Zero-Knowledge Multi-Prover Interactive Proofs

Nan Yang

A Thesis in the Department of
Computer Science and Software Engineering

Presented in Partial Fulfillment of
the Requirements for the Degree of
Master of Computer Science
at Concordia University
Montreal, Quebec, Canada

April 2013

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Nan Yang

Entitled: Zero-Knowledge Multi-Prover Interactive Proofs

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. H. Harutyunyan

_____ Examiner
Dr. S. Bergler

_____ Examiner
Dr. V. Chvatal

_____ Supervisor
Dr. C. Crepeau

_____ Supervisor
Dr. D. Ford

Approved by _____
Chair of Department or Graduate Program Director

_____
Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Date _____

Abstract

Single-prover interactive proofs can recognize **PSPACE**; if certain complexity assumptions are made, they can do so in zero-knowledge. Generalizing to multiple non-communicating provers extends this class to **NEXP**, and at the same time removes the complexity assumption needed for zero-knowledge.

However, it was recently discovered that the non-communication condition might be insufficient to guarantee soundness. The provers can form joint randomness through non-local computation without communicating. This could break protocols that rely on the statistical independence of the provers.

In this work, we analyze multi-prover interactive proofs under the constraint of statistical isolation, introduced in [5], which prohibits non-local computation. We show that there exists perfect zero-knowledge proofs for **NEXP** under statistical isolation.

Acknowledgements

I would like to thank my supervisors David Ford and Claude Crépeau for their unwavering support. Without their guidance, I would have been forever lost in the infinite maze that is mathematics. I would also like to thank my good friend Alexandra Ortan for helping me to $\cdot i \times t$.

This thesis is dedicated to my mother, who put her own dream aside so I could live mine.

Contents

Chapter 1

Introduction

The theory of interactive proofs sits at the intersection of cryptography, computational complexity and information theory. It is a study of those languages L for which there exists a protocol proving $x \in L$ by a computationally unbounded and possibly malicious *prover* to a polynomial-time *verifier* beyond reasonable doubt. The set of all such languages is called **IP**.

There are two main results in this field. The first is that **IP** = **PSPACE** [9]. That is, those languages which require a polynomial amount of space to recognize (with no restrictions on time) are exactly those which are accepted by **IP**. The second is that any protocol of **IP** can be executed in *zero-knowledge*, which informally means that a potentially malicious verifier will learn nothing from the interaction with the prover other than the single bit of information '$x \in L$', where L is the language in question.

However, an assumption is needed to achieve zero-knowledge: the existence of one-way functions. The cryptography side of interactive proofs' family abhors unfounded assumptions, no matter how many it must live with in practice.

In order to remove this assumption, [1] changed the setting of interactive proofs from a single prover to multiple, non-communicating provers. The class of languages accepted under this setting is called **MIP**. They showed that every language in **NP** has a zero-knowledge proof without complexity assumptions. Later works showed that **MIP** = **NEXP**, and that it is possible to recognize the whole of **MIP** in zero-knowledge without complexity assumptions as well.

However, there is a hidden fatal flaw in those protocols. While the assumption is that the provers are not able to communicate through dedicated channels of their own, it was not shown that those protocols would prohibit the provers to communicate through the verifier. Indeed, the zero-

knowledge multi-prover protocol in [1] *requires* that the provers communicate in order to fend off man-in-the-middle attacks by the verifier. No other existing zero-knowledge multi-prover protocol addresses this problem.

In addition, it was later discovered that the 'no-communication' condition that is imposed on these provers might be insufficient to guarantee security; under this condition, the provers can form joint randomness through some non-local processes. These processes do not allow communication, but they nevertheless could be used to break the security of some protocols which relies on the no-communication assumption alone. The verifier could even be used to implement these non-local computations without breaking the no-communication assumption! Clearly, the no-communication assumption is not strong enough.

How then do we guarantee that the verifier is not used by the provers to communicate or perform non-local computations? This is the question that we address in this work.

Our Contribution

We apply the concept of statistical isolation from [5] to form strongly isolating verifiers. Such verifiers can be guaranteed to resist any attempt by the provers to communicate or compute non-locally.

We show that an existing (non-zero-knowledge) multi-prover protocol for oracle-3-SAT, a **NEXP**-complete language, can be implemented by a strongly isolating verifier; however, existing zero-knowledge multi-prover protocols for oracle-3-SAT cannot. We construct a zero-knowledge protocol for oracle-3-SAT that can be implemented by a strongly isolating verifier.

Chapter 2

Mathematical Preliminaries

2.1 Asymptotic Notations

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. We define the following notations:

- $f = O(g)$ if there exists $k, C \in \mathbb{R}$ such that $x > k \Rightarrow |f(x)| < C|g(x)|$
- $f = o(g)$ if $\lim_{x\to\infty} f(x)/g(x) = 0$
- $f = \Omega(g)$ if there exists $k, C \in \mathbb{R}$ such that $x > k \Rightarrow f(x) > Cg(x)$
- $f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$
- $f \sim g$ if $lim_{x\to\infty} f(x)/g(x) = 1$
- f is a negligible function if $f(x) = o(1/\mathbf{poly}(x))$ for every polynomial **poly**

The same asymptotic notations can be defined for functions whose domain or range is a subset of $\mathbb{R}$ analogously.

2.2 Strings and Languages

Any finite set S is called an *alphabet* in the context of language. The exact nature of the elements of S is irrelevant; we only care that they are distinguishable.

We denote by $S^* = \{(s_1, \ldots, s_i)_{i\leq n} : n \in \mathbb{N}, s_i \in S\}$ the set of all finite sequences (or *strings*) of elements of S. Any subset $L \subset S^*$ is called a *language*. The length of a string $x \in L$, denoted $|x|$, is the length of the sequence. Note that S^* is an infinite set unless $S = \emptyset$.

In this work we will mainly be interested in the alphabet $\{0,1\}$ and languages $L \subset \{0,1\}^*$.

Similarly, *states* and *symbols* simply are distinguishable members of finite sets, in the context of Turing machines, to be defined below.

2.3 Turing Machines

A *Turing machine* is a tuple $(Q, \Gamma, b, \Sigma, q_0, q_F, \delta)$ where

- Q is a finite set of states,
- Γ is an alphabet,
- $b \in \Gamma$ is the blank symbol,
- $\Sigma \subset \Gamma$ is a set of input symbols,
- $q_0 \in Q$ is the initial state,
- q_F is the halted state,
- $\delta : Q \backslash \{q_F\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.

Intuitively, a Turing machine operates on an infinitely long tape, consisting on infinitely many cells, each of which contains one symbol from the set Γ. The *input* of a Turing machine is the initial configuration of the tape, on the condition that only finitely many cells contain a non-blank symbol.

A Turing machine reads a single cell at a time and, according to the transition function δ, overwrites the symbol in that cell, and then move left or move right, and enters a new state. The Turing machine *halts* if it enters the state q_F.

The *output* of a Turing machine is the content of the tape after it halts. If it does not halt, then its output is undefined.

We will use the shorthand $M(x) = y$ to denote that upon executing M with input x, it halts and outputs y.

A more formal treatment of Turing machines will be counter-productive here; it can be found in any elementary computability textbook. The proofs for the following results can be found in them.

Theorem 2.1. *Turing machines with multiple tapes are equivalent to those with a single tape.*

Theorem 2.2. *Turing machines with a single tape that is infinite in one direction only are equivalent to those with two-way infinite tapes.*

A turing machine is *probabilistic* if it has an additional infinitely long *random tape* which, when the machine is initialized, is filled with uniformly random symbols from some alphabet which will be clear from context, or explicitly defined.

2.4 Recognition of Languages by Turing Machines

Let $L \subset S^*$ be a language. If there exists a Turing machine M with $S \cup \{\textsf{accept}, \textsf{reject}\}$ as its alphabet such that for every input $x \in S^*$,

- the machine halts after finitely many steps,
- if $x \in L$ then it outputs **accept**,
- if $x \notin L$ then it outputs **reject**,

then we say that L is *computable* and that M *decides* L.

2.5 Circuits

A *circuit* is a finite simple directed acyclic graph where the edges (or *wires*) take on a value from some alphabet and for every vertex there corresponds a function which takes the values of the in-edges and outputs values for the out-edges. Vertices of in-degree zero are *input gates*. Vertices of out-degree zero are *output gates*.

Any particular circuit can only take inputs of a fixed length since there is a fixed number of input gates. Therefore, to represent a function whose domain might contain strings of (possibly infinitely many) different lengths, we use families of circuits $\{C_i\}_{i \in I}$, where each C_i is a circuit, and I is an index set.

A family of circuits $\{C_i\}_{i \in I}$ is *f-uniform* if there exists a Turing machine M such that for all $i \in I$, $M(i)$ produces an appropriately encoded description of C_i, taking less than $f(i)$ steps.

2.6 Complexity of Turing Machines and Circuits

Let M be a Turing machine, x be an input on which M halts, and $|x|$ be the length of x. Let $t(|x|)$ denote the number of steps it takes for M to halt on x. We call t the *complexity function* of M. Very often t cannot be defined explicitly by elementary functions, so we will bound it asymptotically with simpler functions.

Let C be a circuit. The *circuit size* of C is its number of vertices. The *circuit depth* is the length of its longest directed path.

We will adopt the usual convention in deeming Turing machines of polynomial complexity and polynomial-uniform circuits to be those that are *efficient* or *feasible*.

2.7 Complexity Classes

The following common complexity classes will be discussed in this work. They contain languages L with the following properties.

- **P** – L can be recognized in polynomial-time.
- **P/poly** – L can be recognized by a polynomially-sized family of circuits.
- **BPP** – L can be recognized in probabilistic polynomial-time with an error for soundness and completeness of at most $1/3$.
- **NP** – Every $x \in L$ has a *witness* w_x such that $\{(x, w_x) : x \in L\} \in \mathbf{P}$.
- co-**NP** – Every $x \notin L$ has a *witness* w_x such that $\{(x, w_x) : x \notin L\} \in \mathbf{P}$.
- **PSPACE** – There exists a polynomial p and a Turing machine M such that, for all $x \in L$, M accepts x while not using more than $p(|x|)$ cells of M's tape.
- **EXP** – L can be recognized by a Turing machine in exponential-time.
- **NEXP** – Every $x \in L$ has a *witness* w_x such that $\{(x, w_x) : x \in L\} \in \mathbf{EXP}$.

Other complexity classes will be defined if the need arises.

2.8 Probability and Entropy

Let S be a finite set. Let $X : S \to \mathbb{R}$ be a function. We call S the *sample space* and X a *discrete random variable*. We will only deal with discrete random variables in this work, so we will omit the word 'discrete' from now on.

Associated with each random variable X is a probability mass function, $f_X : \mathbb{R} \to [0, 1]$ which defines the probability that X takes on a particular value. That is $f_X(x) = \mathbf{Pr}(X = x)$. This function satisfies

$$\sum_{x \in \mathbb{R}} f_X(x) = 1.$$

The Shannon entropy H of a random variable describes the amount of uncertainty a random variable contains. We define it as

$$H(X) = -\sum_{x \in S} f_X(x) \log_2 f_X(x).$$

In this work, we will be looking at the case where $S = \{0,1\}^n$, or the entropy of bit-strings. In the special case where $n = 1$, $H(X)$ represents the amount of uncertainly we have about the value of a particular bit. When $H(X) = 0$, we know for certain its value, whereas when $H(X) = 1$, we have no idea what it is. For example, if we use X to describe the outcome of a coin-toss, then $H(X)$ describes the bias of the coin, from a coin which only lands on one side ($H(X) = 0$) to a perfectly fair coin ($H(X) = 1$) and everything in between.

A more detailed description of Shannon's entropy function can be found in any information theory textbook.

Chapter 3

Single-Prover Interactive Proofs

3.1 Introduction

Traditionally, *proofs* are static. You should be convinced of a proof by reading it without help from its author. *The proof should speak for itself* is the old adage. We can relate two complexity classes to this analogy. The class **P** can be thought of as those proofs which can be efficiently produced by an author, and the class **NP** as the class of proofs which can be efficiently verified by a reader. Whether $\mathbf{P} = \mathbf{NP}$ is an open problem at the time of this writing, but it is widely believed that we are able to verify much more efficiently than we are able to prove ($\mathbf{P} \subsetneq \mathbf{NP}$). Intuition and experience tells us that this is the case, but we may yet be surprised.

The natural question which arises is that if we *can* interact with the author, do we benefit over having only a static proof? The question is not one of pedagogy, but of computational power. If we call the class of languages we can accept by interacting with a prover **IP**, do we have $\mathbf{NP} \subsetneq \mathbf{IP}$?

Before we answer that, we must decide how to model this interaction. Informally, we can think of **IP** as languages having protocols for two parties, a probabilistic polynomial-time *verifier*, and a potentially malicious all-powerful *prover*. The prover must try and convince the verifier that $x \in L$ given a language L. The protocols must with high probability allow the prover to convince the verifier if indeed $x \in L$ (completeness), and with high probability will fail to convince the verifier if that is not the case (soundness), despite the difference in computational power.

Returning to the question of whether interaction with a prover is beneficial, the answer is 'yes' with an asterisk. It turns out that if we do not demand perfect soundness, then $\mathbf{IP} = \mathbf{PSPACE}$, therefore this question is reduced to answering the $\mathbf{NP} \stackrel{?}{=} \mathbf{PSPACE}$ question, which is open at this

time. Although it is widely believed that **NP** $\neq$ **PSPACE**, unfortunately as of now we once again have only intuition and not concrete proof.

One potential problem with this model is that a verifier, taking advantage of an interaction with an all-powerful prover, could try and learn something other than what was intended, and in a cryptographic setting this may not be acceptable. The solution is the concept of *zero-knowledge proofs*, in which the verifier can learn the one bit of knowledge '$x \in L$' and nothing else.

For most languages in **PSPACE**, all currently known single-prover zero-knowledge proofs depend on the existence of one-way functions. In fact, the existence of a single-prover zero-knowledge proof for **NP**-complete languages which does not use one-way functions would imply the collapse of the polynomial hierarchy. Thus is very unlikely that we can remove this complexity assumption in general, in the single prover case. All existing exceptions, for example graph-isomorphism, are not known to be **NP**-complete.

However, this assumption is not needed in the multi-prover model, which we will discuss in the next chapter.

3.2 Definitions

Definition 3.1. An *interactive Turing machine* is a deterministic Turing machine with

- a read-only input tape,
- a read-only auxiliary input tape,
- a read-write work tape,
- a write-only output tape,
- a read-only incoming communication tape,
- a write-only outgoing communication tape,
- a read-write switch tape,
- and a read-only random tape (also called random coins).

Each interactive Turing machine has an *identity*, which is an unary string $\sigma \in \{1\}^*$. If σ is equal to the contents of its switch tape, then the machine is said to be *active*. Otherwise it is *idle*. While idle, the contents in all of its tapes are not modified.

A set of k interactive Turing machines are *linked* if they have distinct identities, their input and switch tapes coincide, and every pair of the machines M_i, M_j have their own incoming and outgoing communication tapes $\mathbf{in}_i, \mathbf{out}_i, \mathbf{in}_j, \mathbf{out}_j$ such that $\mathbf{in}_i = \mathbf{out}_j$ and $\mathbf{in}_j = \mathbf{out}_i$. All other tapes are distinct.

A *joint computation* of a set of k linked interactive Turing machines, on a common input x, is a sequence of k-tuples representing the local configurations of all machines.

If a machine halts while the contents of the switch tape equals to its identity, then we say that all machines have halted. The *outputs* of these machines are the set of their individual outputs.

Let $S \subseteq \{1, \ldots, k\}$. We adopt the notation $\mathbf{out}_S \langle A_1(a_1), \ldots, A_k(a_k) \rangle (x)$ to denote the random variable associated with the local outputs of the machines indexed by S on common input x, with individual auxiliary inputs $a_1, \ldots, a_k$; unless otherwise specified, the randomization is uniformly and independently over each bit on the random tapes of A_i.

When no S is specified, we will write $\langle A_1(a_1), \ldots, A_k(a_k) \rangle (x)$ as the joint outputs of all machines, randomized over any of their random tapes. If no auxiliary input is specified, then the empty string is used.

Note that in some texts these are called 'interactive Turing machines with auxiliary inputs', but we will not make that distinction since auxiliary inputs are necessary for our constructions later on.

For now, we will only need the case where $k = 2$.

Definition 3.2. A pair of interactive machines (P, V) is an *interactive proof system for a language* L if V is probabilistic polynomial-time in the length of the input x and that

- (Completeness) $\forall (x \in L)\ \exists y\ \forall z \in \{0,1\}^* : \mathbf{Pr}(\mathbf{out}_V \langle P(y), V(z) \rangle (x) = \mathbf{accept}) \geq 2/3$,
- (Soundness) $\forall (x \notin L, P^*, y \in \{0,1\}^*, z \in \{0,1\}^*) : \mathbf{Pr}(\mathbf{out}_V \langle P^*(y), V(z) \rangle (x) = \mathbf{accept}) \leq 1/3$, where P^* is an interactive Turing machine and the probabilities are over V's private random tape.

Define **IP** to be the set of languages having interactive proof systems.

The constant $2/3$ in the completeness definition turns out to be redundant. It is possible to replace it with 1.

Theorem 3.1. *Every language in* **IP** *has an interactive protocol with perfect completeness (an honest prover will never fail to convince the verifier).*

The proof of the above theorem follows from an analysis of the protocol for TQBF found in [8].

3.3 Arthur-Merlin Protocols

In the previous section, the soundness of interactive proofs relies on the fact that the verifier has access to a private random tape which the prover does not – recall that soundness is randomized over these random coins. In fact, this private random tape completely determines the verifier. It would seem counterintuitive that it is possible to design interactive proofs where the only things the verifier sends to the prover are its private coins, but it is. We call such protocols *public coin* or *Arthur-Merlin* protocols.

Definition 3.3. Define the class $\mathbf{AM}[\mathsf{poly}]$ exactly as $\mathbf{IP}$ above, except that the verifier's messages are restricted to be contents of his random tape.

Note that even in this case, the verifier is not expected to give all of its random coins to the prover *at once*.

Theorem 3.2. $\mathbf{AM}[\mathsf{poly}] = \mathbf{IP} = \mathbf{PSPACE}$

Theorem 3.3. *Every language in* $\mathbf{AM}[\mathsf{poly}]$ *has a public-coin protocol with perfect completeness.*

The proofs of the above theorems depend on a generalization of the sumcheck protocol, construction 4.1, made to work on the arithmetization of true quantified boolean formulas, which is a $\mathbf{PSPACE}$-complete language.

For details, see [8], section 8.2.

3.4 Bit-Commitment Schemes

We take a slight detour here to introduce bit-commitment schemes, which are crucial for constructing zero-knowledge proofs later. Suppose that a sender, Alice, wishes to send a message to a receiver, Bob; however, Alice does not want Bob to read the message immediately, so Alice encrypts the message with a private key of her choosing, and then when the time comes to reveal what the message was, she sends the key.

However, most encryption schemes do not offer the guarantee that a message can be decrypted in only one way. Alice could, in principle, encrypt the message m under key k ($\mathbf{enc}_k(m)$) and send it to Bob. Later, she could find some other key k' such that $m \neq \mathbf{dec}_{k'}(\mathbf{enc}_k(m))$.

We say that most encryption schemes, although *concealing* (Bob cannot know what the message is until Alice gives him the key), are not *binding* (Alice can change her mind about what the message was after she sent it).

Bit-commitment schemes, on the other hand, have those two properties. They can be thought of as Alice carving a message into a piece of stone, locking it in an unbreakable safe, then sending the safe to Bob. Only when it is time to reveal what the message was will the combination of the safe be sent.

The two conditions of bit-commitment schemes, binding and concealing, can have different strengths attributed to them: *perfect*, *statistical*, or *computational*. For our purposes, we will only discuss *statistically binding* and *computationally concealing* bit-commitment in the single-prover case.

We will adopt the notation $[a]$ for the commitment of a bit a.

Definition 3.4. A *bit-commitment scheme* is a two-phase, two-party protocol between two probabilistic polynomial-time machines, A (sender Alice) and B (receiver Bob), as follows:

- Alice and Bob agree on a security parameter n. Alice has a bit b that she wishes to commit.
- (commit phase) Alice takes b and her auxiliary information (including random coins) r_A and exchanges messages with Bob, who outputs his auxiliary tape r_B and the transcript T of the interaction.
- (unveil phase) Alice sends (b, r_A) to Bob, who accepts if and only if simulating their commit phase with inputs (b, r_A, r_B) produces the right transcript T.

The protocol must satisfy

- (statistically binding) for all but negligible fraction of r_B there are no random coins for the sender r'_A such that, when Bob simulates the commit phase with $(\overline{b}, r'_A, r_B)$ he produces the transcript T,
- (computationally concealing) for every family of polynomial-time algorithm B^* interacting with A, the ensembles $\{(A(0), B^*)(1^n)\}_{n>0}$ and $\{(A(1), B^*)(1^n)\}_{n>0}$ are computationally indistinguishable.

We will now construct a bit-commitment scheme based on the existence of one-way functions.

Lemma 3.1. *If one way functions exist, then there exist pseudorandom generators* $G : \{0,1\}^* \to \{0,1\}^*$ *such that* $|G(x)| = 3|x|$ *for all* $x \in \{0,1\}^*$.

For a proof of the above lemma, see [12].

Construction 3.1. Let n be the security parameter. To commit a bit, Alice selects $r \in \{0,1\}^{3n}$ uniformly at random and sends it to Bob, who then selects $s \in \{0,1\}^n$ uniformly at random. Suppose that Alice wishes to commit a bit b. If $b = 0$ then she sends $G(s)$, and if $b = 1$ she sends $G(s) \oplus r$.

To unveil a bit, Alice sends s. Bob then computes whether he received $G(s)$ or $G(s) \oplus r$, corresponding to a commitment of 0 and 1, respectively. □

Theorem 3.4. *The above construction is a statistically binding and computationally concealing bit-commitment scheme.*

For a detailed proof of the above theorem, see [7], section 4.4.1.3. We will sketch it here. To see that the construction is statistically binding, without loss of generality suppose that Alice had sent $G(s)$, in order for her to change her mind there must exist s' such that $G(s) = G(s') \oplus r$, which is exponentially unlikely. It is computationally concealing because of Bob is too weak to invert the one-way function, and thus the pseudorandom generator.

In the case where an all-prover prover is committing, it is necessary that we use a scheme that is either statistically or perfectly binding for obvious reasons. The reader might wonder why we did not construct a perfectly concealing and perfectly binding bit-commitment scheme. Unfortunately, with only two parties (a prover and a verifier) and no additional assumptions, it can be shown that this is impossible to achieve.

3.5 Bit-Commitment Equality

Given two commitments from Alice to Bob, $[a]$ and $[b]$, of bits a and b respectively, Alice would like to show Bob that $a \oplus b = c$ for some c without revealing what the commitments are. Specifically, if $c = 0$, then the committed bits are equal.

We show that this is possible by constructing a bit-commitment scheme with equality from an arbitrary bit-commitment scheme.

Construction 3.2. Bit-commitment equality from any existing bit-commitment scheme.

1. Alice and Bob agree on a security parameter k.

2. (Commit with equality) Alice prepares a commitment of bit a by committing random bit pairs $([a_{iL}], [a_{iR}])_{i \leq k}$ of random bits a_{iL}, a_{iR} such that for all $i \leq k$, $a_{iL} \oplus a_{iR} = a$. She does the same for bit b as $(b_{iL}, b_{iR})_{i \leq k}$.

3. (Equality check) Suppose that Alice wishes to show that $a = b$. For each i, she computes $d_i = a_{iL} \oplus b_{iL}$. She sends $\{d_i\}$ to Bob. Bob tosses k coins $\{c_i\}$ and sends them to Alice. For each i, if $b_1 = 0$ then Alice unveils $[a_{iL}], [b_{iL}]$, else she unveils $[a_{iR}], [b_{iR}]$. Bob checks whether for all i, $a_{iL} \oplus b_{iL} = d_i$ or $a_{iR} \oplus b_{iR} = d_i$. If any of these fails, reject. Otherwise accept. □

A proof of its security can be found in [10].

As constructed, the commitments are not reusable. That is, once a proof of equality (or inequality) between two bits has been executed, neither commitments can be used to show its equality (or inequality) to any other commitments. The above construction can be slightly modified so that the commitments are reusable (attributed to Steven Rudich in [10]).

It is easy to see that bit-commitment equality retains the concealing and binding strengths of the scheme on which it is based; also, since it is only polynomially slower, we will assume henceforth that all bit-commitments are done with reusable equality.

3.6 Committed Circuit Evaluation

Suppose that Alice and Bob share a boolean gate G, and that Alice has committed bits $[a]$, $[b]$ and $[c]$ to Bob. Alice wishes to convince Bob that $c = G(a, b)$ without revealing their values. We show that this is possible with the help of bit-commitment equality from the previous section.

Recall that a boolean gate can be thought of as a truth table $[(i, j, G(i, j))]_{0 \le i,j \le 1}$.

Construction 3.3. Committed gate evaluation.

1. Alice and Bob agree on a security parameter k.
2. Alice takes the truth table for gate G, applies a random permutation to its rows, and commits this permuted truth table to Bob.
3. Bob flips a coin d. If $d = 0$ then Bob asks Alice to unveil the commitments of the permuted truth table for G. If $d = 1$ then Bob asks Alice to prove to him, using bit-commitment equality, that the triplet $([a], [b], [c])$ is equal to one of the rows of the committed, permuted truth table.
4. Steps 2 and 3 are repeated k times. □

A proof of security for the above construction can be found in [10].

In order to evaluate an entire circuit in committed form, Alice commits a triple $([a], [b], [G(a, b)])$ for each gate. She and Bob then topologically sort the circuit, and then evaluate the circuit gate-by-gate along the sorted order. When the input $[a]$ of a gate is the output of a previous gate $[G(x, y)]$,

Alice convinces Bob using bit-commitment equality that $a = G(x, y)$. Finally, when the output gate is shown to have been evaluated correctly, Alice can then choose to unveil the output, if she desires.

In particular, a prover P and a verifier V can agree on a circuit of some other verifier Λ to be evaluated in committed form. V provides a source of randomness that P is bound to, and P can show V that 'I am doing everything Λ would have done correctly, in encrypted form'. This is the basic idea behind the technique of turning interactive proofs into zero-knowledge proofs. This is the idea of 'public-coin interactive proofs' which will be discussed later.

We also need for V to "provide a source of randomness" to P. This will be the topic of the following section.

3.7 Two-Party Coin-Toss

Suppose that Alice is on the phone with Bob and they wish to toss a coin. They do not trust each other, they cannot meet up in person to physically toss a coin, and they have no mutually trusted friends to help them. Is it possible for them to agree on a coin flip?

With the help of bit-commitment, it is possible. The protocol is simple:

Construction 3.4. Two-party coin-toss (also called coin-toss in a well).

1. Alice selects a random bit a and commits it to Bob as $[a]$.
2. Bob selects a random bit b and sends it to Alice in the clear.
3. Alice unveils commitment $[a]$ to Bob.
4. They output $a \oplus b$. □

If at least one of them chooses a random bit, $a \oplus b$ will be a random bit: since bit-commitment is concealing, Bob does not know a, so he cannot control $a \oplus b$; and since bit-commitment is binding, Alice cannot change the unveiling of a, so she cannot control $a \oplus b$. The only way for $a \oplus b$ to be non-random is for both of them to collude; they cannot unilaterally predict or control the outcome.

If Alice wishes that the outcome be unknown to Bob, they can execute the following instead.

Construction 3.5. Oblivious two-party coin-toss.

1. Alice selects a random bit a and commits it to Bob as $[a]$.
2. Bob selects a random bit b and sends it to Alice in the clear.

3. Alice commits a bit $[c]$ to Bob and convinces him that $[a] \oplus b = [c]$ using circuit evaluation in committed form. □

This way, Alice can commit a random bit to Bob who can be certain that the bit was the outcome of a uniformly random event, even if he does not know what it is. This is how a verifier can 'provide a source of randomness' to a prover.

3.8 Zero-Knowledge Proofs

So far, our approach to interactive proofs has been from a complexity-theoretic perspective. Now we will do so from a cryptographic perspective. Previously, the provers were the potentially malicious parties. Now, we will develop a scenario in which the verifier can behave badly.

When executing an interactive proof protocol, the ultimate goal of the prover is to convince the verifier that '$x \in L$'. Suppose that there is a trusted third party, who interacts in the verifier's place with the prover. This third party can then come to the same conclusion as to whether '$x \in L$' as the verifier. But now the verifier needs only to know this one bit of 'knowledge' from the trusted third party.

We would like a formal way of capturing this ideal case of having a trusted third party, thus we need to formally define what it means for the verifier to not acquire 'knowledge'.

A natural place to start is to define it in terms of information theory. However, this is too coarse. Information theory deals with the question *what can we know about x and how certain do we know it?* When x is given, everything there is to know about it is defined with certainty, including whether $x \in L$. It does not address the fact that the verifier is bounded in computational power, and that some things about x are harder to compute than others; indeed, it does not take into account the very fact that the verifier is interacting with an all-powerful prover because it is too weak to recognize L.

Intuitively then, we want to say that the verifier has not learned any new 'knowledge' from an interaction if it cannot compute anything that it could not have from before the interaction. This is the idea of *simulation*. If a conversation between a prover and a verifier can be simulated by a machine with no more computational power than the verifier (in this case, an expected polynomial-time *simulator*) without interacting with the prover, then the verifier cannot have learned anything from the conversation.

Incidentally, if the reader finds that malicious verifiers are too artificial of a motivation for studying zero-knowledge, we can rephrase it in the terms of an optimization problem – namely,

minimizing knowledge leak. We are sure that the reader agrees that this is an interesting question in itself.

We define two types of zero-knowledge, corresponding to two levels of 'security'.

Definition 3.5. Let (P, V) be an interactive proof system. The *view* of the verifier on a common input x, denoted $\mathbf{view}(P, V, x)$, is the random variable of the sequence of V's outputs during its interaction with P, over the random coins of V.

Definition 3.6. Let (P, V) be an interactive proof system for a language L. We say that P is a *zero-knowledge interactive Turing machine* if the following holds, depending on the type of zero-knowledge considered.

For every probabilistic polynomial-time verifier V^*, there is an expected polynomial-time *simulator* S such that for all $x \in L$:

- (perfect zero-knowledge) $S(x)$ and $\mathbf{view}(P, V^*, x)$ are identically distributed.
- (computational zero-knowledge) $S(x)$ and $\mathbf{view}(P, V^*, x)$ are computationally indistinguishable: for every probabilistic polynomial-time distinguisher D, there exists N such that

$$|x| > N \Rightarrow |\mathbf{Pr}(D(x, S(x)) = \mathbf{accept}) - \mathbf{Pr}(D(x, \mathbf{view}(P, V^*, x)) = \mathbf{accept})| < \mathbf{negl}(|x|).$$

Where we think of $S(x)$ as a random variable over its random coins.

We say that (P, V) is a *zero-knowledge interactive proof system for* L if P is a zero-knowledge interactive Turing machine and (P, V) satisfies completeness and soundness. □

There is another type of zero-knowledge, called *statistical zero-knowledge*. We will not consider it, since in the single-prover case we have computational zero-knowledge, and in the multi-prover case we can achieve perfect zero-knowledge.

Some texts require the simulator to be probabilistic polynomial-time as opposed to expected polynomial-time. The two definitions are equivalent in that the resulting classes of languages having zero-knowledge proofs are identical.

An implicit assumption made in the definition of zero-knowledge is that if the verifier deviates from the protocol, then the prover will abort. This accounts for the *arbitrary* behavior of V^* that a simulator must respond to.

3.9 Zero-Knowledge for NP

Because of the properties of **NP**-completeness and polynomial-time reductions, we will only need to exhibit a zero-knowledge protocol for a single **NP**-complete language. We choose the *graph 3-coloring* problem.

Definition 3.7. Let $G = (V, E)$ be a simple, finite, non-directed graph. A *3-coloring* of G is a function $f : V \to \{1, 2, 3\}$ such that for all $(a, b) \in E$, $f(a) \neq f(b)$. The *graph 3-coloring* problem is to decide, given G, whether such a function exists.

Theorem 3.5. *Graph-3-coloring is* **NP***-complete.*

The above theorem is a well-known result. For details, see [8].

Thus if we wish to prove in zero-knowledge the membership of an **NP**-complete language, we will simply need to reduce it in polynomial-time to an instance of graph 3-coloring, and execute the zero-knowledge protocol for that instance.

Construction 3.6. The following is a zero-knowledge graph 3-coloring protocol between prover Alice and verifier Bob.

1. They receive a graph $G = (V, E)$ and a security parameter n as a common input. We assume that Alice knows a 3-coloring function $f : V \to \{1, 2, 3\}$.

2. Alice chooses uniformly at random a permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$. Alice commits the values $\pi(f(v))$ to Bob for each vertex $v \in V$.

3. Bob chooses an edge $(a, b) \in E$ uniformly at random and asks Alice to unveil the commitments for a and b. Bob rejects if they have the same value.

4. Steps 2 and 3 are repeated n times. Bob accepts if he did not reject at any point of this repetition. □

Theorem 3.6. *The above construction is a zero-knowledge protocol for graph 3-coloring.*

The intuition behind the above theorem is that if the graph is not 3-colorable, then there will be an edge with two vertices committed with the same color. The probability that the verifier will ask for such an edge to be unveil is at least $1/|E|$, during a particular repetition, thus the probability that a malicious prover will get away with cheating decreases exponentially in the number of runs. For details, see [7], section 4.4.2.

The strength of zero-knowledge depends on the strength of the concealing property of the bit-commitment protocol used in step 2. In our case, it is a computationally concealing bit-commitment (construction 3.1), and therefore it is computational zero-knowledge.

3.10 Zero-Knowledge for PSPACE

Now we will extend the **NP** zero-knowledge protocol to **PSPACE**.

Recall that $\mathbf{AM}[\mathsf{poly}] = \mathbf{IP}$ (section 3.3), and therefore we can convert any interactive proof to a public-coin protocol with perfect completeness, consisting of a polynomial-time machine A (Arthur) and an all-powerful machine M (Merlin). The transcripts after executing such a protocol are of the form $(a_1, m_1, \ldots, a_k, m_k)$, where a_i are the coin flips sent by Arthur and m_i are Merlin's replies, in order.

The key observation to make here is that if we replaced m_i with a commitment $[m_i]$ in the transcript, then the statement 'this is a valid transcript where Arthur accepts' is in **NP**, since if it were true, then we can prove it in polynomial-time by unveiling the commitments $[m_i]$ and simulating Arthur to check whether the transcript is valid (recall that Arthur is a deterministic polynomial-time machine if we fix his random coins, and the protocol has perfect completeness, so a true statement will always be accepted).

Construction 3.7. Given $L \in \mathbf{PSPACE}$, prover P wishes to prove in zero-knowledge to verifier V that $x \in L$ for some string $x \in \{0,1\}^*$.

1. P and V agree on a public-coin protocol with perfect completeness (A, M) for L.
2. Whenever Arthur tosses coins, P and V execute a committed two-party coin-toss protocol where P commits to V the outcomes of the coin-tosses.
3. Whenever Merlin responds, P commits its answers to V.
4. When the Arthur-Merlin protocol is finished, V reduces the transcript to an instance of graph-3-coloring G such that the transcript is valid if and only if there is a 3-coloring of G. V sends the graph to P.
5. P verifies that G is indeed one that is 3-colorable if and only if the transcript is valid.
6. P and V execute a graph-3-coloring zero-knowledge protocol for G. V accepts or rejects accordingly. □

There is an alternative way of constructing zero-knowledge protocols for **PSPACE**. If we instead have a committed-with-equality transcript $([a_1], [m_1], \ldots, [a_k], [m_k])$, then a prover who knows how to unveil the commitments can evaluate Arthur's circuit with those inputs then send the committed *circuit* to a verifier. The prover then convinces the verifier that the circuit is a valid evaluation in committed form.

Construction 3.8. Given $L \in$ **PSPACE**, prover P wishes to prove in zero-knowledge to verifier V that $x \in L$ for some string $x \in \{0,1\}^*$.

1. P and V agree on a public-coin protocol with perfect completeness (A, M) for L.
2. Whenever Arthur tosses coins, P and V execute a committed two-party coin-toss protocol where P commits to V the outcomes of the coin-tosses.
3. Whenever Merlin responds, P commits its answers to V.
4. When the Arthur-Merlin protocol is finished, P evaluates Arthur's circuit with the coins tossed earlier and his answers to V. P then commits the values of every gate to V.
5. P proves to V that the committed circuit is valid using circuit evaluation in committed form.
6. P unveils the output of the circuit. V accepts or rejects accordingly. □

Once again the bit-commitment is computationally concealing, hence the strength of zero-knowledge is computational. For details on how to construct simulators for the above constructions, see [7], theorem 4.4.12.

Chapter 4

Multi-Prover Interactive Proofs

4.1 Introduction

The soundness of single-prover interactive proofs holds against computationally unbounded provers. This is the worst possible case in terms of computational power for an adversary. Even so, we can catch any such cheating prover for all languages in **PSPACE**.

Suppose that the verifier now has access to two or more provers who are prohibited from communicating with each other. Have we gained anything? The intuition is that of a detective interrogating two suspects in isolated rooms. Although the suspects may have a joint alibi, under separate interrogation the detective might uncover inconsistencies. Thus, having extra non-communicating provers may actually weaken their individual ability to cheat without losing the generality and simplicity of computational unboundedness. This multi-prover model was introduced in [1].

This turns out to be a powerful setup. There are two main results. The first is that the class of languages having multi-prover interactive proofs (**MIP**) is the same as non-deterministic exponential-time (**NEXP**), a complexity class that is believed to be much larger than **PSPACE**. The second is that it is possible for any language in **MIP** to be accepted in *perfect* zero-knowledge.

We will present the proof that **MIP** = **NEXP**, then discuss how the no-communication assumption may be insufficient to guarantee soundness, and finally look at how this problem can be addressed.

4.2 Definitions

Definition 4.1. $(P_1, \ldots, P_k, V)$ is called a *k-prover interactive protocol* if the following holds.

1. They form a set of $k+1$ linked interactive Turing machines, but without communication tapes between P_i and P_j if $i \neq j$.

2. Each P_i is all powerful, and V is probabilistic polynomial-time.

3. There exists an infinitely long read-only random tape accessible by all P_i, but not V.

The P_i are *provers* and V is the *verifier*.

Definition 4.2. A language L has a k-prover interactive proof if there exists a probabilistic polynomial-time interactive machine V such that:

1. (completeness) $\exists(P_1, \ldots, P_k)$ such that $(P_1, \ldots, P_k, V)$ is a k-prover interactive protocol and $\forall(x \in L)\ \exists(y_1, \ldots, y_k) \in \{0,1\}^*\ \forall z \in \{0,1\}^*\ \mathbf{Pr}(\mathbf{out}_V \langle P_1(y_1), \ldots, P_k(y_k), V(z)\rangle = \mathbf{accept}) \geq 2/3$.

2. (soundness) $\forall(P_1, \ldots, P_k)$ such that $(P_1, \ldots, P_k, V)$ is a k-prover interactive protocol, $\forall(x \notin L)\ \forall(y_1, \ldots, y_k) \in \{0,1\}^*\ \forall z \in \{0,1\}^*\ \mathbf{Pr}(\mathbf{out}_V \langle P_1(y_1), \ldots, P_k(y_k), V(z)\rangle = \mathbf{accept}) \leq 1/3$.

Definition 4.3. Let $\mathbf{MIP}_k$ denote languages which have a k-prover interactive proof. Let $\mathbf{MIP} = \bigcup_{k\geq 1} \mathbf{MIP}_k$.

4.3 MIP = NEXP

We will prove containment in the two directions, thereby proving equality.

Theorem 4.1. $\mathbf{MIP} \subseteq \mathbf{NEXP}$.

As shown in [2], we obtain this upper bound by proving equivalence between multi-prover proofs and oracle machines. Unlike provers, oracles have no memory, although they are not necessarily deterministic. Therefore an oracle's future answers does not depend on its prior ones. They simply represent the black-box computation of some function in a single step.

Definition 4.4. Let M^H be a probabilistic polynomial-time machine with access to oracle H. A language L is accepted by M if and only if

1. $\forall x \in L \exists$ oracle H such that $\mathbf{Pr}[M^H(x) = \mathbf{accept}] > 2/3$,

2. $\forall x \notin L \forall$ oracle H', $\mathbf{Pr}[M^{H'}(x) = \mathbf{accept}] < 1/3$,

where $n = |x|$.

Lemma 4.1. *A language L is accepted by a probabilistic polynomial-time oracle machine if and only if it is accepted by a multi-prover interactive protocol.*

We sketch a proof of theorem 4.1. Using the above lemma, given $L \in \mathbf{MIP}$, there exists a probabilistic oracle machine M which accepts L. A non-deterministic exponential-time machine can be created to accept L by guessing the oracle's answers to all questions whose length is bounded by a polynomial, then go through all possible random coins of M, tallying the number of accepting instances. Details can be found in [2].

Theorem 4.2. $\mathbf{NEXP} \subseteq \mathbf{MIP}$.

We will present the strategy that was used in [3] to prove this theorem. We will first review a single-prover protocol for languages in co-**NP**. Then we will present a **NEXP**-complete language which can be accepted by an *oracle/prover hybrid protocol*. Finally, we will combine this with arguments from [3] which shows that the oracle can be replaced by a prover. Proofs of the following lemmas and the proof of correctness of the following constructions can be found in [3].

Definition 4.5. Let $\phi : \{0,1\}^m \to \{0,1\}$ be a Boolean function. An *arithmetization* of ϕ is a polynomial $f(x_1, \ldots, x_m) \in \mathbb{Q}[X_1, \ldots, X_m]$ such that for all $z \in \{0,1\}^m$, $\phi(z) = 0 \Leftrightarrow f(z) = 0$.

Equivalently, the $\phi(z) = 0 \Leftrightarrow f(z) = 0$ condition can be replaced with $\phi(z) = 1 \Leftrightarrow f(z) = 0$.

Lemma 4.2. *An arithmetization of a Boolean function $\phi : \{0,1\}^m \to \{0,1\}$ can be constructed in polynomial-time.*

The fact that a Boolean function is a *tautology*, that the function that always evaluates to true regardless of input, can be expressed by the equation

$$\sum_{x_1=0}^{1} \cdots \sum_{x_m=0}^{1} f(x_1, \ldots, x_m)^2 = 0$$

For a suitably chosen polynomial f. We present the following protocol called the *sumcheck protocol* which shows that tautologies can be proven by an interactive proof.

Construction 4.1. Sumcheck protocol.

Let $\phi(x_1, \ldots, x_m)$ be the 3-CNF formula which the prover P is trying to show to be a tautology to a verifier V.

1. V takes ϕ and computes its arithmetization f according to 4.2 and sends it to P.

2. V and P agree on a set $I \subset \mathbb{Q}$ of size at least $2dm$ where d is the degree of f.

3. V assigns $b_0 = 0$, which is supposed to be equal to the sum

$$\sum_{x_1=0}^{1} \cdots \sum_{x_m=0}^{1} f(x_1, \ldots, x_m)^2 = 0$$

4. $i \leftarrow 1$.

5. P sends the coefficients of the univariate polynomial in x,

$$g_i(x) = h(r_1, \ldots, r_{i-1}, x) = \sum_{x_{i+1}=0}^{1} \cdots \sum_{x_m=0}^{1} f(r_1, \ldots, r_{i-1}, x, x_{i+1}, \ldots, x_m)^2$$

6. V checks whether $b_{i-1} = g_i(0) + g_i(1)$. If not, abort.

7. V chooses a random $r_i \in I$, computes $b_i = g_i(r_i)$ and sends r_i to P.

8. If $i \leq m$ then $i \leftarrow i + 1$ and go to step 4.

9. V checks whether $b_m = f(r_1, \ldots, r_m)^2$. □

The idea behind this protocol is that the two different polynomials of degree n will agree on at most n points, so that if the prover cheats, it would have to cheat during every round. At the end (step 9), the polynomial which was adaptively constructed by the prover must agree with f on some input. If the prover has cheated at some point, there is a high chance that the final results would disagree. For a proof of its correctness, see [8].

Note that the last step does not require interaction. V merely substitutes the random values it has chosen during the protocol's loop into the explicit polynomial f^2 and checks whether it is equal to the last value stated by P.

Definition 4.6. Let $r, s > 0$ be integers. Let z, b_1, b_2, b_3 be strings of variables, where $|z| = r$ and $|b_i| = s$. Let $B(z, b_1, b_2, b_3, t_1, t_2, t_3)$ be a Boolean formula in $r+3s+3$ variables. A Boolean function $A : \{0,1\}^s \to \{0,1\}$ is a *3-satisfying oracle* for B if

$$B(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 1$$

for every string z, b_1, b_2, b_3.

B is *oracle-3-satisfiable* if such a function A exists.

The *Oracle-3-SAT* problem (B, r, s) asks whether a Boolean formula B is oracle-3-satisfiable, where r and s denote the lengths of z and b_i, as above.

Oracle-3-SAT problems are similar to demonstrating that a Boolean function is a tautology, except that the last three inputs are reserved for the oracle. Not surprisingly, we will adapt the sumcheck protocol for oracle-3-SAT later.

Lemma 4.3. *Oracle-3-SAT is* **NEXP***-complete.*

Lemma 4.4. *Let $L \in$* **NEXP** *and $x \in L$. Then it is possible to compute in polynomial-time an instance of oracle-3-SAT (B, r, s) such that B is oracle-3-satisfiable if and only if $x \in L$.*

Lemma 4.5. *Given an instance of oracle-3-SAT (B, r, s), it is possible to compute in polynomial-time an integer polynomial g with the same variable symbols as B such that a function $A : \{0,1\}^s \to \mathbb{Q}$ is a 3-satisfying oracle if and only if*

$$\sum_{z \in \{0,1\}^r, b_i \in \{0,1\}^s} g(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0$$

Definition 4.7. A polynomial $f \in \mathbb{Q}[X_1, \ldots, X_m]$ is *multilinear* if the degrees of all variables are ≤ 1.

Lemma 4.6. *Let $A : \{0,1\}^s \to \mathbb{Q}$. Then A has a unique multilinear extension $\hat{A} : \mathbb{Q}^s \to \mathbb{Q}$. Furthermore,*

- *If A only takes on integer values then so does $\hat{A}$ when restricted to $\mathbb{Z}$.*
- *Let $I = \{0, \ldots, N-1\}$. Then for any $x \in I^s$ we have that $|\hat{A}(x)| < (2N)^s$.*

Note that $\hat{A}$ might contain exponentially many summands.

Lemma 4.7. *Suppose that an oracle H stores a function A. Then there is a polynomial-time oracle machine interacting with H that will always accept if A is multilinear, but reject with probability exponentially close to 1 in some security parameter if A is not.*

Oracles differ from provers in that they do not have memory, so they do not let their current answer depend on any previous answers. When we ask a prover to simulate an oracle, we are asking it to answer non-adaptively. We can catch a cheating prover by asking one of the queries to a second prover who we have not communicated with before, then amplify this probability by repeating. A prover which is only asked a single question can be thought of as an oracle. Details can be found in [3].

Now we have the necessary preliminaries to construct a multi-prover protocol for **NEXP**.

Construction 4.2. A multi-prover protocol (P_1, P_2, V) for $L \in \textbf{NEXP}$. Given $x \in L$ as common input.

1. V computes an instance of oracle-3-sat (B, r, s) if and only if $x \in L$ as per lemma 4.4.

2. By lemma 4.5, V converts (B, r, s) into a polynomial g in $r + 3s + 3$ variables with integer coefficients such that (B, r, s) is satisfied if and only if there exists a Boolean function A' whose multilinear extension A is such that

$$\sum_{z \in \{0,1\}^r, b_i \in \{0,1\}^s} g(z, b_1, b_2, b_3, A(b_1), A(b_2), A(b_3)) = 0$$

   The provers do the same.

3. (multilinearity test) V asks P_1 to simulate an oracle storing the function A. V executes the protocol from lemma 4.7 to decide whether to accept A as multilinear. If this test fails, abort the entire protocol. Let $Q_1, \ldots, Q_k$ be V's questions during this phase.

4. (sumcheck with oracle) V and P_1 execute the sumcheck protocol from construction 4.1 to check that the equation from step 2 holds. Note that the values of A are supplied by P_1 during this step. Let $Q_{k+1}, \ldots, Q_{k+3}$ be V's questions during this phase.

5. (non-adaptiveness test) V chooses uniformly at random an i such that $1 \leq i \leq k + 3$ and asks Q_i to P_2. If P_2's answer differs from that of P_1, reject. Otherwise accept.

Steps 1 to 5 are repeated until the completeness and soundness probabilities are $\geq 2/3$ and $\leq 1/3$ respectively. □

Proposition 4.1. *The above protocol has perfect completeness.*

The idea behind the proof of theorem 4.2 is that given $L \in \textbf{NEXP}$, we reduce it to an instance of oracle-3-SAT, then we check it with multiple provers using construction 4.2. Details can be found in [3].

Chapter 5

Non-Locality

5.1 Introduction

The previous chapter introduced the idea of multi-prover protocols whose soundness rests on the fact that the provers cannot communicate, a fact which was used to force non-adaptive behavior upon them.

There are two problems with existing multi-prover protocols. The first is that although the provers have no direct way of communication, they can still do so through V if V allows it, inadvertently or not. So although it is possible that a particular protocol requires V to courier a message across provers for whatever reason, in most cases this is probably not desired. The existing analyses of multi-prover protocols do not rule out that the protocols, and hence V, can be exploited by the provers to communicate.

The second problem is that the provers can technically share a 'non-communicating non-local box', a trusted party implementing a multiparty computation that does not allow communication. This, in turn, may be used to break some protocols which can be 'proven secure' with just the no-communication assumption. While this may be very contrived, it is allowed. We give a simple example below.

Construction 5.1. A non-local box known as a PR-box.

1. P_1 inputs bit a. P_2 inputs bit b.
2. P_1 receives a uniformly random bit s. P_2 receives $t = s \oplus (a * b)$. □

Details about this construction can be found in [11].

Since s is uniformly random and independent of a and b, the PR-box does not allow the provers to communicate. However, with its help, the provers can establish the joint input-output correlation $((a, b), (s, t))$ where $a \oplus b = s * t$ with certainty, whereas without the PR-box they can only do so with 3/4 chance, when the inputs are uniformly random. Therefore if the soundness of a protocol depends on this gap of 1/4 in probability, provers with PR-boxes will break it.

As a concrete example, we invite the reader to skip ahead and look at construction 6.1. With the help of a PR-box, we will now break it in the same way as in [5]. For the ith bit of the string, P_2 inputs into the PR-box the ith bit of the verifier's random string $a = z_i$, and obtains an ouput s, which is sent to the verifier. P_1 discloses some random bit x. To unveil a bit e, P_1 inputs into the PR-box $b = x \oplus e$ and obtains $t = s \oplus (z_i * b)$, which is sent to the verifier, along with e.

If $b = 0$, then the verifier will find that $t = s$, which is the correct ith bit for unveiling. If $b = 1$, then $t = s \oplus z_i$, again the correct ith bit for unveiling. Do this for every bit of the unveil string. P_1 can thus unveil either way, at will. This is not good.

An example of a violation of the soundness of non-zero-knowledge multi-prover protocols, based on quantum entanglement (a special kind of non-locality), can be found in [4].

While it is possible to remedy these particular cases with an additional assumption such as *the provers do not share entanglement* or *the provers do not share non-local boxes*, in general, analyses of multi-prover protocols do not take into account of any possible non-locality which might be inadvertently established by the verifier, a much weaker condition than the verifier accidentally allowing communication. We can even plainly ask the verifier to execute some non-local box (such as the PR-box, or simulating entanglement) with the provers, with the verifier acting as the trusted party, and it would not be a violation of the no-communication assumption.

It is also not possible to guarantee soundness when many multi-prover protocols are composed, for the same reasons. Specifically, it is not possible to guarantee that repeating a primitive protocol, which normally would amplify soundness, would in fact not break it by inadvertently allowing the provers to communicate or establish non-locality.

Determining the exact class of languages that can be accepted by a multi-prover protocol when the provers have access to non-locality is an open problem which we will not discuss further. Trivially, it must lie somewhere between **PSPACE** and **NEXP**.

Rather, we will discuss how to guarantee that the provers are unable to exploit the verifier into allowing communication or establishing non-local correlations in the first place, and the languages that a verifier can accept normally and in zero-knowledge under such constraints.

5.2 Isolation

Suppose that the provers have only the verifier to talk to, and not some other third parties or non-local boxes. We want to make sure that the verifier does not unwillingly establish non-local correlation between provers. We define as in [5] the notion of *isolation*.

Definition 5.1. Let $(P_1, \ldots, P_k, V)$ be a k-prover interactive protocol. V is *strongly isolating* if every message sent or received by the verifier is tagged with a tag $T \in \mathcal{P}(\{1, \ldots, k\})$ as follows:

- Every message computed without any previous messages is tagged with $T = \{1, \ldots, k\}$.
- Every message received from prover P_i is tagged with $T = \{i\}$.
- Every message computed from previous messages is tagged with the intersection of the tags of those messages.

A message (m, T) can be sent to prover P_i only if $i \in T$. In particular, if $T = \emptyset$ then the message cannot be sent to anyone.

We call this notion *strong* isolation in order to differentiate it from another notion of isolation (one based on information theory) found in [5].

Theorem 5.1. *If V is a strongly isolating verifier then the provers cannot establish non-local correlations using V. In particular, they cannot use V to communicate with each other.*

The intuition behind the fact that a strongly isolating verifier cannot let provers communicate is that a message tagged with T cannot contain any information about messages tagged with $\{1, \ldots, k\}$ T. Thus through an inductive argument, if a message is sent to a prover, that message cannot contain information that the recipient does not already possess.

As for non-local computation, without loss of generality suppose that P_1 sends the verifier a bit a_1, and the verifier replies with a uniformly random bit b_1. The verifier receives P_2's message a_2 and replies b_2. For there to be any correlation between b_2 and a_1 or b_2 and b_1, the verifier must have done some kind of computation involving all three bits. But then such a message would have been tagged with the empty tag, and would not have been sent to P_2.

For details, see [5].

Definition 5.2. A multi-prover protocol is *strongly isolating* if it can be implemented with a strongly isolating verifier.

5.3 NEXP Revisited

We will take another look at the multi-prover protocol for **NEXP** introduced in the previous chapter.

Theorem 5.2. *Protocol 4.2 can be implemented with a strongly isolating verifier.*

Proof. Let us look at the construction step by step and tag all of the messages sent by V.

1. V does not send any messages in this step.
2. V does not send any messages in this step.
3. The queries q_i sent to P_2 are generated from V's random tape only, thus are tagged $(q_i, \{1, 2, 3\})$ and are cleared to be sent to P_2. One such query q_k is selected at random, thus is still tagged as $(q_k, \{1, 2, 3\})$. This query is cleared to be sent to P_3.
4. This step involves looking at construction 4.2 and noting that all of the messages sent by V to P_1, including b_1, b_2 and b_3, are random numbers generated independent from any previous messages. Those messages are thus tagged with $\{1, 2, 3\}$.
5. b_1, b_2, b_3 are cleared from the previous step to be sent to P_2 or P_3.

All messages have been accounted for, thus V is strongly isolating. □

The fact that this protocol is already strongly isolating is fortuitous, but unsurprising. P_2 only needs to answer a single question regarding the multilinear function. The difficulty will arise in the context of zero-knowledge, in which case the provers must somehow jointly fend off the verifier's attempts at gaining knowledge. Already they cannot communicate, but now we must place the additional constraint of strong isolation upon them.

Chapter 6

Zero-Knowledge MIP

6.1 Introduction

The difficulty with turning construction 4.2 into a zero-knowledge proof under isolation lies in the fact that P_2 must answer an encrypted question in order to show non-adaptivity. P_1 must be the one who encrypts it. V must make sure that it is encrypted correctly, and that the question is not an attempt at communication. P_2 must make sure the question was not altered by V in a man-in-the-middle attack. Finally, P_2 must answer the question in committed form.

The existing solution is essentially the following. All the parties agree on some random hash function which will be used by the provers for authentication, and P_1 will commit a random string which will be used as a one-time pad between it and P_2. P_1 takes an existing oracle question of V's choosing and commits the hash. P_1 shows V obliviously that the hashing was done correctly. P_1 then encrypts the question with the one-time pad and shows that the encryption was done with the committed random string. This guarantees V that the ciphertext is that of the question, and not some attempt at communication, while the hash confirms to P_2 that the question genuine. V sends P_2 the encrypted question and the hash, who decrypts and authenticates it before answering.

The reason for this elaborate dance between the parties is that the verifier must ask P_2 a question which it has already asked P_1 in order for the protocol to remain zero-knowledge. Suppose that the verifier asks one prover question x, and the other question y, then a polynomial-time simulator would have to store a possibly exponentially large function A in order to simulate the transcript. This is because it must simulate acceptance or rejection by the verifier based on whether $A(x) \stackrel{?}{=} A(y)$, which it cannot know.

The problem with the above solution is that the message which must be couriered by V from P_1 to P_2 will be tagged with P_1's tag. Thus it cannot be sent if the verifier is strongly isolating.

Our solution is simple. We will not ask V to courier any message. Instead, we will ask the provers to encrypt the answer with the question as the encryption key, and we will let the verifier ask a non-adaptive test question of its choosing to each prover. If the same question is asked, then the verifier will receive the same random string. Otherwise, it will receive two independent and uniformly random strings.

6.2 Preliminaries

Zero-knowledge for multi-prover interactive proofs is defined in the same way as the single-prover case: as the ability to generate the view of any verifier in polynomial time with the same distribution as the actual conversations with honest provers.

Definition 6.1. Let $(P_1, \ldots, P_k, V)$ be a multi-prover interactive proof system. The *view* of the verifier on an input x, denoted $\textsf{view}\,\langle P_1(y_1), \ldots, P_k(y_k), V(z)\rangle\,(x)$, is the sequence of its outputs during its interaction with $P_1, \ldots, P_k$ on input x and auxiliary inputs $y_1, \ldots, y_k, z$. Note that it is a random variable which depends on the random coins of V.

Definition 6.2. Let $(P_1, \ldots, P_k, V)$ be a multi-prover interactive proof system for a language L. Let $P_L(x)$ be the set of auxiliary inputs for the provers that satisfies completeness for x.

We say that $\{P_1, \ldots, P_k\}$ is a *zero-knowledge interactive Turing machine set* if for every probabilistic polynomial-time verifier V^*, there is an expected polynomial-time *simulator* S such that $\forall x \in L \forall y_1, \ldots, y_k \in P_L(x), \forall z \in \{0,1\}^*$, $S(x,z)$ and $\textsf{view}\,\langle P_1(y_1), \ldots, P_k(y_k), V^*(z)\rangle\,(x)$ are identically distributed, where the randomness is over the random coins of S and V.

$(P_1, \ldots, P_k, V)$ is a *zero-knowledge multi-prover interactive proof system* if $\{P_1, \ldots, P_k\}$ is a set of zero-knowledge interactive Turing machines.

Zero-knowledge multi-prover protocols are a special case of general multi-prover protocols. Thus the definition of strongly isolating protocols carries over; i.e., a zero-knowledge multi-prover protocol is strongly isolating if it can be implemented with a strongly isolating verifier.

6.3 Information-Theoretically Secure Bit-Commitment

The strength of zero-knowledge in the single-prover case rested on the strength of the bit-commitment protocol. We will show that it is possible to construct a statistically binding and perfectly concealing

bit-commitment scheme in the multi-prover case (we will focus on the 2-prover case). This will give us perfect zero-knowledge protocols for all languages in **NP** by switching from computationally binding bit-commitments to perfect ones [1].

This bit-commitment is from a prover to the verifier only. We will not need commitment for the other direction.

Construction 6.1. Statistically binding, perfectly concealing 2-prover bit-commitment protocol.

All parties agree on a security parameter k. P_1 and P_2 partition some of their private random tape into $k+1$-bit strings $\{(c_i, w_i)\}_{i \leq \mathbb{N}}$, where c_i are bits and $|w_i| = k$.

Pre-computation phase:

- V chooses a k-bit string z uniformly at random and sends it to P_2.
- P_2 responds with $d_i = w_i \oplus c_i \cdot z$, for $1 \leq i \leq N$, where N is sufficiently large (depending on the protocol which uses this bit-commitment scheme as a sub-protocol), and $c_i \cdot z$ are thought of as the product between a scalar c_i and a vector z, over $\mathbb{Z}_2$.

Commit phase:

- P_1 wishes to commit b_i to V as $[b_i] = b_i \oplus c_i$.

Unveil phase:

- P_1 sends w_i to V.
- V computes $c_i = 1$ if $d_i \oplus w_i = z$, or $c_i = 0$ if $d_i \oplus w_i = \vec{0}$ and recovers $b_i = [b_i] \oplus c_i$. V rejects if $d_i \oplus w_i$ does not equal to either z or $\vec{0}$. □

The prover which executes the pre-computation phase with the verifier for bit i cannot be the prover who unveils bit i. It is possible and simpler to delegate a prover P_{BC} to do nothing but execute this pre-computation phase, in a given protocol. Any other prover could then be trusted to unveil bits. Moreover, any prover (again, except P_{BC}) can unveil any bit, not necessarily one committed by itself, if it so chooses. They simply have to partition their private random tape appropriately beforehand.

It is sufficient for the verifier to fix a single 'master key' z and execute the pre-computation phase many times with a dedicated prover. This way, during the course of the protocol, provers will be able to show commitment equality, as will be discussed below.

It is clear that without executing the pre-computation phase with the appropriate prover, the verifier cannot know what bit has been committed.

Theorem 6.1. *Construction 6.1 is a perfectly concealing and statistically binding bit-commitment scheme*

Proof. Perfectly concealing – the verifier does not know the provers' shared uniformly random secret string w_i. Thus $d_i \oplus w_i$ is uniformly random from the verifier's point of view, and thus so is c_i and hence b_i.

Statistically binding – The prover who unveils does not know the verifier's secret string z which is used during the pre-computation phase. The verifier will only accept two strings during unveiling, w_i and $w_i \oplus z$. The prover can only guess the second one with exponentially small probability. □

It follows that every language in **PSPACE** has a perfect two-prover interactive proof by augmenting construction 3.8 with the above perfectly concealing bit-commitment.

It also follows that we have two-party information-theoretically secure coin-tossing by executing construction 6.1 with the perfect bit-commitment scheme above.

6.4 Bit-Commitment Equality

A feature of the above bit-commitment scheme is that there is a proof of bit-commitment equality built-in. If a prover wishes to show to the verifier that $[b_1]$ is a commitment to the same bit as $[b_2]$, then the prover needs only to provide $w_1 \oplus w_2$. The verifier computes $d_1 \oplus d_2 \oplus w_1 \oplus w_2 = c_1 \cdot z \oplus c_2 \cdot z = E$.

Now, we have the following.

- If $E = \vec{0}$, then $c_1 = c_2$, and thus $b_1 = b_2$ if and only if $[b_1] \oplus [b_2] = 0$.
- If $E = z$, then $c_1 \neq c_2$, and thus $b_1 = b_2$ if and only if $[b_1] \oplus [b_2] = 1$.
- If E is neither $\vec{0}$ nor z, the verifier rejects.

As before, any prover can prove that any two commitments are equal, even if said prover is not the one who commits, since all of the provers share the same private random tape, and its partitioning.

Note that this only works if the two commitments $[b_1]$ and $[b_2]$ are under the same random key the verifier sent during their respective pre-computation phase. However, as discussed above, it is sufficient for the verifier to set a single 'master key' for an entire protocol.

This bit-commitment equality is naturally reusable. After $[b_1]$ and $[b_2]$ are shown to be equal by unveiling $w_1 \oplus w_2$, a prover can show that $[b_3]$ is equal to both of them by revealing $w_1 \oplus w_3$. Neither w_1, w_2, nor w_3 has been revealed during this process.

6.5 Strongly Isolating Perfect Zero-Knowledge MIP

Recall that the technique used to in construction 3.7, a zero-knowledge protocol for all of **PSPACE**, involved proving to the verifier that the committed transcript is a valid one. This is an **NP** statement which is reducible to an instance of the graph-3-coloring problem.

This technique will not work directly in the case of **MIP** because given a committed transcript, it is not possible for the verifier to know that it was generated by provers who were not actually communicating, and the bit-commitment used is perfectly concealing, making the statement no longer an **NP** one[1]. We will instead adapt construction 3.8 for **MIP**.

By theorem 5.2, the **NEXP**-complete protocol, construction 4.2 is already strongly isolating. The phases which only involves P_1 can be made zero-knowledge by executing it in committed form, as in the single-prover case. The phase in which the verifier asks P_2 a question can also be executed in committed form.

The problem is that in this interaction with P_2, the verifier must ask a question that it has asked P_1 in order to assure zero-knowledge. This problem is addressed in [1] by asking the first prover to compute the verifier's question in committed form, add a message authentication tag using a secret key that the provers share and finally ask the verifier to courier the message to P_2, who then checks whether the authentication is valid before executing its part of the protocol.

This solution is problematic because it cannot be implemented with a strongly isolating verifier even if the provers were to show that their authentication tag is not an attempt at communication. The message and its authentication would be tagged with P_1's tag only, so they could not be sent to P_2.

We present a different zero-knowledge **MIP** protocol which addresses this problem. Our solution basically asks the provers to encrypt an answer with a key that is based on the verifier's question. This way, there is no need for P_1 to authenticate the question that the verifier will ask P_2, since an honest verifier will receive two answers which can be proven to be equal, but a dishonest one will receive two independent uniformly random answers. This can be simulated, proving zero-knowledge; and it can be implemented by a strongly isolating verifier, proving isolation.

[1]Since the bit-commitment is perfectly concealing, there exists unveilings of either bit, whereas if the bit-commitment is computationally concealing, there is only one possible unveiling, which is the **NP**-witness

Construction 6.2. (Strongly isolating perfect zero-knowledge multi-prover protocol for **MIP**)

Let $L \in$ **NEXP** be a language, let $x \in L$ and let (P_1', P_2', Λ) be a multi-prover protocol from construction 4.2 with perfect completeness which accepts x. We construct a zero-knowledge protocol for provers P_1, P_2, P_{BC} and verifier V. There will be at most N bit-commitments. Let K be the number of coins needed by Λ to compute each question.

P_1 and P_2 agree on $N+1$ random strings $w_1, \ldots, w_N$ and γ of length K.

All parties agree on a security parameter σ.

The provers *normalize* the lengths of their answers uniformly to (a suitably large enough) D. This can be done by padding short answers with blank symbols, for example. We will not explicitly mention the normalization of the provers' messages in the protocol since it is straightforward.

1. (pre-computation)

   (a) The verifier chooses a commitment master key Γ of length σ uniformly at random.

   (b) V and P_{BC} execute the pre-computation phase of construction 6.1 $2N$ times using Γ as his random string for all bit-commitments, preparing N strings to be used for unveiling by P_1 and P_2 respectively.

   (c) V chooses uniformly at random a permutation π in S_N and sends it to P_1.

   (d) V constructs the arithmetization g of an instance of oracle-3-SAT (B, r, s) which can be satisfied if and only if $x \in L$. V asks the provers to store the appropriate oracle A.

   (e) P_1 commits $[\gamma]$ to V.

2. (multilinearity test) Let k be the number of oracle queries in this phase. For $1 \leq i \leq k$:

   (a) P_1 and V execute committed coin-tosses where P_1 commits $[w_{\pi(i)}]$ before receiving the K-bit random string r_i from V. The committed coins are $[w_{\pi(i)} \oplus r_i]$.

   (b) P_1 evaluates the circuit of Λ in committed form under committed input $[w_{\pi(i)} \oplus r_i]$, resulting in a committed question $[Q_i]$ which is sent to V. This is the question which would have been asked by Λ with coins $w_{\pi(i)} \oplus r_i$.

   (c) P_1 commits his answer $[A(Q_i)]$.

3. After committing all of his answers, P_1 and V evaluate a circuit description of Λ in committed form with inputs $[A(Q_1)], \ldots, [A(Q_k)]$. P_1 unveils the circuit's output. If it rejects, V rejects.

4. (sumcheck with oracle):

   Let

   $$g(z, Q_{k+1}, Q_{k+2}, Q_{k+3}, A(Q_{k+1}), A(Q_{k+2}), A(Q_{k+3}))$$

   be the arithmetization in question, and let z be a string of length r and $Q_{k+1}, Q_{k+2}, Q_{k+3}$ be strings of length s, as described in construction 4.2. V and P_1 execute construction 4.2 in committed form, using committed coin-toss as usual. Let r_{k+1}, r_{k+2}, r_{k+3} be V's half of the coin tosses for each bit of Q_{k+1}, Q_{k+2}, Q_{k+3} respectively, and let $w_{\pi(k+1)}$, $w_{\pi(k+2)}$, $w_{\pi(k+3)}$ be P_1's half. At the end of this phase, P_1 will show that the committed final value is equal to

   $$g([z], [Q_{k+1}], [Q_{k+2}], [Q_{k+3}], [A(Q_{k+1})], [A(Q_{k+2})], [A(Q_{k+3})]),$$

   an evaluation in committed form of g using the committed random bits that was used during the protocol's loop. If this fails, V rejects.

5. (non-adaptiveness test):

   (a) V randomly chooses $1 \leq i \leq k+3$, the index of an oracle query which was made to P_1, and sends i to P_1.

   (b) P_1 responds with the commitment $[\Omega_1] = [A(Q_i)] \oplus \langle [r_i], [\gamma] \rangle$, where the brackets indicate the dot-product, and proves to V that Ω_1 is computed correctly, using the commitments it has already made.

   (c) V sends $(r_i, \pi(i))$ to P_2.

   (d) P_2 computes[2] Q_i from r_i, $w_{\pi(i)}$ and Λ's circuit. It responds with the commitment $[\Omega_2] = [A(Q_i)] \oplus \langle [r_i], [\gamma] \rangle$.

   (e) P_2 shows, using bit-commitment equality from construction 6.1, that $\Omega_1 = \Omega_2$ (recall that with our multi-prover bit-commitment scheme, any prover can unveil any bit, including those which are not its own). If equality is valid, V accepts. Otherwise reject. □

Theorem 6.2. *The above construction is a strongly isolating zero-knowledge multi-prover protocol.*

Proof.

Completeness – If $x \in L$, an honest prover is simply executing construction 4.2 in committed form, so Λ will always accept, and hence so will the verifier.

[2] In construction 4.2, P_2 does not learn the index of the question that is asked to him. To reduce the soundness of the current protocol to construction 4.2, we introduced the permutation π of the indices of the questions to P_1. This does not change P_1's part of the protocol, but makes oblivious to P_2 the index of the question.

Soundness – We claim that, assuming isolation, a prover cannot know how to unveil a bit differently than when the commitment occurs. That is, if the prover does not know how to unveil a bit when it is committed, it will never gain the information necessary at any later point. This, coupled with the statistical binding condition, guarantees that the prover cannot cheat after commitment, and that the soundness of our protocol reduces to the soundness of the underlying protocol from [2].

To see this, note that the pre-computation phase is completed before the verifier engages with anyone else. During this phase, to prepare for the ith bit-commitment $[b_i]$, P_{BC} sends to the verifier x. From this point on, there are only two strings which the verifier will accept for the unveiling of the $[b_i]$: x and $x \oplus \Gamma$.

Let H be Shannon's entropy function. Let X and Y be random variables denoting P_1's knowledge of x and $x \oplus \Gamma$ respectively. Since Γ is uniformly random and independent of x, and also unknown to P_1, we have $H(X) + H(Y) \geq \sigma$.

Let U denote the messages sent by the verifier to P_1. Since we have isolation, $H(X|U) = H(X)$, and $H(Y|U) = H(Y)$. This gives us $H(X|U) + H(Y|U) = H(X) + H(Y) \geq \sigma$. Note that P_1 can only unveil if $H(X) < \epsilon$ or $H(Y) < \epsilon$ for small epsilon, since the probability of P_1 guessing the bits correctly decreases exponentially as $\epsilon \to 0$.

From this we can conclude that if a prover can unveil a commitment in one way after interaction with V, then it can do so at the time of commitment, and that it can only unveil in at most one way.

Now, P_1 and V must evaluate Λ in committed form. So if a string is not in the language, either Λ rejects with probability $2/3$, or P_1 fails the non-adaptiveness test with some probability. Conditioned on Λ rejecting, there must be a gate which is wrongfully evaluated, and which P_1 must either unveil to be the case or not know how to unveil at all. Λ being a polynomial-sized circuit, the probability of P_1 being detected is thus at least $(2/3)\mathbf{poly}(|\Lambda|)$, which can be amplified exponentially fast.

If P_1 fails the non-adaptiveness test, [2] showed that there is at least $1/\mathbf{poly}(k)$ of being detected. By the same arguments regarding P_1, P_2 must fail to unveil correctly, or at all, with at least such a probability, in the non-adaptiveness test.

Perfect Zero-knowledge – We provide a perfect simulator for a probabilistic polynomial-time machine V^* interacting with the three provers P_1, P_2, P_{BC}.

We will restate here the assumption that if V^* deviates from the protocol by not sending a message from the space of expected messages (for example, if the verifier sends a string of incorrect length, or containing symbols from an invalid alphabet), then the simulator outputs what it has so far and aborts, mirroring what the provers would do.

The simulation is long but straightforward, and relies on the following three facts:

1. The simulator will execute the pre-computation phase as P_{BC} with V^*, thus it will know V^*'s bit-commitment master key Γ. Hence, the simulator can unveil any commitment as either 0 or 1. In particular, it can always unveil the output of Λ to be **accept**, and it will always succeed in convincing the verifier that a gate has been correctly evaluated in committed form.

   A malicious verifier may execute the protocol out of order, for example by interacting with P_{BC} last. This is not a problem, as the verifier needs both the information from either P_1 or P_2, and the pre-computation strings from P_{BC} to fully unveil a bit. Thus the simulator can retroactively decide which strings to send when emulating P_{BC} appropriately.

2. The provers' answers are committed strings of equal length D, and by the perfectly concealing property of the bit-commitment, from the verifier's point of view, they are independent uniformly random strings of the same length.

3. Whether the final outputs Ω_1 and Ω_2 are equal depends on whether the verifier has asked the two provers the same question during the non-adaptive test. By the perfectly concealing property of the bit-commitment and the fact that the verifier does not know the uniformly random string γ, either Ω_1 and Ω_2 are the same random string (if the verifier has asked the same question, as it should), or two independent uniformly random strings (if the verifier cheats and does not ask the same question).

Here is a high-level description of the simulator. The ability of the simulator to unveil any bit-commitment either way makes a detailed description redundant.

1. The simulator begins by taking V^* and fill its random tape with as many fresh coins as needed. It executes the pre-computation phase by emulating P_{BC} and obtains V^*'s bit-commitment master key Γ. The simulator commits a uniformly random string γ to be used later during the non-adaptiveness test. From then on it ignores V^*'s messages and responds with uniformly random strings of the correct length.

2. The simulator initiates the multilinearity test. It coin-tosses with V^*, who sends the strings r_i, and computes its questions Q_i correctly. The simulator ignores the questions and answers with $[A(Q_i)]$, uniformly random strings of length D.

   When everything has been computed, the simulator uses the fact that it can unveil any commitment either way to show that Λ accepts.

3. The simulator executes the zero-knowledge sumcheck exactly in the same way. It will succeed again by the fact that it can open the commitments either way. Let $Q_{k+1}, \ldots, Q_{k+3}$ be the questions asked by V^* during this phase, and let $r_{k+1}, \ldots, r_{k+3}$ be V^*'s half of the coin-tosses.

4. For the non-adaptiveness test, V^* will ask P_1 to encrypt the existing answer $A(Q_i)$. The simulator emulates P_1 by answering honestly by computing in committed form $[\Omega_1] = [A(Q_i)] \oplus \langle [r_i], [\gamma] \rangle$ with V^*, where $[A(Q_i)]$ is the bogus answer it gave earlier.

   V^* will then ask P_2 to toss coins using r. The simulator ignores the coins and commits a uniformly random string $[\Omega_2]$.

   If $r = r_i$, then the simulator unveils $[\Omega_1]$ and $[\Omega_2]$ to be the same uniformly random string by tossing coins. If not, then the simulator will unveil them to be two independent uniformly random strings by tossing twice as many coins.

Strong Isolation – All messages sent by V are independent coin-flips, which are computed without any previous inputs. Thus they are all cleared to be sent to every prover. This includes $(r_i, \pi(i))$ which, although is sent to both P_1 and P_2, is not computed from any previous messages. □

We reiterate the two key ideas behind the simulator.

The first is that the simulator behaves as all three provers, in particular P_{BC}, and thus it knows the bit-commitment master key Γ. This allows the simulator to unveil any commitment either way, allowing for any circuit to be correctly evaluated in committed form, and its output to be unveiled to be whatever the simulator wants. The fact that the underlying protocol (P_1', P_2', Λ) has perfect completeness means that the simulator does not have to guess Λ's accept probability when $x \in L$.

The second is that the final answers are two *identical* uniformly random strings when the verifier is honest, and two *independent* random strings when the verifier is not. The simulator only has to check the question, and does not have to know anything about the oracle A.

Chapter 7

Open Problems

It was shown in [1] that, in the case of non-zero-knowledge mutli-prover protocols, two provers suffice. In fact, if we do not consider the problem of isolation, two provers suffice for zero-knowledge multi-prover protocols as well. However, our approach to isolation uses three provers, one of which is dedicated to the pre-computation phase of our bit-commitment protocol. This is because in the non-adaptiveness test of construction 6.2, P_2 must be able to unveil the commitments of P_1, so a third prover is used to achieve statistical binding. We leave the problem of whether it is possible to construct *strongly isolating* zero-knowledge protocols for **MIP** with only two provers to the reader.

In the case of single-prover zero-knowledge proofs, there is a *sequential composition theorem* which states that executing different zero-knowledge proofs in sequence remains zero-knowledge [7]. We would like to see a composition theorem which preserves both zero-knowledge and strong isolation. One possible problem in the case of strong isolation is that while going from one sub-protocol to another we would keep the same provers, and thus it is unclear how messages should be tagged. Also, since the provers have different roles, it is unclear whether having them switch roles from one sub-protocol to the next would affect zero-knowledge or isolation. Some notion of a multi-prover protocol 'parity' may need to be defined, so that protocols of a certain form, when composed with protocols of the same form (but not necessarily the same protocol), retain zero-knowledge and isolation.

In this work, we answered the question of how to isolate the provers so that it is impossible for non-local correlations to form between them. But suppose that we were to grant the provers the ability to perform arbitrary non-communicating non-local computation, how would soundness be affected? What class of languages would be accepted by such non-locally augmented multi-prover protocols? The answer must lie somewhere between **PSPACE** and **NEXP**.

Bibliography

[1] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. 1988. Multi-prover interactive proofs: how to remove intractability assumptions. In Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC '88). ACM, New York, NY, USA, 113-131. DOI=10.1145/62212.62223

[2] L. Fortnow, J. Rompel, and M. Sipser. 1994. On the power of multi-prover interactive protocols. Theor. Comput. Sci. 134, 2 (November 1994), 545-557. DOI=10.1016/0304-3975(94)90251-8

[3] L. Babai, L. Fortnow, and C. Lund. 1990. Nondeterministic exponential time has two-prover interactive protocols. In Proceedings of the 31st Annual Symposium on Foundations of Computer Science (SFCS '90). IEEE Computer Society, Washington, DC, USA, 16-25 vol.1. DOI=10.1109/FSCS.1990.89520

[4] R. Cleve, P. Høyer, B. Toner, and J. Watrous. 2004. Consequences and Limits of Nonlocal Strategies. In Proceedings of the 19th IEEE Annual Conference on Computational Complexity (CCC '04). IEEE Computer Society, Washington, DC, USA, 236-249. DOI=10.1109/CCC.2004.9

[5] C. Crépeau, L. Salvail, J.-R. Simard and A. Tapp. 2011. Two provers in isolation. In Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security (ASIACRYPT '11), Dong Hoon Lee and Xiaoyun Wang (Eds.). Springer-Verlag, Berlin, Heidelberg, 407-430. DOI=10.1007/978-3-642-25385-0_22

[6] G. Brassard, C. Crépeau, and M. Yung. 1991. Constant-round perfect zero-knowledge computationally convincing protocols. Theor. Comput. Sci. 84, 1 (July 1991), 23-52. DOI=10.1016/0304-3975(91)90259-5

[7] O. Goldreich. 2000. Foundations of Cryptography: Basic Tools. Cambridge University Press, New York, NY, USA.

[8] S. Arora and B. Barak. 2009. Computational Complexity: A Modern Approach (1st ed.). Cambridge University Press, New York, NY, USA.

[9] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. 1990. Everything provable is provable in zero-knowledge. In Proceedings on Advances in cryptology (CRYPTO '88), Shafi Goldwasser (Ed.). Springer-Verlag New York, Inc., New York, NY, USA, 37-56.

[10] C. Crépeau, J. Graaf and A. Tapp. 1995. Committed Oblivious Transfer and Private Multi-Party Computation. Advances in Cryptology (CRYPTO '95), D. Coppersmith (Ed). Springer, Berlin, Heidelberg, 110-123. DOI=10.1007/3-540-44750-4_9

[11] S. Popescu and D. Rohrlich. Quantum nonlocality as an axiom. Foundations of Physics, 24:379385, 1994. 10.1007/BF02058098

[12] J. Håstad, R. Impagliazzo, L.-A. Levin, and M. Luby. A pseudo-random generator from any one-way function. SIAM J. Comput., 28(4):12–24, 1993.