}^2)\cdot U(\mathcal{G}^k)\cdot U([G_0 \rightarrow A_0]\times[G_0 \rightarrow A_1])\cdot U([G_{q_1} \rightarrow H_1]\times\cdots\times[G_{q_k} \rightarrow H_k])$$

and this is just $\mathbf{P}_{(P\leftrightarrow\hat{V}(R))(G_0,G_1)}(C)$.

Next we consider the case that C is incomplete. This is considerably simpler since there is only one point at which $M_{\hat{V}}$ would output an incomplete conversation. This is in the first run in step 7 of **Try Mode**. Thus the probability of the incomplete conversation C is just the probability that $M_{\hat{V}}$, in the first run, gets α_1 when it picks (A_0, A_1) in step (M4) and gets α_2 when it picks $(H'_1, \ldots, H'_k)$ in step 5 of **Try Mode**. This probability is

$$\mathbf{P}_{M_{\hat{V}}((G_0,G_1),R)}(C) = \mathbf{P}_{\mathcal{G}\times\mathcal{G}}(\alpha_1)\cdot\mathbf{P}_{\mathcal{G}^k}(\alpha_2) \ ,$$

which again equals $\mathbf{P}_{(P\leftrightarrow\hat{V}(R))(G_0,G_1)}(C)$. □

6.3 Termination

The idea is that all the potentially infinite loops are being generated according to a common rule: there is some event E which happens with probability p such that

- If E occurs the first time then we halt
- If $\overline{E}$ occurs the first time then we wait for $\overline{E}$ to occur again.

It can be shown that this procedure halts in expected 2 tries regardless of the value of p.

A full proof that the simulator halts in expected polynomial time will appear in the final paper.

Acknowledgements

We thank Gilles Brassard, Oded Goldreich, and R. Venkatesan.

References

[1] Brassard, G., C. Crépeau, and M. Yung, "Everything in NP can be Argued in Perfect Zero Knowledge in a Bounded Number of Rounds," ICALP 89.

[2] Fortnow, L, "The Complexity of Perfect Zero Knowledge," STOC 87.

[3] Goldreich, O., and H. Krawczyk, "On the Composition of Zero Knowledge Proof Systems," ICALP 90.

[4] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing but their Validity", Technical Report #498, Technion, 1988; preliminary version in FOCS 86.

[5] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proofs," *SIAM J. Comput.*, vol. 18, No. 1, pp. 186-208, February 1989; preliminary version in STOC 85.

[6] Goldwasser, S., S. Micali, and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM J. Comput.*, vol. 17, No. 2, pp. 281-308, April 1988.

[7] Tompa, M. and H. Woll, "Random Self Reducibility and Zero Knowledge Interactive Proofs of Possession of Information," FOCS 87.