A Survey on Zero Knowledge Proofs

Li Feng and Bruce McMillin

Contents

Abstract

Zero Knowledge Proofs (ZKPs) are interactive protocols in which one party, named the prover, can convince the other party, named the verifier, that some assertion is true without revealing anything other than the fact that the assertion being proven is true. This chapter is a survey on ZKPs including their background, important concepts, applications for NP problems, and composition operations of ZKPs. The remarkable property of being both convincing and yielding nothing except the assertion is indeed valid makes ZKPs very powerful tools for the design of secure cryptographic protocols. In this chapter, ZKPs are constructed for the exact cover and 0-1 simple knapsack problem.

1. Background and Motivation

In order to deal with the problem of distrustful parties convincing each other that the message they are sending is indeed computed according to their predetermined procedure without yielding any secret knowledge, a number of privacy-preserving communication protocols have been proposed and designed, including for example, the fuzzy vault scheme, k-anonymity, nondeducibility, ZKPs, and so on.

Shamir provided a scheme to share a secret in [39]. Rather than straightforward encryption, the main idea is to divide data D into n pieces in such a way that D is easily reconstructible from any k pieces, but even combining k-1 pieces won't reveal any information about D. This scheme is called a (k, n) threshold scheme. Threshold schemes are ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate.

The fuzzy vault is a novel cryptographic notion developed by A. Juels and M. Sudan in [31], which can be considered as a form of error-tolerant encryption operation. In a fuzzy vault scheme, one party places a secret value k in a fuzzy vault and "locks" it using a set A of elements from some public universe U; the other party can "unlock" the vault to get the secret value k using a set B of similar length only if A and B overlap substantially. A fuzzy fault scheme can be constructed based on any type of linear error-correcting code.

Another privacy-preserving framework for communication is called k-anonymity. A data record is k-anonymous if and only if it is indistinguishable in its identifying information from at least k specific records. k-anonymity provides a formal way of preventing re-identification of data to fewer than a group of k data items. k-anonymity provides a formal way of generalizing aggregated data without violating the privacy [30].

Nondeducibility is a security model based on information theory, especially for information sharing. The model states that information flows in a system from high-level objects to low-level objects if and only if some possible assignment of values to the low-level objects in the state is incompatible with a possible assignment of values to the state's high-level objects [33]. To be more specific, a system is considered nondeducible secure if it is impossible for a low-level user, through observing visible events, to deduce anything about the sequence of inputs made by a high-level user.

The notion of ZKPs, which is first introduced by Goldwasser, Micali and Rackoff [24], is to obtain a system in which it is possible for a prover to convince a verifier of his knowledge of an assertion without disclosing any information except the validity of his assertion. ZKP study has become a very active research area because of its fascinating nature of a seemingly contradictory definition; ZKPs have the remarkable property of being both convincing and yielding nothing except that the assertion is indeed valid, which makes them very powerful tools for the design of a secure cryptographic protocol [19].

2. Introduction

2.1. Interactive Proof and Its Properties

Before presenting the definition of ZKPs, we first introduce the notion of an overarching class, the interactive proof.

Definition (Interactive Proof [24]) An interactive proof system for a language L is a pair of interactive Turing Machines, $(P,\ V)$ in which V, the verifier, executing a probabilistic polynomial-time strategy and P, the prover, executing a computationally unbounded strategy, satisfy two properties: completeness and soundness.

- Completeness can be expressed as:

$$Prob[(P, V)(x) \; = \; 1 \mid x \in L] \; \geq 1 - |x|^{-c} \;\; (c > 0)$$

- Soundness can be expressed for every interactive Turing machine P^*as:

$$Prob[(P^*, V)(x) \; = \; 0 \mid x \notin L] \; \geq 1 - |x|^{-c} \;\; (c > 0)$$

Completeness means that the verifier accepts the proof if the statement or assertion is true. In another words, an honest verifier will always be convinced of a true statement by an honest prover. Soundness means if the fact is false, the verifier rejects the proof, which indicates that a cheating prover can convince an honest verifier that some false statement is actually true with only a small probability. Both conditions allow a negligible error probability, which plays an important role in ZKPs with a robust notion of rareness; a rare event should occur rarely even if we repeat the experiment for a feasible number of times [22].

Proofs here do not represent the traditional mathematical concept of a proof. Mathematical proofs are strict, using either self evident statements obtained or from proofs established beforehand. Here, proofs are not a fixed and static object; they are similar to the dynamic process used to establish the truth of a statement throughout the exchange of information. Thus, they are considered to be probabilistic rather than absolute. The class of problems having interactive proof systems is denoted as IP [22].

2.2. A Simple Example

“In cryptography, the zero-knowledge protocol is an interactive method for one party to prove to another that a statement is true, without revealing anything other than the veracity of the statement [19]”. Each party in the protocol does a sequence of actions. While receiving a message from the other party, it performs a private computation, then it sends a message to the other party. Those actions will be repeated many rounds. Then the verifier either accepts or rejects the prover’s proof.

A very simple example of a ZKP could be illustrated as a story of Ali Baba's Cave [2]. In this story Figure 1, Peggy (the prover) has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a circle, with the entrance on one side and the magic door blocking the opposite side. Victor (the verifier) says he'll pay her for the secret, but not until he's sure that she really knows it. Peggy says she'll tell him the secret, but not until she receives the money. They devise a scheme by which Peggy can prove that she knows the word without telling it to Victor. Peggy goes into a random branch of the cave without letting Victor knowing which branch she chose. Standing at the entrance to the cave, Victor calls out a random branch he wants Peggy to come out from. If Peggy indeed knows about the secret password, she can obey every time. If she doesn't know the secret password, she has a 50% of initially fooling Victor. If Victor is happy with a 1 in 1024 chance that Peggy is cheating, there will be 10 iterations since $\frac{1}{2^{10}} = \frac{1}{1024}$.

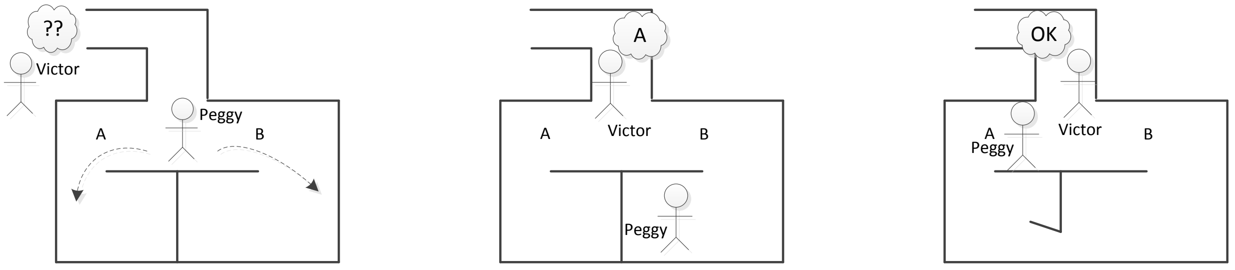


Figure 1. Ali Baba's Cave

2.3. Computational Indistinguishability

"Effective similarity" is a widely accepted notion in Modern Cryptography. If the differences between the objects can't be observed through a feasible computation, then the two objects are considered equivalent. In the definition of zero knowledge, it is impossible to tell the differences among computationally indistinguishable objects. The notion of computational indistinguishability underlies the definition of general zero-knowledge [22].

For $S \subseteq \{0,\ 1\}^*$, the probability ensembles $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$ contain X_α and Y_α, which is a distribution that ranges over strings of length polynomial in $|\alpha|$. A polynomial-size circuit $\{D_n\}$ means there exists a deterministic Turing Machine which has a running time polynomial in n on the circuit. $\Pr\left[D_n\left(X_\alpha\right)=1\right]$ denotes the probability that circuit D_n outputs 1 on input X_α.

Definition (Computational Indistinguishability [23], [43]) $X=\{X_\alpha\}_{\alpha\in S}$ and $Y=\{Y_\alpha\}_{\alpha\in S}$ are computationally indistinguishable if for every family of polynomial-size circuits $\{D_n\}$, every polynomial p, and sufficiently large n and every $\alpha\in\{0,\ 1\}^n\cap S$,

$$\left|\Pr\left[D_n\left(X_\alpha\right)=1\right]-\Pr\left[D_n\left(Y_\alpha\right)=1\right]\right|<\frac{1}{p(n)}$$

where the probabilities are taken over the relevant distribution.

2.4. One-way Function

In general, Modern Cryptography is always concerned with a question of whether one-way functions exist. One-way functions provide us the equivalent of digital lockable boxes. They are functions that are easy to evaluate but hard (on the average) to invert, which has an intractability characteristic.

Definition (One-way Function [22]) A function $f\ :\ \{0,\ 1\}^*\rightarrow\{0,\ 1\}^*$ is called one-way if the following two conditions hold:

1. Easy to evaluate: There exists a polynomial-time algorithm A such that $A\left(x\right)=f(x)$ for every $x\in\{0,\ 1\}^*$.

2. Hard to invert: For every family of polynomial-size circuits $\{C_n\}$, every polynomial p, and all sufficiently large n,
$$Pr[C_n\left(f\left(x\right)\right)\in f^{-1}\left(f\left(x\right)\right)]<\frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible choices of $x\in\{0,\ 1\}^n$.

Because the problem of factoring large integers is computationally intractable, the function that takes as input two (equal length) primes and outputs their product is widely believed to be a one-way function [30].

2.5. Simulation Paradigm

The simulation paradigm is another important notion, which was first developed in the context of zero-knowledge, presented by Goldwasser, Micali and Rackoff [5]. The crucial point is that for every algorithm that represents the strategy of the verifier, there exists a simulator that can simulate the entire interaction of the verifier and the honest prover without access to the prover's auxiliary information. It is an approach which makes the adversary gains nothing if whatever it can obtain by unrestricted adversarial behavior can be obtained within essentially the same computational effort as the prover. In this way, the adversary gains nothing.

A simulator is defined as a method or procedure that generates fake (generated without the prover) views that are indistinguishable from a genuine (generated with the prover) view of a proof [40]. To be more specific, whatever a verifier might have learned from the interaction with the prover, he could have actually learned by himself by running the simulator. Figure 2 illustrates the idea. The concept of a simulator helps defining the zero knowledge property in another way, that is, a proof of knowledge has the zero knowledge property if there exists a simulator for the proof.

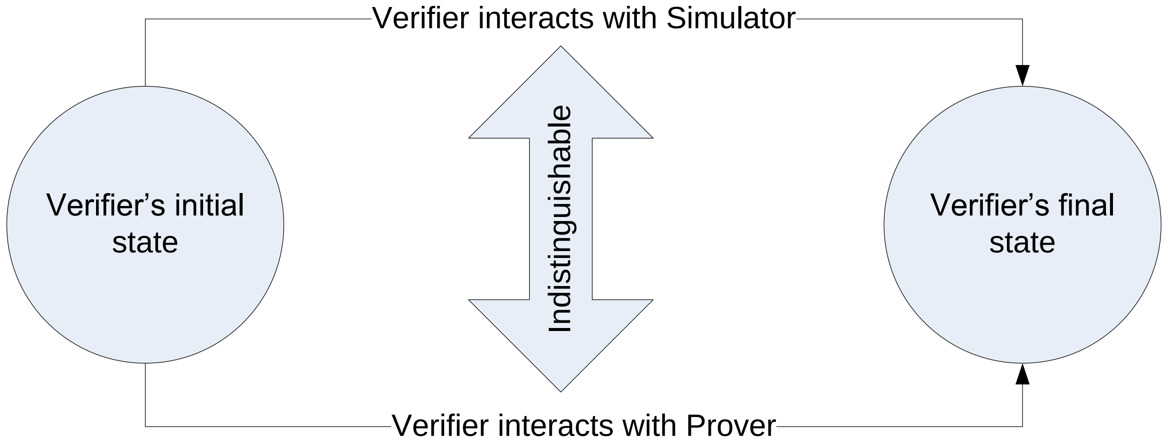

Verifier interacts with Simulator
Verifier's initial state
Indistinguishable
Verifier's final state
Verifier interacts with Prover

Figure 2. Simulator

2.6. Definition of ZKP

Based on the above important concepts, ZKP is defined as an interactive proof system with zero-knowledge. The zero-knowledge property means no amount of knowledge should pass between the prover and verifier.

Definition (Zero-knowledge [24]) An interactive strategy A is auxiliary-input zero knowledge on inputs from the set S if, for every probabilistic polynomial-time (interactive) strategy B^* and every polynomial p, there exists a probabilistic polynomial-time (non-interactive) algorithm C^* such that the following two probability ensembles are computationally indistinguishable [22]:

1. $\{(A,\ B^*(z))\,(x)\}_{x\in S\ ,\ \ z\in\{0,\ 1\}^{p(|x|)}} \overset{\text{def}}{=}$ the output of B^* when having auxiliary-input z and interacting with A on common input $x \in S$; and

2. $\{C^*(x,\ z)\}_{x\in S\ ,\ \ z\in\{0,\ 1\}^{p(|x|)}} \overset{\text{def}}{=}$ the output of C^* on input $x \in S,\ \ z \in \{0,\ 1\}^{p(|x|)}$.

The first one represents an actual execution of an interactive protocol. The second one represents the computation of a stand-alone procedure: the simulator does not interact with anybody. The more basic definition of zero knowledge is obtained by eliminating the auxiliary-input z from this definition. But almost all known ZKPs are in fact auxiliary-input zero-knowledge.

Since whatever can be extracted from interaction with A on input $x \in S$ can also be extracted from x itself, nothing was gained by the interaction itself. Thus, in the concept of ZKPs, the verifier does not obtain further information about the assertion other than its validity. Furthermore, there is no degradation with repeated usage of ZKPs, which means the number of iterations the protocol runs won't change the chances of success of obtaining the secret.

Zero knowledge is a security property which could protect the prover from leaking unnecessary information to any verifier.

2.7. Witness Indistinguishability

The notion of witness indistinguishability (WI) was introduced by Feige and Shamir in [12], which means the view of any verifier is “computationally independent” of the witness that is used by the honest prover as auxiliary private input.

Loosely speaking, for any NP-relation, an argument system for the corresponding language is called WI if no feasible verifier can distinguish the case in which the prover uses one NP-witness to some input from the case in which the prover is using a different NP-witness to the same input.

A witness is defined as a string w such that $(x, w) \in R$ [5]. For a binary relation R, $L\left(R\right)$ is defined as

$$\{x|\exists\ w\ s.t\ (x, w) \in R\}$$

Specifically, when proving that a string x is in $L\left(R\right)$, the prover gets a witness w such that $(x, w) \in R$. If the proof system is WI, then for any w, w' such that $(x, w) \in R$,$(x, w') \in R$, and for any polynomial-size verifier, the view of the verifier when interacting with the prover that uses w as auxiliary input is computationally indistinguishable from its view when interacting with the prover that uses w' as auxiliary input. The WI property is preserved under arbitrary composition of protocols [12].

2.8. Honest Verifier VS General Cheating Verifier

The definition of zero-knowledge refers to all feasible verifiers as either honest or cheating. Typically the verifier is viewed as an adversary that is trying to cheat and attempt to gain something by interacting with the prover. With respect to the honest verifier, it will interact with the prover as expected, excepting that it may also maintain a record of the entire interaction and output this record in the end [22].

2.9. Black-box Simulator VS Non-black-box Simulator

The definition of zero-knowledge requires that the interaction of the prover with any verifier is computational indistinguishable with an ordinary probabilistic polynomial-time machine that interacts with no one.

Since whatever can be computed after interacting with a prover can be computed on any random input by itself, the ZKP is computationally equivalent to a trusted oracle. This is the notion of a black-box simulator. A black-box simulator is one that can simulate the interaction of the prover with any such a verifier when given oracle access to the strategy of that verifier. To be more specific, a black-box simulator indicates that using any verifier as a black box, produces fake views that are indistinguishable from the verifier's actual interaction with the prover [5]. Most of previous zero-knowledge proofs or arguments are established using a black-box simulator.

But how can the simulator generate an interaction that is indistinguishable from the actual interaction with the prover? First, the simulator has access to the verifier's random question transcript, which means it can actually determine the next question that the verifier is going to ask. Second, the simulator has many ways to answer the verifier's questions. The technique is called rewinding which means if the simulator fails to answer a question provided by the verifier, it simply rewinds the verifier back and asks again [5].

However, when using the verifier's strategy as a black-box, it is hard for the simulator to take advantage of the knowledge of the verifier's random question transcript. Therefore, there are restrictions and negative results while applying black-box simulator. In [17], Goldreich and Krawczyk has shown non-existence of black-box 3-round ZKPs where a round defines number of steps in the protocol.

In order to overcome the restrictions of black-box simulators, non-black-box simulators were developed. The power of non-black-box simulators has been discovered by Barak in [5]. Barak has shown how to construct non-black-box simulators and obtained several results which are considered to be unachievable through a black-box simulator. He presented a zero-knowledge argument system for NP with following properties:

- It is the first zero-knowledge argument with a constant number of rounds and negligible soundness error.
- It is an Arthur-Merlin (public coins) protocol. The Arthur-Merlin protocol, introduced by Babai [4], is an interactive proof system in which the verifier's coin tosses are known to the prover too.

- It has a strict polynomial time simulator rather than expected polynomial time.

By taking advantage of the strategies of the verifier, since the knowledge of the verifier is not an oracle in non-black-box simulator, Barak's result calls for the re-evaluation of many common beliefs, such as non-existence of (black-box) 3-round ZKPs and impossibility of (black-box) constant-round concurrent zero-knowledge.

Barak's construction uses the FLS technique which was presented by Feige, Lapidot and Shamir in [11]. Two phases in the FLS technique will be combined together to obtain a ZKP. The first phase is called the generation phase, in which the prover and verifier will generate a string. In the second phase, the prover proves to the verifier that either the theorem is true or that the generated string satisfies some property. The FLS technique is based on the notion of WI.

Barak's framework is described in [5]:

- Common Input: x and a language L, such that $x \in L$, n is security parameter that makes sure this protocol remains zero-knowledge when executed up to n times concurrently.
- Auxiliary input to prover: w - a witness to the fact that $x \in L$.

1. $Phase1 : GenerationPhase$. The prover and verifier engage in some protocol whose output is a string τ.
2. $Phase2 : WIProofPhase$. The prover proves to the verifier using a WI argument that it knows either a string w such that $(x, w) \in R$ or a string σ such that $(\tau, \sigma) \in S$. R and S are two statements based on language L that the prover wants to convince to the verifier. R is a binary relation that (x, w) is tested. $L(R) \overset{def}{=} \{x|\exists w\ s.t.\ (x,\ w) \in R\}$. It means that if $x \in L(R)$ and w is a string such that $(x, w) \in R$. then w is a witness for the fact that $x \in L(R)$. S is a binary relation that (τ, σ) is tested. $L(S) \overset{def}{=} \{\tau|\exists \sigma\ s.t.\ (\tau,\ \sigma) \in S\}$. Similarly, it means that σ is a witness for the fact that $\tau inL(S)$. More formally, the prover proves to the verifier using a WI proof knowledge that it knows a witness to the fact that $(x, \tau) \in L(S')$ where S' is defined so that $(x, \tau) \in L(S')$ if and only if there exists w' such that either $(x, w') \in R$ or $(\tau, w') \in S$.

2.10. Quality of ZKPs

The soundness condition is considered as a "security" property because it protects the verifier from adversarial behavior by the prover [42]. Usually, it has two commonly used versions:

- Statistical Soundness: If $x \notin L$, then for all, even computationally unbounded, strategies P^*, V accepts in $(P^*, V)(x)$ with probability at most 1/3. This gives rise to interactive proof systems.
- Computational Soundness: If $x \notin L$, then for all polynomial-time strategies P^*, V accepts in $(P^*, V)(x)$ with probability at most 1/3. This gives rise to interactive argument systems.

Zero knowledge is another "security" property, which protects the prover from leaking unnecessary information to the verifier. It comes in three versions of ZKPs based on three interpretations of "similarity".

1. The perfect zero-knowledge notion was suggested by Brassard and Crepeau [8]. The prover is restricted to polynomial time, while the verifier may have unlimited computing power. The computation simulating the actual interaction is identical to the original one.
2. Statistical zero-knowledge requires that the computation simulating the actual interaction is statistically (e.g. in variation distance) close to the original one (with negligible error - smaller than any polynomial fraction in the length of common input) [22]. What's more, statistical zero-knowledge requires that the zero-knowledge condition hold even for computationally unbounded verifier strategy V^*.
3. Computational (or rather general) zero-knowledge requires that the transcript of the interaction is indistinguishable only under the computational complexity assumption, which means that no polynomial-time algorithm can distinguish the two distributions except with negligible probability. It only requires the zero-knowledge condition hold for polynomial-time verifier strategy V^*.

The following definitions are from [29]. Let $(A,\ B)$ be an interactive protocol. Let T, the transcript, be a random variable denoting the verifier view during the protocol on input x. That is, T is the sequence of messages between the prover and verifier under the sequences of coin tosses for A and B. The string h denotes any private input that the verifier may have with the only restriction that its length is bounded by a polynomial in the length of the common input. $M(x,\ h)$ is distributed over the coin tosses of M on inputs x and h.

1. $(A,\ B)$ is perfect zero-knowledge for L if $\exists$ a probabilistic polynomial time Turing machine M s.t. $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$, the random variable $M(x,\ h)$ and T are identically distributed.

2. $(A,\ B)$ is statistically zero-knowledge for L if $\exists$ a probabilistic polynomial time Turing machine M s.t. $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$,for all constant $c > 0$ and sufficiently large $|x|$, $\sum_\alpha |prob\left(M\left(x,\ h\right) = \alpha\right) - prob(T = \alpha)| < \frac{1}{|x|^c}$

3. $(A,\ B)$ is computationally zero-knowledge for L if $\exists$ probabilistic polynomial time Turing machine M s.t. $\forall$ polynomial size circuit families $C = C_{|x|}$, $\forall$ constant $a,\ d > 0$, for all sufficiently large $|x|$ s.t. $x \in L$, and for all strings h such that $|h| < |x|^a$,

$$prob\left(C_{|x|}\left(\alpha\right) = 1 \;\middle|\; \alpha\ random\ in\ M(x,\ h)\right) - prob\left(C_{|x|}\left(\alpha\right) = 1 \;\middle|\; \alpha\ random\ in\ T\right) < \frac{1}{|x|^d}$$

Based on the two security conditions - soundness and zero knowledge that ZKPs has, we can obtain four kinds of ZKPs. They can be denoted as SZKP, CZKP, SZKA and CZKA, in which the prefix of S indicates statistical and C indicates computational, while the suffix of P indicates interactive proofs (statistical soundness) and A indicates interactive arguments (computational soundness).
SZKPs are defined in the definition of zero-knowledge that the distribution ensembles are required to be statistically indistinguishable rather than computationally indistinguishable. As far as security is concern, SZKPs are of course the most attractive because the following several reasons [22]:

- SZKPs offer information-theoretic security to both parties. CZKPs only offer computational security to the prover, and SZKA only offer computional security to the verifier. While SZKPs hold their security properties regardless of the computational power of the verifier.

- SZKPs provide a clean model for the study of various questions regarding zero-knowledge.
- SZKPs have some properties from a complexity theoretic point of view. It contains a number of specific problems believed to be hard such as Graph Nonisomorphism, Graph Isomorphism, Quadratic Residuosity, and Quadratic Nonresiduosity which are Karp-reducible and is closed under complementation.

Recalling the definition of zero-knowledge, it is a CZKP rather than a SZKP because there do not exist commitment schemes that are simultaneously statistically hiding and statistically binding, which will be discussed in the next section. Under standard complexity assumptions, SZKA can also be constructed for every NP. The intractability assumption used for constructing SZKA for NP seems stronger than the assumption used for constructing CZKP for NP. Assuming both constructions exist, which is more preferable is more or less depending on the application: is it more important to protect the prover's secrets or to protect the verifier from being convinced of false assertions?

3. NP problem and ZKPs

One of fascinating aspects of complexity theory is that if any one of the problems in the class could been constructed an efficient solution, it could automatically obtain efficient solutions for the entire class.

Definition (The Class NP) NP is the class of decision problems X that admit a proof system $F \subseteq X \times Q$, s.t. there exists a polynomial $p(n)$ and a polynomial-time algorithm A such that [40]:

1. $\forall\ x\ \in X\ \exists q \in Q$ s.t. $(x,\ q) \in F$ and moreover, the size of q is at most $p(n)$, where n is the size of x.
2. For all pairs $(x,\ q)$, algorithm A can verify whether or not $(x,\ q) \in F$.

Definition (Polynomial Reduction) Let A and B be two problems. We say that A is polynomially Turing reducible to B if there exists an algorithm for solving A in a time that would be polynomial if we could solve arbitrary instances of problem B at unit cost [40].

Definition (NP-complete Problem) A decision problem X is $NP-complete$ if $X \in NP$ and for every problem $Y \in NP$, Y is polynomially Turing reducible to X [40].

It is not yet known if there are efficient solutions (polynomial time) to these problem in NP-complete. Classic examples of these problems include satisfying a boolean formula, the traveling salesman problem, the knapsack problem and so on. But the validity of a proposed solution is easily tested. For example, consider the Hamiltonian Cycle Problem: it is hard to find an efficient algorithm to solve this problem, but it is very easy to verify if a sequence of nodes is a Hamiltonian cycle.

Assuming the existence of secure encryption functions, it has been shown that graph three-colorability, which is known as a NP-complete problem, has a ZKP [19]. Since any problem in NP can be reduced to graph three-colorability according to Karp [32], it ensures that any language in NP has associated with a ZKP system.

In the physical world, the secure encryption functions are represented by opaque, lockable boxes which are usually assumed to be available for the prover, and only the prover has the key. In order to implement the physical boxes algorithmically, a cryptographic primitive named “commitment scheme” [16] will be used . In ZKPs, the way of locking information is to “committing a secret bit” [44], which can be implemented by using any one-way function (assumed to exist).

This commitment scheme describes a two-stage protocol between the prover and the verifier [16]. In the first stage, the prover commits to a value b, which could be digital analogies of sealed envelopes (or, letter, locked boxes). This means sending the value to the verifier through binding some unique value without revealing the original value to the verifier (as when getting a locked box). This property is called hiding. The “hiding” property says that the verifier does not know anything about b during the commit stage. Later on, the prover “decommits” or “reveals” this value to the verifier. This means sending some auxiliary information that allows the verifier to read the uniquely committed value (as when sending the key to the lock) in order to assure that it is the original value. This property is called binding. The binding property means that after the commit stage, there is at most one value that the verifier can successfully open [41].

Recall that two security properties - hiding and binding - can be statistical (holding against computationally unbounded cheating strategies, except with negligible probability) or computational (holding against polynomial-time cheating strategies, except with negligible probability). It's impossible to achieve statistical security for both hiding and binding. But any one-way function could be used in the commitment scheme for statistical hiding or statistical binding.

The widely accepted theorem on how to construct CZKPs system for any NP-set was demonstrated by Goldreich, Micali and Wigderson [39]. And the first construction of SZKA was given by Brassard, Chaum and Crepeau independently [6]. Feige, Lapidot and Shamir designed a ZKP on identification schemes in [14]. Constant-round ZKPs were first showed by Feige and Shamir in [13].

4. ZKPs Applications

In this section, several problems are introduced with their ZKPs construction. To provide a ZKP for a general NP statement, one can translate it into an instance of the graph three-colorability problem using a Karp reduction [32]. Another way to provide ZKPs avoiding a Karp reduction is using the simulation method after exploiting the properties of the specific NP-complete language.

4.1. ZKP for Graph Three-Colorability

Definition (Graph Three-Colorability) A graph $G(V,\ E)$ is said to be three-colorable if there exists a mapping $\phi\ :V\ \rightarrow \{1,\ 2,\ 3\}$ (called a proper coloring) such that every two adjacent vertexes are assigned different colors (i.e., each $(u,\ v) \in E$ satisfies $\phi(u) =\ \phi(v)$). Such a three-coloring induces a partition of the vertex set of the graph to three independent sets. The language graph three-colorability, denoted G3C, consists of the set of undirected graphs that are three-colorable.

Figure 3 is a Peterson graph [1] with three colored vertices (Red, Green and Blue).

Suppose graph $G(V,\ E)$ is colored by ϕ ($\phi\ :V\ \rightarrow \{1,\ 2,\ 3\}$). Let $n = |V|\,,\ m = |E|$ and $S_3 = Sym\{1,\ 2,\ 3\}$. Since the graph is (simple and) connected, n and m are polynomially related (i.e., $n - 1 \leq m\ < n^2/2$).

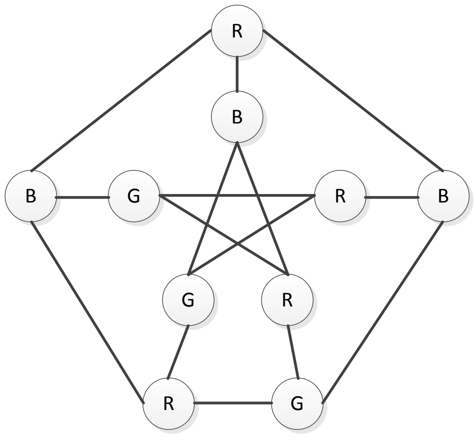


Figure 3. 3Colored Graph

Common input: A graph $G(V,\ E)$

The following four steps are executed m^2 times, each time using independent coin tosses [19].

1. P: The prover chooses at random an assignment of three colors to the three independent sets induced by ϕ, colors the graph using these three colors, and places these colors in n locked boxes each bearing the number of the corresponding vertex. More specifically, the prover chooses a permutation $\pi \in_R S_3$, places $\pi(\phi(i))$ in a box marked i ($\forall\, i \ \in V$), locks all boxes and sends them (without the keys) to the verifier.

2. V: The verifier chooses at random an edge $e \in_R E$ and sends it to the prover. (Intuitively, the verifier asks to examine the colors of the endpoints of $e \ \in E$)

3. P: If $e = (u,\ v) \in E$, then the prover sends the keys of u and v. Otherwise, the prover does nothing.

4. V: The verifier opens boxes u and v using the keys received and checks whether they contain two different elements of $\{1,\ 2,\ 3\}$. If the keys do not match the boxes, or the contents violates the condition then the verifier rejects and stops. Otherwise, the verifier continues to the next iteration.

If the verifier has completed all m^2 iterations then it accepts.

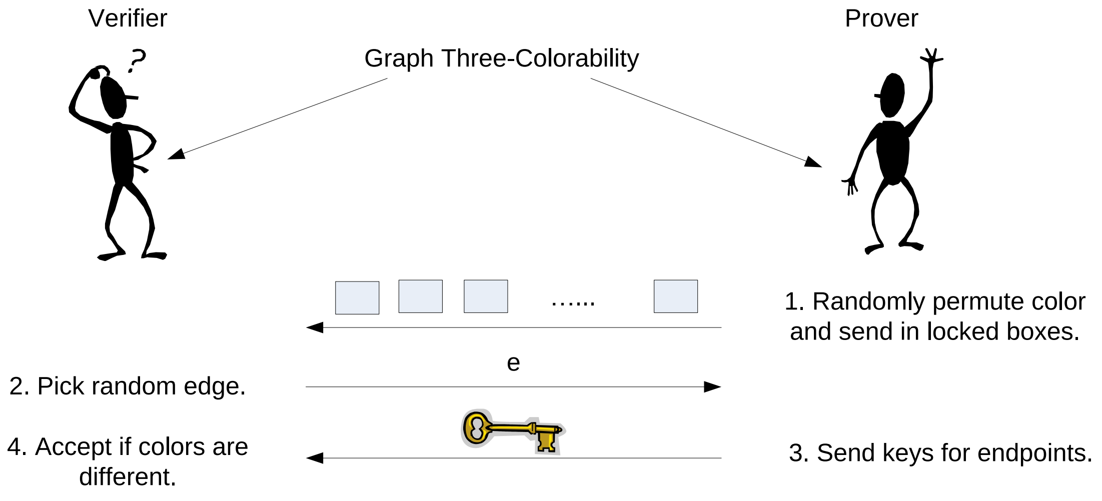

Verifier
Prover
Graph Three-Colorability
1. Randomly permute color and send in locked boxes.
e
2. Pick random edge.
4. Accept if colors are different.
3. Send keys for endpoints.

Figure 4. ZKP for G3C

Figure 4 illustrates the ZKP for G3C.

This protocol constitutes a ZKP for G3C since it satisfies three properties.

- Completeness: If the graph is three-colorable, then any pair of boxes u and v corresponding to some edge of the graph will certainly be colored differently. Therefore, an honest verifier will complete all m^2 iterations and accept with probability 1.

- Soundness: If the graph is not three-colorable and the verifier follows the protocol, then no matter how the prover plays, at each round the verifier will reject with probability at least $\frac{1}{m}$. The probability that the verifier will accept is bounded by $(1-\frac{1}{m})^{m^2} \approx \exp(-m)$.

- Zero-knowledge: The only information received by the verifier at each round is a pair of different randomly selected elements of $\{1, 2, 3\}$. It is crucial that the prover uses at each round an independently selected random permutation of the colors. Thus, the names of three classes at one round are uncorrelated to the names at another round.

4.2. ZKP for Feige-Fiat-Shamir Identification Scheme

The Feige-Fiat-Shamir identification scheme is one of the classic authentication zero-knowledge schemes [11]. The security of the Feige-Fiat-Shamir system is based on the fact that it is difficult to extract square roots modulo large composite integers of unknown factorization [40].

Initialization [35]:

1. A trusted center T selects and publishes an RSA-like modulus $n = pq$, but keep the primes p and q secret.
2. The prover selects a secret s coprime to n, $1 \leq s \leq n-1$, computes $v = s^2 \; mod \, n$, and registers v with T as her public key.

Identification Protocol [35]:

The following steps are executed t times, each time using independent random coin tosses.

1. P: The prover choses a random r, $1 \leq r \leq n-1$ and sends $x = r^2 \; mod \, n$ to the verifier.
2. V: The verifier randomly selects a bit $\alpha \; \in \{0, 1\}$ and sends α to the prover.
3. P: The prover computes and sends to the verifier y, where $y = r \; (if \; \alpha = 0)$, or $y = rs \, mod \, n \; (if \; \alpha = 1)$
4. V: The verifier rejects if $y = 0$ or if $y^2 \neq x \cdot v^{\alpha} mod \, n$.

Figure 5 illustrates the ZKP for Feige-Fiat-Shamir identification scheme.

If the verifier has completed all t iterations, then he accepts.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: Suppose the prover possesses the secret s. Then she can always correctly provide the verifier with $y = r$ or $y = rs \; mod \, n$ upon request. Therefore, an honest verifier will complete all t iterations and accept with probability 1.

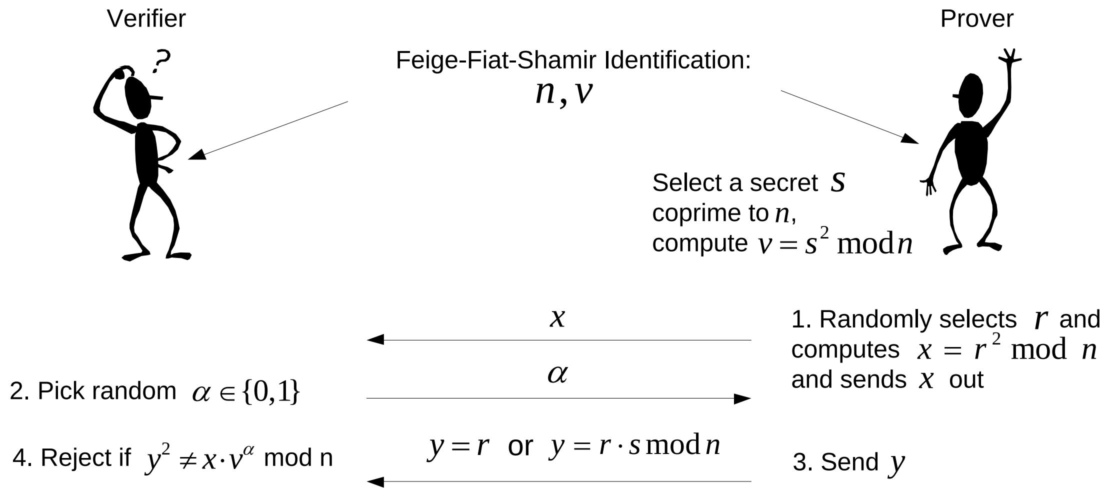

Verifier
Prover
Feige-Fiat-Shamir Identification:
n, v
Select a secret s coprime to n, compute $v = s^2 \bmod n$
x
1. Randomly selects r and computes $x = r^2 \bmod n$ and sends x out
2. Pick random $\alpha \in \{0,1\}$
α
4. Reject if $y^2 \neq x \cdot v^\alpha$ mod n
$y = r$ or $y = r \cdot s \bmod n$
3. Send y

Figure 5. ZKP for Feige-Fiat-Shamir Identification Scheme

- Soundness: Suppose the prover does not possess the secret s. Then, during any given round, she can provide only one of $y = r$ or $y = rs\ mod\ n$. Therefore, an honest verifier will reject with probability $\frac{1}{2}$ in each round, so the probability the verifier will be fooled is 2^{-t}.

- Zero-knowledge: The only information revealed in each round is $x = \ s^2 mod\ n$, and either $y = r$ or $y = rs\ mod\ n$. Such pairs (x, y) could be simulated by choosing y randomly, then define $x = y^2$ or $x = y^2/v$. Such pairs are computationally indistinguishable from the interaction with the prover.

4.3. ZKP for Graph Isomorphism

The Graph Isomorphism (GI) problem is basically asking the question: Given two graphs G_0 and G_1, is there a bijection between their sets of nodes that preserves edges?

Definition (Graph Isomorphism) Two graphs $G\,(V, E)\, and\ G'\,(V, E')$ are isomorphic if and only if there exists a permutation $\pi \in S_{|V|}$ (the symmetric group of $|V|$ elements) such that $(u, v) \in E$ iff $(\pi\,(u)\,, \pi\,(v)) \in E'$. We can write $G' = \pi G$.

There is an isomorphism between two graphs in Figure 6. $f(a) = 1, f(b) = 6, f(c) = 3, f(d) = 5, f(e) = 2, f(f) = 4$

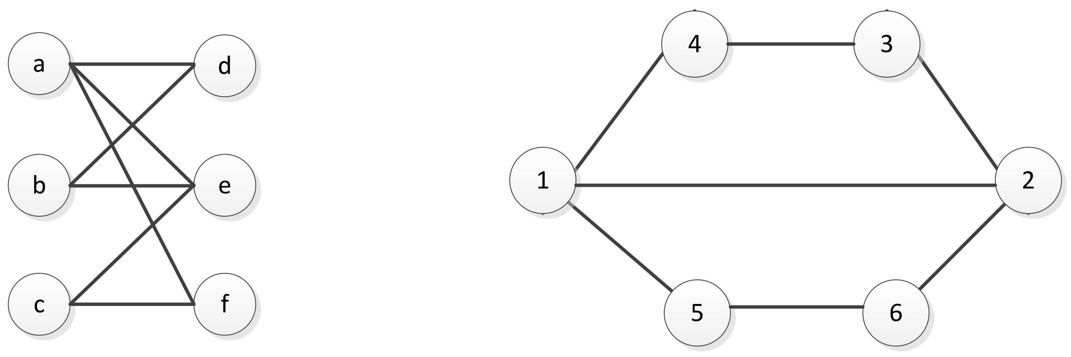


Figure 6. Graph Isomorphism

The language graph isomorphism (GI) consists of all the pairs of isomorphic graphs.

Common Input: Two graphs $G_0(V,\ E)$ and $G_1\,(V,\ E')$. Let ϕ denote the isomorphism between G_0 and G_1, that is $G_1 = \phi G_0$.

The following steps are executed t times, each time using independent random coin tosses [35].

1. P: The prover generates a graph, H, which is a random isomorphic copy of G_1. This is done by selecting a permutation $\pi \in_R S_{|V|}$, and computing $H = \pi G_1$. The prover sends the graph H to the verifier.

2. V: The verifier chooses at random $\alpha \in_R \{0, 1\}$, and sends α to the prover.

3. P: If $\alpha = 1$, then the prover sends $\beta = \pi$ to the verifier, else ($\alpha = 0$) the prover sends $\beta = \pi \cdot \phi$. Here, $\pi \cdot \phi$ indicates the combination of π and ϕ.

4. V: If the permutation β received from the prover is not an isomorphism between G_α and H (i.e., $H \neq \beta G_\alpha$), then the verifier stops and rejects; otherwise, he continues.

Figure 7 illustrates the ZKP for graph isomorphism.

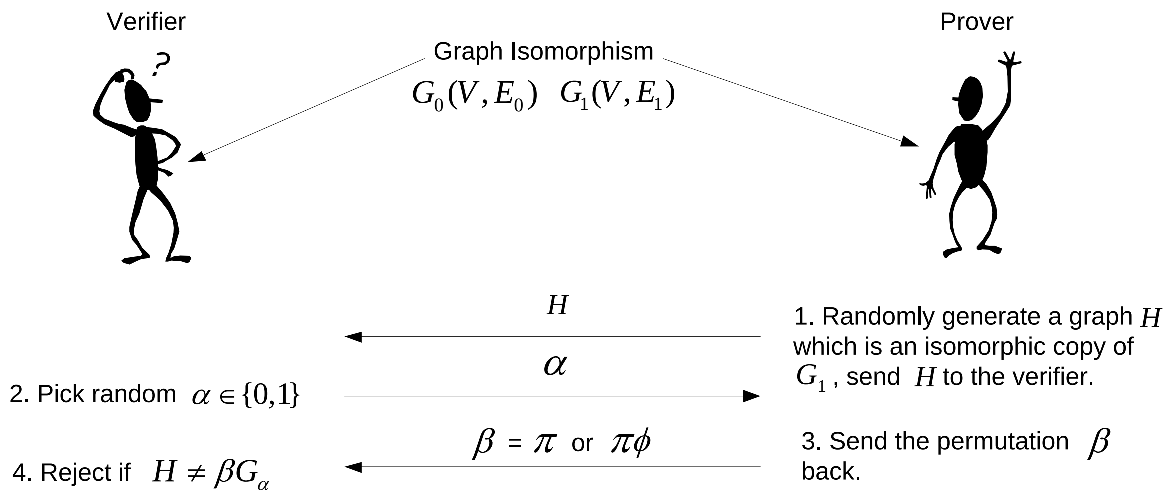

Verifier
Prover
Graph Isomorphism
$G_0(V, E_0)$ $G_1(V, E_1)$
H
1. Randomly generate a graph H which is an isomorphic copy of G_1, send H to the verifier.
α
2. Pick random $\alpha \in \{0,1\}$
$\beta = \pi$ or $\pi\phi$
3. Send the permutation β back.
4. Reject if $H \neq \beta G_\alpha$

Figure 7. ZKP for Graph Isomorphism

If the verifier has completed t iterations of the above steps, then he accepts.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: If $(G_0,\ G_1) \in GI$, then the random isomorphic copy H of G_1will always be isomorphic to both G_0 and G_1. Therefore, an honest verifier will complete all iterations and accept with probability 1.

- Soundness: If $(G_0,\ G_1) \notin GI$, then the random isomorphic copy H of G_1 will be isomorphic to only one of G_0 or G_1. Therefore, an honest verifier will reject with probability $\frac{1}{2}$ in each round.

- Zero-knowledge: The only information revealed in each round is either π or $\pi \cdot \phi$ where $\pi {\in_R} S_{|V|}$. Due to the random selection of π, a simulator which computes a random isomorphic copy of G_0 or G_1 or both is computationally indistinguishable from interaction with the prover.

4.4. ZKP for Hamiltonian Cycle

Definition (Hamiltonian Cycle) Let G be a graph. A Hamiltonian Cycle in G is a cycle that passes through all the nodes of G exactly once. The n nodes of G are labeled $N_{1,}\ N_{2,}\ \ldots\ldots N_n$.

Figure 8 has a Hamiltonian Cycle labeled by the solid line.

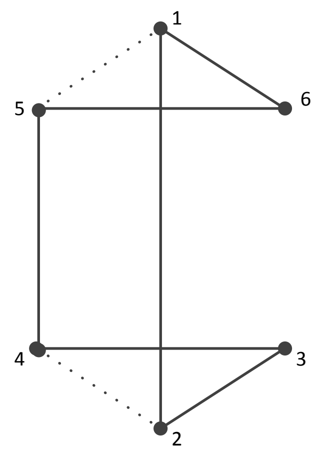


Figure 8. Hamiltonian Cycle

The following protocol will be executed for k rounds.

1. P: The prover encrypts G with the boxes in secret by randomly mapping n labeled nodes $N_1, N_2, \ldots\ldots N_n$ 1-1 into n labeled boxes $B_1, B_2, \ldots\ldots B_n$ in such a way that each one of the $n!$ permutations of the nodes into the boxes is equally probable. For every pair of boxes $(B_i,\ B_j)$ prepare a box labeled B_{ij}. This box is to contain a 1 if the node placed in B_i is adjacent to the node in B_j; 0 otherwise. All $(n + C_2^n)$ boxes are then to be locked and presented to the verifier.

2. V: Upon receiving $(n + C_2^n)$ boxes, the verifier has two choices:

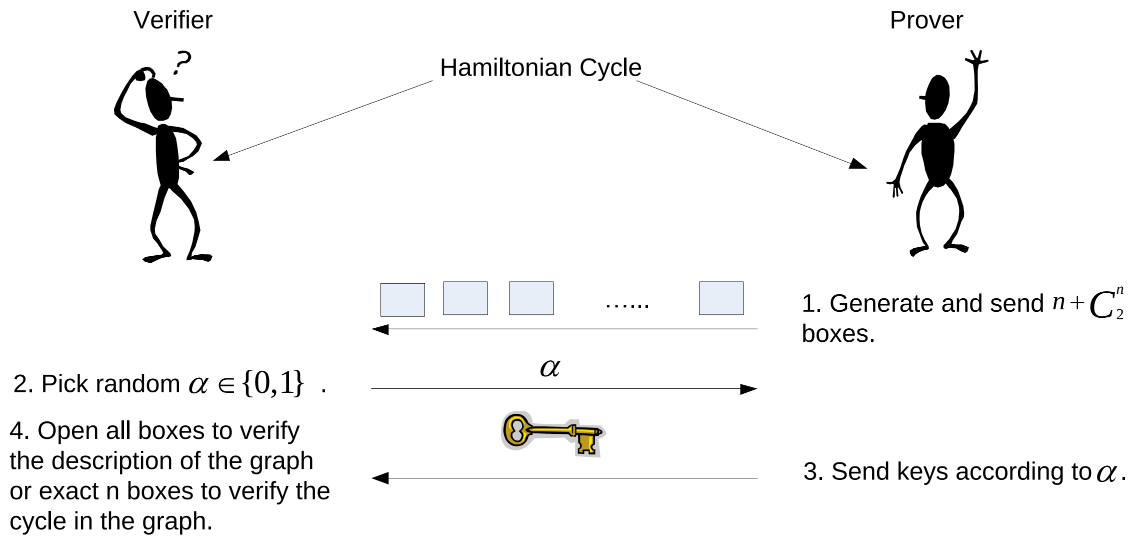
   - $(\alpha = 0)$ The verifier can request that the prover unlock all the boxes. In this case, the verifier may check that the boxes contain a description of G. (For example, if N_1 is adjacent to both N_2, N_5 but no other nodes in G. If N_1 is in B_i, N_2 in B_j, and N_5 in B_k, then there should be a 1 in both B_{ij}, B_{ik}, and a 0 in B_{ix} for every other value of x)

   - $(\alpha = 1)$ The verifier can request that the prover open exactly n boxes $B_{ij}, B_{jk}, B_{kl} \ldots\ldots B_{l'i}$, which can demonstrate that G contains a Hamiltonian Cycle if all these boxes all contain a 1.

Since the B_i are not opened, the sequence of node numbers defining the Hamiltonian Cycle in G is not revealed.

3. P: The prover opens the appropriate boxes according to one of the two random requests from the verifier (either the graph or the Hamiltonian Cycle)

4. V: The verifier accepts the proof if the prover complies and rejects otherwise.



Verifier
Prover
Hamiltonian Cycle
1. Generate and send $n + C_2^n$ boxes.
2. Pick random $\alpha \in \{0,1\}$.
α
3. Send keys according to α.
4. Open all boxes to verify the description of the graph or exact n boxes to verify the cycle in the graph.

Figure 9. ZKP for Hamiltonian Cycle

Figure 9 illustrates the ZKP for Hamiltonian Cycle.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: If the prover knows the graph, G, contains a Hamiltonian Cycle, he can successfully show the graph or cycle according to the requests from the verifier. So the verifier will complete all k rounds and accept with probability 1.

- Soundness: If the graph doesn't contain a Hamiltonian Cycle, the prover's probability of convincing the verifier that he does know the graph contains a Hamiltonian Cycle is $1/2$ in each round. His probability of convincing the verifier that he does know are $\leq 1/2^k$ when there are k rounds.

- Zero-knowledge: In each round, the verifier either obtains an encrypted graph of G or a random n cycle. When the prover reveals all the boxes that describe the G, it is only one of the $n!$ random mappings of the n nodes of G into the n labeled boxes. When the prover reveals the boxes containing a cycle, it is just a random n cycle. Thus the verifier gets no useful information.

4.5. Another ZKP for Graph Three-Colorability

The n nodes of G are labeled $N_{1,}\ N_{2,}\ \ldots\ldots N_n$. Suppose the graph is colored with Red, White and Blue. If node N_i is colored red, it is called N_i^R.

1. P: The prover prepares $3n$ pairs of boxes $\langle B_1^c,\ B_1\rangle$, $\langle B_2^c,\ B_2\rangle$, $\ldots\ldots\langle B_{3n}^c,\ B_{3n}\rangle$. Without revealing to the verifier, the prover randomly maps $3n$ nodes

   $N_1^R,\ \ldots\ldots N_n^R,\ N_1^W,\ \ldots\ldots N_n^W,\ N_1^B,\ \ldots\ldots N_n^B$ 1-1 into the $3n$ pairs of boxes. Each of the $(3n)!$ permutations mapping the $\{N_i^x\}$ on to the $\left\langle B_j^c,\ B_j\right\rangle$ is equally probable. The prover puts x (the color) into B_j^c, and puts i (the node number) into B_j. For every pair of number-containing boxes $(B_i,\ B_j)$, prepare a box labeled B_{ij}. This box contains 1 if the node of G in B_i has the color $B_i^c,$ the node of G in B_j has the color $B_j^{c'}$, and if the node in B_i is adjacent in G to the node in B_j and contains 0 otherwise. All boxes are then locked and presented to the verifier.

2. V: Upon receiving $(2 \cdot 3n + C_2^n)$ boxes, the verifier have two choices:

   (a) The verifier can request that the prover unlock all the boxes B_{ij} and all the numbers containing boxes B_i, but none of the colors containing B_i^c. In this way, the prover reveals the graph G without revealing its coloring. The verifier can check the boxes contain a correct description of G.

   (b) The verifier can request that the prover open the $3n$ boxes $\{B_i^c\}$ to reveal the colors they contain, and then open just those boxes B_{ij} such that B_i^c contains the same color as B_j^c. The opened boxes B_{ij} will all contain a 0 if and only if any 2 nodes that are colored the same are not adjacent in the graph represented by the boxes. The verifier can check the correct three-coloring.

3. P: The prover opens the appropriate boxes according to one of the two random requests from the verifier (either all boxes without colors or adjacent vertices with different colors)

4. V: The verifier accepts the proof if the prover complies and rejects otherwise.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: If the graph is three-colorable, those nodes that are colored the same are not adjacent in graph. So the verifier will complete all rounds and accept with probability 1

- Soundness: If the graph is not three-colorable, the prover's probability of convincing the verifier that he does know the graph is three colorable is $\frac{1}{2}$ in each round. His chance of convincing the verifier that he does know are $\leq 1/2^k$ when there are k rounds.

- Zero-knowledge: In each round, the verifier either obtains an encrypted graph or boxes containing "0". When the prover reveals all the boxes that describe the graph, it is only one of the random mappings of the n nodes of graph into n labeled boxes. When the prover reveals the boxes containing "0", the verifier gets no useful information without the corresponding numbers of nodes.

4.6. ZKP Sketch for SAT

The quadratic residuosity problem is the question of distinguishing by calculating the quadratic residues modulo n, where n is a composite number of two odd primes p and q. Under the assumption of quadratic residuosity that two states are computational indistinguishable through the calculation of the quadratic residues modulo n, a protocol for satisfiability (SAT) is suggested that directly simulates a circuit that evaluates given instances of SAT [26].

The general technique in [8] is through the simulation of an arbitrary boolean circuit without disclosing the inputs or any intermediary results. At the end of the protocol, if the final output of the circuit is 1, then the circuit is satisfiable, but nothing else.

Let $u = b_{1,} b_{2,} \ldots\ldots b_k$ be a k bit string of the prover. For each $1 \leq i \leq k$, let z_i and z_i' be the two encryptions of b_i randomly chosen by the prover. It is easy for the prover to convince the verifier that the k bit strings encrypted by $z_{1,}\ z_{2,}\ldots\ldots z_k$ and $z'_{1,}\ z'_{2,}\ldots\ldots z'_k$ are identical without providing the verifier with any additional information by the following string equality protocol.

Definition (String Equality Protocol): For each $i,\ 1 \leq i \leq k$, the prover gives the verifier some $x_i \in \mathbb{Z}^*_{\ltimes}$(denoting the set of integers relatively prime to n between 1 and $n-1$) so that $z_i z'_i \equiv x_i^2 (mod\, n)$.

Definition (Boolean Computation Protocol): Consider any boolean function $B : \{0,\ 1\}^t \rightarrow \{0, 1\}$ agreed upon between the prover and the verifier, and any bits $b_{1,}\ b_{2,}\ldots\ldots b_t$ only known to the prover. For $1 \leq i \leq t$, let z_i be an encryption of b_i known to the verifier. Let $b = B\left(b_{1,}\ b_{2,}\ldots\ldots b_t\right)$. The prover produces an encryption z for b and convinces the verifier that z encrypts the correct bit without giving the verifier any information on the input bits $b_{1,}\ b_{2,}\ldots\ldots b_t$ nor on the result b.

A permuted truth table for the boolean function B is introduced here, which is a binary string of length $(t+1)2^t$ formed of 2^t blocks of $t+1$ bits. The last bit of each block is the value of B on the other t bits of the block. Let s be the number of permutations agreed upon between the prover and the verifier.

1. P: The prover randomly chooses s permuted truth tables for B and discloses encryptions for each of them.

2. V: The verifier selects a random subset $X \subseteq \{1,\ 2,\ \ldots\ldots,\ s\}$ and sends it to prover as a challenge

3. P: The prover chooses one of the following options based on the request from the verifier.

   - For each $j \in X$, the prover opens the entire encryption of the j^{th} permuted truth table.

   - For each $j \notin X$, the prover points to the appropriate block in the encryption of the j^{th} permuted truth table and uses the following string equality protocol to convince the verifier that $z_{1,}\ z_{2,}\ldots\ldots z_t z$ encrypts the same bit string as this block.

4. V: The verifier makes the following verifications.

   - The verifier checks if it is a valid truth table for B.

   - The verifier checks if $z_{1,}\ z_{2,}\ldots\ldots z_t z$ encrypts the same bit string.

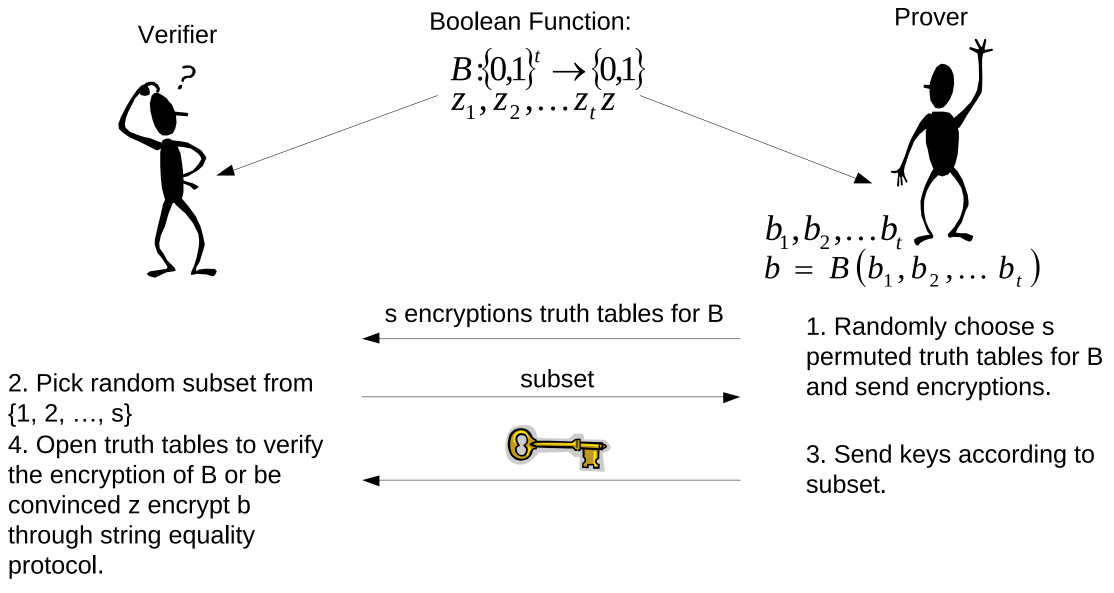

Verifier
Boolean Function:
$B:\{0,1\}^t \to \{0,1\}$
$z_1, z_2, \ldots z_t z$
Prover
$b_1, b_2, \ldots b_t$
$b = B(b_1, b_2, \ldots b_t)$
s encryptions truth tables for B
subset
1. Randomly choose s permuted truth tables for B and send encryptions.
2. Pick random subset from {1, 2, …, s}
3. Send keys according to subset.
4. Open truth tables to verify the encryption of B or be convinced z encrypt b through string equality protocol.

Figure 10. ZKP for Boolean Computation

Figure 10 illustrates the ZKP for boolean computation.

Based on the above discussions, a ZKP sketch has been designed for SAT. $f : \{0,\ 1\}^k \to \{0,\ 1\}$ is the function computed by some satisfiable boolean formula for which the prover knows that there is an assignment $b_{1,}\ b_{2,} \ldots\ldots b_k \in \{0,\ 1\}$ so that $f\,(b_{1,}\ b_{2,} \ldots\ldots b_k) = 1$. Assume the boolean formula is given using arbitrary unary and binary boolean operators. The prover will produce encryptions $z_{1,}\ z_{2,} \ldots\ldots z_k$ of $b_{1,}\ b_{2,} \ldots\ldots b_k$. Then the prover will guide the verifier through the encrypted evaluation of the formula, using the boolean computation protocol, one boolean operator at a time. The result will be a z which is the encryption for the value of $f\,(b_{1,}\ b_{2,} \ldots\ldots b_k)$. Then the prover opens z and shows the verifier that it encrypts a 1.

4.7. ZKP for Circuit Computations

In [26], it is possible to directly simulate the computation of any given computational device. Simulation is used to separate the data encryption from the encryption of the device's structural (or control) information. It directly proves the result of the computation, avoiding the Karp reduction from a specific NP-complete problem.

In the protocol, the essential idea is that the prover constructs and sends to the verifier a copy of a simulation of the computing device. This copy should include encoding of the possible input, intermediate results, and output data. In addition, it includes encoding of structural information about the computing device. Upon receiving the encoding information, the verifier chooses a bit $\alpha \in \{0,\ 1\}$ and sends it back.

- With probability $\frac{1}{2}$ the verifier decides to verify, that is to request that the prover open all the encryptions in the copy. In this way, the verifier can check if the construction is a legal one with the correct structural information.
- With probability $\frac{1}{2}$ the verifier chooses to compute, in which case the prover opens only the result of the computation. In order to prove the output presented is in fact the computed result, the prover opens only parts that are involved in the computation, while other information is left encrypted. The unopened information appears random.

The process is repeated r times; each time the verifier either chooses to verify or chooses to compute. If all verifications are successful, and all the computations produce a connection to the same opened output, then the verifier accepts the result of the computation.
The prover uses an encryption procedure E and simulates the computation of a circuit C in a minimum-knowledge fashion with the verifier. Let E(C) denote the encryption.

1. P: The prover probabilistically encrypts C using an encryption procedure E. Then the prover sends E(C) to the verifier.
2. V: The verifier chooses a bit $\alpha \in \{0,\ 1\}$ and sends it back.
3. P: The prover chooses one of the corresponding actions based on the requests from the verifier.
    - $\alpha = 0$ {verify}: the prover opens all the encryptions of all gates in E(C) and sends the cleartext circuit to the verifier

- $\alpha = 1$ {compute}: the prover opens only the pointers of the specific computation an sends the cleartexts of these pointers to the verifier including the outputs

4. V: The verifier does the corresponding verifications.

- $\alpha = 0$, the verifier verifies that C is properly encrypted;
- $\alpha = 1$, the verifier verifies that the opened pointers lead from the (unopened) input entries to the output pointers.

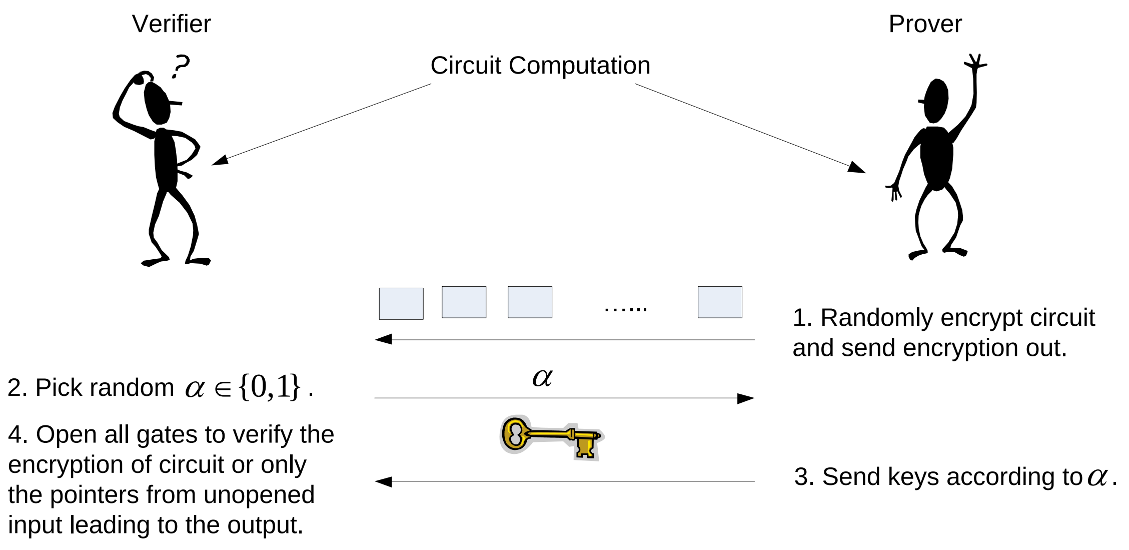

Verifier
Prover
Circuit Computation
1. Randomly encrypt circuit and send encryption out.
2. Pick random $\alpha \in \{0,1\}$.
α
3. Send keys according to α.
4. Open all gates to verify the encryption of circuit or only the pointers from unopened input leading to the output.

Figure 11. ZKP for Circuit Computation

Figure 11 illustrates the ZKP for circuit computation.

If all computations give the same value and all verifications are successful then the verifier accepts. Otherwise, the verifier rejects.

4.8. ZKP for Exact Cover

Definition (Exact Cover): A set $\{u_i,\ i = 1,\ 2,\ \ldots\ldots t\}$. A family $\{S_j\}$ is a subset of the set $\{u_i,\ i = 1,\ 2,\ \ldots\ldots t\}$. Exact cover means there is a subfamily $\{T_h\} \subseteq \{S_j\}$, such that sets T_h are disjoint and $\bigcup T_h = \bigcup S_j = \{u_i,\ \ i = 1,\ 2,\ \ldots\ldots t\}$.

The following protocol will be executed for t rounds. The verifier uses a coin toss in each round.

Let $|T_h| = m$, $|S_j| = n$

1. P: The prover prepares the following four kinds of boxes:

    - t labeled boxes $B_{1,}\ B_{2,}\ \ldots\ldots B_t$ which will contain set $\{u_i,\ \ i = 1,\ 2,\ \ldots\ldots t\}$ in a random order
    - n labeled boxes $A_{1,}\ A_{2,}\ \ldots\ldots A_n$ which will contain element in S_j
    - $n \cdot t$ black boxes in a matrix as follows, if the set member in B_i is in the S_j of A_j, then insert a 1; otherwise insert a 0.

$$\begin{array}{ccccccc} & B_1 & B_2 & B_3 & \ldots & \ldots & B_t \\ A_1 & 1 & 0 & 0 & \ldots & \ldots & 0 \\ A_2 & 0 & 0 & 1 & \ldots & \ldots & 0 \\ \ldots & 0 & 1 & 0 & \ldots & \ldots & 0 \\ \ldots & 0 & 0 & 0 & \ldots & 1 & 0 \\ A_n & 1 & 0 & 0 & \ldots & \ldots & 1 \end{array}$$

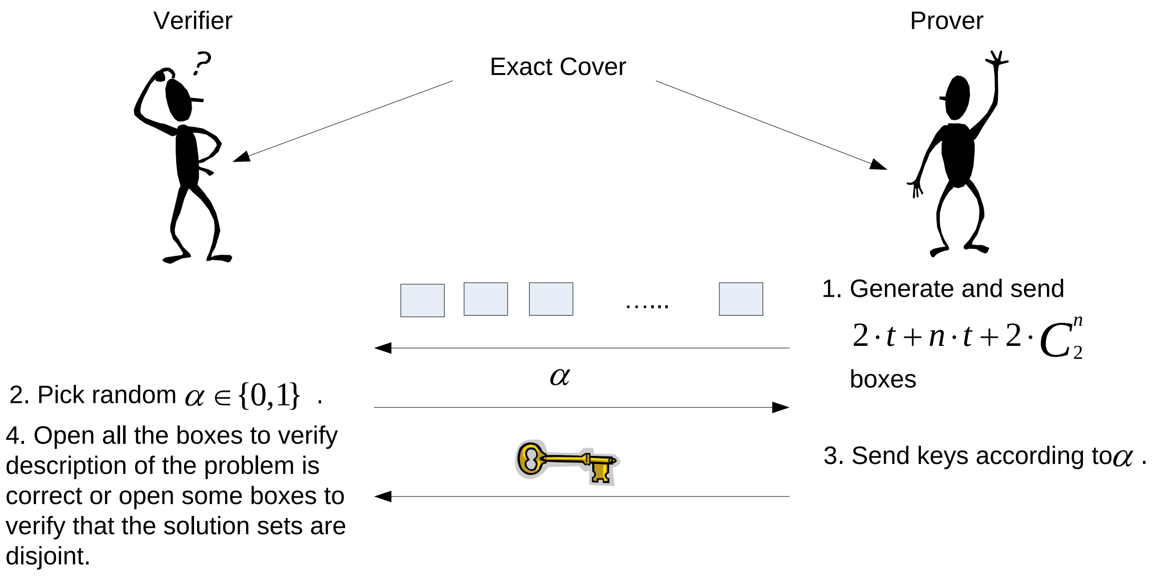
    - For every two rows in matrix, a pair of boxes is prepared as $< Index * Index, Result >$. The first box contains indexes of two rows. The second box contains the product operation of any two row vectors (one is row vector, another is the transpose of row vector). If the result of product operation is 0, $Result$ = 0; otherwise, $Result$ = 1.

2. V: Upon receiving $(2 \cdot t + n \cdot t + 2 \cdot C_2^n)$ black boxes, the verifier has two choices:

    - $(\alpha = 0)$ The verifier can ask to open all the boxes to verify if the description of the problem is correct.
    - $(\alpha = 1)$ The verifier asks the prover to open some pairs of boxes with $Result$ = 0. With the index showing in these boxes, the verifier also asks to open the corresponding row in the matrix to check if $\bigcup T_h = \bigcup S_j = \{u_i,\ \ i = 1,\ 2,\ \ldots\ldots t\}$.)

3. P: The prover opens the appropriate boxes according to the two random requests from the verifier (either all the boxes describing the exact cover problem or disjoint solution sets).

4. V: The verifier accepts the proof if the prover complies and reject otherwise.



Verifier
Prover
Exact Cover
1. Generate and send $2 \cdot t + n \cdot t + 2 \cdot C_2^n$ boxes
2. Pick random $\alpha \in \{0,1\}$.
α
3. Send keys according to α .
4. Open all the boxes to verify description of the problem is correct or open some boxes to verify that the solution sets are disjoint.

Figure 12. ZKP for Exact Cover

Figure 12 illustrates the ZKP for exact cover.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: If the prover knows the exact cover, he can encrypt the correct information in the matrix. So the verifier will complete all rounds and accept with probability 1.

- Soundness: If the prover doesn't know the exact cover, the prover's chance of convincing the verifier that he does know the exact cover is $\frac{1}{2}$ in each round: either the exact cover problem is correctly encrypted or there are disjoint solution sets. His chance of convincing the verifier that he does know the exact cover are $\leq 1/2^k$ when there are k rounds.

- Zero-knowledge: In each round, the verifier either obtains a random matrix with "0" and "1" or obtains one element in set $\{S_j\}$ which is known to the verifier beforehand. Thus, there is no useful information transferred.

An example will be used to illustrate ZKP for exact cover , suppose: $S = \{A, B, C, D, E, F\}$. $\{u_i\} = \{1, 2, 3, 4, 5, 6, 7\}$. $A = \{1, 4, 7\}, B = \{1, 4\}, C = \{4, 5, 6\}, D = \{3, 5, 6\}, E = \{2, 3, 5, 6\}, F = \{2, 7\}$.

In ZKP for exact cover, the prover will prepare seven black boxes named $B_1, B_2, ...B_7$, which will contain elements in $\{u_i\}$ in a random order. Additional six black boxes named $A_1, A_2, ...A_6$ will be used to contain sets in $\{S_i\}$. With A_i, B_i, the following matrix will be prepared.

$$\begin{array}{cccccccc} & & 2 & 1 & 6 & 5 & 3 & 7 & 4 \\ & & B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 \\ S_6 & A_1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ S_1 & A_2 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ S_4 & A_3 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ S_2 & A_4 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ S_3 & A_5 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ S_5 & A_6 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}$$

The prover also prepares following pair of boxes which contain indexes and result of * operation of any two rows:

1*2, 1	1*3, 0	1*4, 0	1*5, 1	1*6, 1
2*3, 0	2*4, 1	2*5, 1	2*6, 0	3*4, 0
3*5, 0	3*6, 1	4*5, 1	4*6, 0	5*6, 1

Table 1. Black-Boxes for Index and Product Operation

The prover will send all of those black boxes to the verifier. Upon receiving $(2 \cdot 7 + 6 \cdot 7 + 2 \cdot 15) = 86$ black boxes, the verifier has two choices:

- The verifier asks the prover to open all black boxes to verify the boxes contain the public exact cover problem.

- The verifier asks the prover to open some pair of boxes with $Result = 0$. The prover will open (13, 0) and (34, 0). With index 1, 3 and 4, the verifier ask the prover to open row 1, 3 and 4 in the matrix and check if elements in row 1, 3 and 4 are combined together covering all elements in the set $\{u_i\}$

4.9. ZKP for 0-1 Knapsack

Definition (0-1 Knapsack): There are n kinds of items. Each kind of item i has a value v_i and a weight w_i. The maximum weight that a bag can carry is W. The 0-1 knapsack algorithm determines the subsets of items which give maximum value without exceeding the maximum weight of the bag. Mathematically, the 0-1 knapsack problem can be formulated as:

- Maximize $\sum_i^n v_i \cdot x_i$ with $x_i \in \{0, 1\}$
- Subject to $\sum_i^n w_i \cdot x_i \leq W$ with $x_i \in \{0, 1\}$

The simple case of the problem with the property that for each kind of item the weight equals the value $w_i = v_i$ is as follows.
The 0-1 simple knapsack problem is to find a binary n-vector x such that a given W equals to $w \cdot x$. A supposed solution x is easily checked in at most n additions, but finding a solution is believed belong to NPC. As n is larger than 100 or 200, the number of operations grows exponentially and computationally infeasible for exhaustive trial and error search over all 2^n possible x.
However, the degree of difficulty is crucially dependent on the choice of w. If $w = (1, 2, 4, ..., 2^n)$, then solving for x is equivalent to finding the binary representation of W. Generally, if for all i, $w_i > \sum_{j=1}^{i-1} w_j$, then x is also easily found. $x_n = 1$ if and only if $W \geq w_n$ and for $i = n-1, n-2, ..., 1$, $x_i = 1$ if and only if $W - \sum_{j=i+1}^{n} x_j \cdot w_j \geq w_i$. A trapdoor knapsack is defined as one in which carefully choice of w as above. They form a proper subset of all possible knapsacks and their solutions are not as difficult as the hardest knapsacks in NP theory [34]. In this paper, we are not considering trapdoor knapsacks.

According to the Karp's reduction theorem, the ZKP for knapsack problem can be generated in a similar way as the ZKP for exact cover problem.

The following protocol will be executed for t rounds. The verifier uses a coin toss in each round.

1. P: The prover prepares the following four kinds of boxes:

   - n labeled boxes $B_{1,}\ B_{2,}\ \ldots\ldots B_n$ which will contain items $w_1, w_2, \ldots\ldots, w_n$ in a random order
   - m boxes $S_1, S_2, \ldots\ldots S_m$ will contain some numbers which are summation of random items from w_i
   - $m \cdot n$ black boxes in a matrix as follows, if the item B_i is included in sum S_i, then insert a 1; otherwise insert a 0.

$$\begin{array}{ccccccc} & B_1 & B_2 & B_3 & \ldots & \ldots & B_n \\ S_1 & 1 & 0 & 0 & \ldots & \ldots & 0 \\ S_2 & 0 & 0 & 1 & \ldots & \ldots & 0 \\ \ldots & 0 & 1 & 0 & \ldots & \ldots & 0 \\ \ldots & 0 & 0 & 0 & \ldots & 1 & 0 \\ S_m & 1 & 0 & 0 & \ldots & \ldots & 1 \end{array}$$

   - For every two rows in matrix, a pair of boxes is prepared as $< Index * Index, Result >$. The first box contains indexes of two rows. The second box contains the product operation of any two rows vectors (one is row vector, another is the transpose of row vector). If the result of product operation is 0, $Result = 0$; otherwise, $Result = 1$.

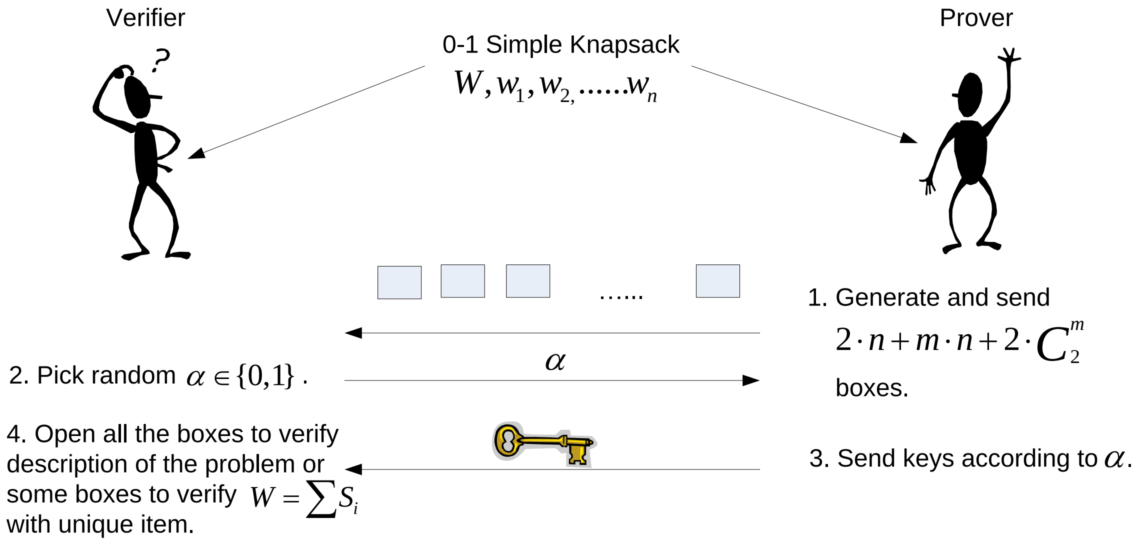
2. V: Upon receiving $(2 \cdot n + m \cdot n + 2 \cdot C_2^m)$ black boxes, the verifier has two choices:

   - $(\alpha = 0)$ The verifier asks the prover to open all the boxes to verify if they are satisfied the original simple knapsack problem.
   - $(\alpha = 1)$ The verifier asks the prover to open a subset of the summation boxes S_i with $\sum S_i = W$. The verifier also asks to open all the pairs of boxes to check if $Result = 0$

within the subset of S_i.

3. P: The prover opens the appropriate boxes according to the two random requests from the verifier (either all the boxes describing the knapsack problem or those boxes where W is equal to some summation S_i with $Result = 0$ for each pair).

4. V: The verifier accepts the proof if the prover complies and rejects otherwise.



Verifier
Prover
0-1 Simple Knapsack
$W, w_1, w_{2,} ......w_n$
1. Generate and send $2 \cdot n + m \cdot n + 2 \cdot C_2^m$ boxes.
......
2. Pick random $\alpha \in \{0,1\}$.
α
3. Send keys according to α.
4. Open all the boxes to verify description of the problem or some boxes to verify $W = \sum S_i$ with unique item.

Figure 13. ZKP for 0-1 Simple Knapsack

Figure 13 illustrates the ZKP for 0-1 simple knapsack. If the verifier has completed t iterations of the above steps, then he accepts.

This protocol is a ZKP because it upholds the properties of completeness, soundness and zero-knowledge.

- Completeness: If the prover knows the solution to the simple knapsack, he can encrypt the correct information in all the boxes. So the verifier will complete all rounds and accept with probability 1.

- Soundness: If the prover doesn't know the solution to the simple knapsack, the prover's chance of convincing the verifier that he does know the solution is $\frac{1}{2}$ in each round: either the simple knapsack problem or some summation equal to constraint W. His chance of convincing the verifier that he does know the exact cover are $\leq 1/2^k$ when there are k rounds.

- Zero-knowledge: In each round, the verifier either obtains a random matrix with "0" and "1" or obtains some random numbers. Thus, there is no useful information transferred.

5. Advanced Topics in Composing ZKPs

5.1. Composing ZKPs

ZKPs have many applications in cryptographic areas such as identity verification, authentication, key exchange and so on. Provided that one-way functions exist, ZKPs can be used to force parties to behave according to a predetermined protocol, such as multi-party secure computations [18]. The next natural question is whether the zero-knowledge condition is preserved under a variety of composition operations: sequential, parallel and concurrent. This question not only belongs to the theoretical area, but is crucial to the application of ZKPs in the cryptographic area. For example, we want to make sure the information obtained by the verifier during the execution of a ZKP will not enable him to extract any additional knowledge from subsequent executions of the same protocol.

For $T \in \{sequential, parallel, concurrent\}$, a protocol is T-zero-knowledge if it is zero-knowledge under a composition of type T [22]. The definitions of T-zero-knowledge are derived by considering that the verifiers can initiate a polynomial number of interactions with the prover using scheduling type T.

Protocol repetition, which repeats the basic protocol many times independently and links them together, is mainly used for error reduction. Protocol composition is different than protocol repetition; the prover is not assumed to coordinate its actions between different executions of the protocol [38]. If the verifier has some misbehavior in protocol repetition, the prover can refuse to continue execution. By contrast, in protocol composition, the prover is still obilgated to continue interaction with other executions of the protocol. The obvious way to perform repetition would be to execute the basic protocol sequentially, but it suffers from a high round-complexity, resulting in a high number of message exchanges in the execution. Parallel repetition that conducts the basic protocol in parallel is used as a way of decreasing the error probability while maintaining the number of rounds of message.

1. In sequential composition, the ZKP is invoked (polynomially) many times, where each invocation follows the termination of the previous one [22]. Assuming a basic ZKP contains four-round $P1, V1, P2, V2$ message passing, Figure 14 shows the sequential composition. It is the most basic case of protocol composition. Every protocol that is zero-knowledge is sequential-zero-knowledge, which means zero-knowledge is closed under sequential composition.

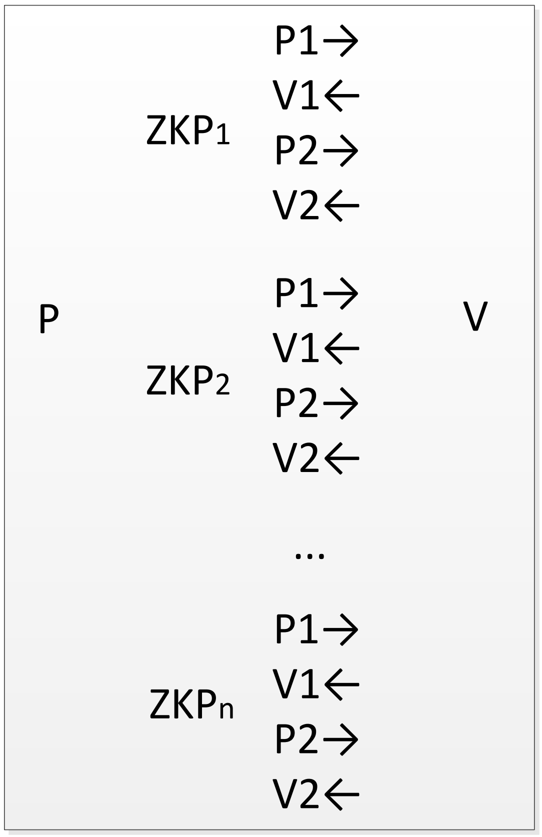
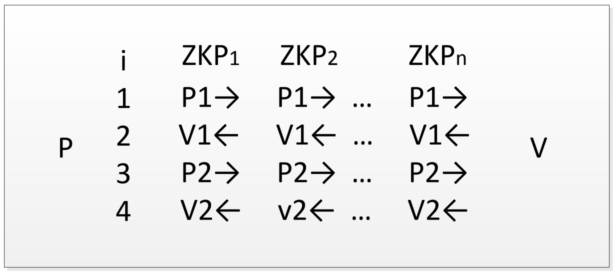


Figure 14. Sequential Composition of ZKP

2. In parallel composition, (polynomially) many instances of the ZKP are invoked at the same time and proceed at the same pace [22]. Under the assumption of a synchronous model of communication, the executions are totally synchronized so that the ith messages in all instances are sent exactly at the same time. As shown in Figure 15, assuming a basic ZKP contains four-round $P1(i = 1), V1(i = 2), P2(i = 3), V2(i = 4)$ message passing, the ith message in all ZKPs should be sent out at the same time.



	i	ZKP1	ZKP2		ZKPn	
	1	P1→	P1→	…	P1→	
P	2	V1←	V1←	…	V1←	V
	3	P2→	P2→	…	P2→	
	4	V2←	v2←	…	V2←	

Figure 15. Parallel Composition of ZKP

Goldreich and Krawczyk in [17] presented a simple protocol that is zero-knowledge, but is not closed under parallel composition. The example is described in Table 2:

Consider the prover holding a secret and a random function $f : \{0,1\}^{2n} \rightarrow \{0,1\}^n$, and willing to participate in the protocol. The verifier is supposed to send the prover a binary value $\{0,1\}$:
For choice 0, the prover uniformly selects $\alpha \in \{0,1\}^n$, and sends it to the verifier which is supposed to reply with a pair of n-bit long strings, denoted (β, γ). The prover checks whether or not $f(\alpha\beta) = \gamma$. The input of f function, $\alpha\beta$, is the concatenation operation of two n-bit string. If the equality is satisfied, the prover sends the verifier the secret he has. For choice 1, the verifier is supposed to uniformly select $\alpha \in \{0,1\}^n$, and send it to the prover, which selects uniformly $\beta \in \{0,1\}^n$, and replies with the pair $(\beta, f(\alpha\beta))$.

Table 2. Protocol Not Closed Under Parallel Composition

The prover's strategy is zero-knowledge because:

- If the verifier chooses 0, then it is infeasible for the verifier to guess a passing pair (β, γ) since the α is randomly chosen by the prover. The verifier doesn't obtain anything from the interaction.
- If the verifier chooses 1, then it obtains a pair $(\beta, f(\alpha\beta))$. Since β is selected by the prover, for any α, the value $f(\alpha\beta))$ is random to the verifier and $(\beta, f(\alpha\beta))$ is indistinguishable from a uniformly selected pair of n-bit long strings.

However, if the verifier conducts two concurrent executions with the prover, there will be information leakage, illustrated in Table 3. In this case, the verifier chooses 0 in one session and 1 in another session. Upon receiving the prover's message α in the first session, the verifier sends α as its own message in the second session and obtains a pair $(\beta, f(\alpha\beta))$ from the prover's execution of the second session. Then the verifier sends the pair $(\beta, f(\alpha\beta))$ to the first session of the prover, which will satisfy the equation $f(\alpha\beta) = \gamma$ and the verifier will obtain the desired secret.

V = 0	V = 1
$P \overset{\alpha}{\rightarrow} V$	
	$P \overset{\alpha}{\leftarrow} V$
	$P \overset{(\beta, f(\alpha\beta))}{\rightarrow} V$
$P \overset{(\beta,\gamma)}{\leftarrow} V$	
$P \overset{secret}{\rightarrow} V\ (f(\alpha\beta) == \gamma)$	

Table 3. Parallel Composition Leads to Information Leakage

Based on the above example, zero-knowledge is not closed under parallel composition in general [38]. However, assuming one-way functions exist, it has been proven that there exists zero-knowledge protocols for NP that are closed under parallel composition and having a constant number of rounds [15]. Conceptually, the independently execution of one basic protocol will give no information to other execution of basic protocol since the prover will respond independently based on message from each execution.

Using a simulator is the technique to demonstrate the zero-knowledge property. The idea behind it is that whatever the verifier might have learned from interacting with the prover, the verifier could learn by himself by running the simulator. The simulator has two advantages over the prover to compensate for not knowing the secret as the prover does [5]. The first advantage is that the simulator knows and can determine the questions the verifier will ask. The second advantage is that the simulator has many choices to answer the verifier's questions. After it fails to answer a question from the verifier, the simulator simply choose not to output this interaction and goes back to some point to output only the successful interaction. This process is called rewind which helps the simulator to rewind back from the failure and try again. The black-box simulator, which simulates the interacting of the prover with the verifier without access to the strategy of that verifier, is used to demonstrate the zero-knowledge [15].

3. Concurrent composition is a more general protocol composition, which was first considered by Dwork, Naor and Sahai [10]. Here (polynomially) many instances of the protocol are invoked at an arbitrary time and proceed at an arbitrary pace [22]. This allows one or multiple verifiers to engage in many proofs with the prover and arbitrarily interleave the messages by running some of the protocols ahead in order to gain information that will enable it to attack some of the other protocols. Within each proof, the verifier must follow the proper order of the steps. Among different proofs, the verifier can interleave arbitrarily. For example, the verifier may execute the first step of proof 1, then execute all steps of proof 2 in order to obtain some knowledge in advance to execute the remaining steps in proof 1. Concurrent composition is shown in Figure 16.

   There are two models of concurrent composition: a purely asynchronous model and an asynchronous model with timing. The purely asynchronous model is a simpler model and requires no assumptions about the underlying communication channels. While an asynchronous model with timing, each party is assumed to hold a local clock such that the relative clock rates are bounded by an *a priori* known constant.

   The timing model assumes that each party holds a local clock such that the relative clock rates are bounded by an *a priori* known constant [10]. There are two constants involved in the timing

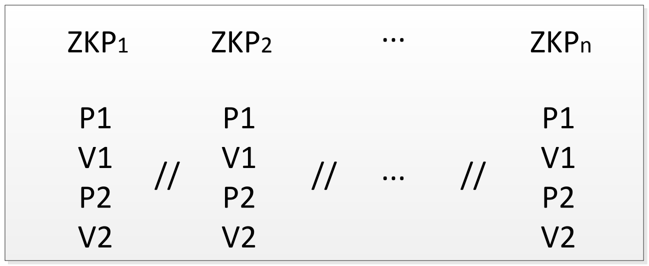


Figure 16. Concurrent Composition of ZKP

assumption: ρ and Δ. ρ is a global time bound on the relative rates of the local clock and is known to all parties. Δ is an upper bound on the message handling and delivery time. Under this timing model, a constant-round concurrent ZKP can be constructed as shown in [10].

Some problems will arise if time-driven operations are considered. The use of time-driven operations, such as timeout of incoming messages, receipt and delay of outgoing messages, will slow down the execution of the protocol. However, in the absence of more appealing alternatives, the use of this timing model still is considered reasonable [22].

5.1.1 The Richardson-Kilian Concurrent ZKPs Protocol

Under standard intractability assumptions, concurrent ZKPs exist for NP in a purely asynchronous model by Richardson and Kilian (RK) [37]. In this model, a verifier is allowed to run up to k interactive proofs with the prover. The RK protocol consists of two stages: an $O(k)$ preamble messages and a main body. The first stage is independent of the common input, in which the verifier and the prover will be engaged in $O(k)$ message exchanges. First, the verifier commits to k random n-bit strings $v_1, v_2, ..., v_k \in \{0,1\}^n$, where n is the security parameter of the protocol that represents the number of concurrent executions and k is the parameter that determines the number of rounds. In the following k iterations between the prover and verifier, the prover commits to a random n-bit string, p_j, and the verifier decommits to the corresponding v_j. In the second stage, the prover provides proof either for the statement is true or for some $j \in \{1, 2, ..., k\}$ $v_j = p_j$. Table 4 illustrates two stages.

First Stage:
$V \to P$: Commit to $v_1, v_2, \cdots, v_k$
For $j = 1, 2, \cdots, k$
$P \to V$: Commit to p_j
$V \to P$: Decommit to v_j

Second Stage:
$P \leftrightarrow V$: Zero Knowledge Proof for statement is true or $\exists j\ s.t.\ v_j = p_j$

Table 4. RK Protocol

The technique used here is from WI in section 2.7: instead of proving a statement T, the prover proves a weaker theorem $T \vee W$, where W is a statement that will fail with extremely high probability [37]. In the actual interactions, there is little chance for the prover to guess v_j from the verifier. Thus, the prover has no choice but to provide the proof for the statement T.

It is necessary to examine whether the zero-knowledge property is still preserved in the protocol. Recall that in order to demonstrate the zero-knowledge property, a simulator has to be generated that can simulate the views of every polynomial-time adversary interacting with the prover. Under concurrent composition, the simulator's task becomes more complicated [38]. In the RK protocol, the preamble messages will be used by the simulator to its advantage. Whenever the simulator may cause $v_j = p_j$ to happen for some j, it can simulate the rest of the protocol and provide a WI proof. The larger the number of rounds in the preamble, the more resistant the result is to concurrent attacks, and the easier the simulation task is. However, the number of rounds in the protocol will also increase.

5.1.2 The Improved Concurrent ZKPs Protocol

Prabhakharan, Rosen and Sahai (PRS) proposed a more sophisticated concurrent ZKP in [36] with $O(\alpha(n) \cdot logn)$ rounds of iteration, where $\alpha(\cdot)$ is any super-constant function such as $loglogn$. The PRS protocol also consists of two stages, which is illustrated in Table 5. In the first stage, the verifier will commit to a random n-bit string σ, and to two sequences $\{\sigma^0_{i,j}\}^k_{i,j=1}$, and $\{\sigma^1_{i,j}\}^k_{i,j=1}$. Each of the sequences consists of k^2 random n-bit strings, such that for every i, j the value of $\sigma^0_{i,j} \oplus \sigma^1_{i,j} = \sigma$. The total number of committed strings in this step is $2 \cdot k^2 + 1$. This is followed by k iterations between the prover and the verifier. In the jth iteration, the prover will send a random k-bit string, $r_j = r_{1,j}, r_{2,j}, ..., r_{k,j}$, and the verifier will decommit k strings of $\sigma_{1,j}^{r_{1,j}}, \sigma_{2,j}^{r_{2,j}}, ..., \sigma_{k,j}^{r_{k,j}}$. After first stage, the verifier has opened a total of k^2 strings in the two sequences.

In the second stage, the prover and the verifier will engage in a three-round protocol. After the prover commits secret values about the statement to the verifier, the challenge question σ will be decommitted with all the σ, $\{\sigma_{i,j}^{1-r_{i,j}}\}_{i,j=1}^{k}$ that were not revealed in the first stage. Upon checking that the strings satisfy the constraint ($\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$), the prover decommits the corresponding value to the verifier.

First Stage:
$V \rightarrow P$: Commit to σ, $\{\sigma_{i,j}^0\}_{i,j=1}^{k}$, $\{\sigma_{i,j}^1\}_{i,j=1}^{k}$, for each i, j, $\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$.
For $j = 1, 2, \cdots, k$:
$P \rightarrow V$: Send k-bit string $r_j = r_{1,j}, r_{2,j}, ..., r_{k,j}$
$V \rightarrow P$: Decommit k strings of $\sigma_{1,j}^{r_{1,j}}, \sigma_{2,j}^{r_{2,j}}, ..., \sigma_{k,j}^{r_{k,j}}$

Second Stage:
$P \Rightarrow V$: Perform n independent copies of commitments of secrets
$V \Rightarrow P$: Decommit to σ and to $\{\sigma_{i,j}^{1-r_{i,j}}\}_{i,j=1}^{k}$
$P \Rightarrow V$: Answer according to the value of σ if $\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$ is satisfied

Table 5. PRS Protocol

Both the RK and the PRS protocol follows the same structure of adding a preamble phase to the basic protocol. This phase is only used for sucessful simulation to prove the zero-knowledge property. Applying this protocol, Blum's Hamiltonian Cycle concurrent zero-knowledge proof has been demonstrated. Thus, every language in NP has a concurrent zero-knowledge proof system. A concurrent ZKP for the 0-1 simple Knapsack of Table 6 that is shown in section 4.9, yields the PRS protocol in Table 7.

A detailed description is given as follows. In the first stage, the verifier will commit an n-bit random string σ and $2k^2$ n'-bit random strings as shown in Table 8 with the restriction that $\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$ as shown in Table 9:

For the next $2k$ iterations, the verifier will decommit k strings in each iteration to the prover as shown in Table 10:

Common input: n items, each item i has a weight w_i and a weight constraint W.
P1: The prover prepares all the black boxes.
V1: Upon receiving $(2 \cdot n + m \cdot n + 2 \cdot C_2^m)$ black boxes, the verifier has two choices:
$(\alpha = 0)$ The verifier asks the prover to open all the boxes to verify if they satisfy the original simple knapsack problem.
$(\alpha = 1)$ The verifier asks the prover to open a subset of the summation boxes S_i with $\sum S_i = W$. The verifier also asks to open all the pair of boxes to check if $Result = 0$ within the subset of S_i.
P2: The prover opens the appropriate boxes according to the two random requests from the verifier (either all the boxes describing the knapsack problem or whose boxes where W is equal to some summation S_i with unique item).
V2: The verifier accepts the proof if the prover complies and rejects otherwise.

Table 6. Basic ZKP for 0-1 Simple Knapsack

Common input: n items, each item i has a weight w_i. A parameter k is used for determining the number of rounds and also works as the security parameter that represents the number of concurrent executions.
First stage: $2k + 2$ rounds interactions between the prover and the verifier (independent of the knapsack problem)

Verifier's preliminary step: the verifier selects and commits to an n'-bit string σ and two sequences of n'-bit strings: $\{\sigma_{i,j}^0\}_{i,j=1}^k$, $\{\sigma_{i,j}^1\}_{i,j=1}^k$, for each i, j, $\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$. The total committed strings are $2k^2 + 1$.
For $j = 1, 2, \cdots, k$:
Prover's jth step: Send k-bit string $r_j = r_{1,j}, r_{2,j}, ..., r_{k,j}$ to the verifier
Verifier's jth step: Decommit k strings of $\sigma_{1,j}^{r_{1,j}}, \sigma_{2,j}^{r_{2,j}}, ..., \sigma_{k,j}^{r_{k,j}}$

Second stage: The prover and verifier engage in n' parallel executions of Basic ZKP for the 0-1 simple knapsack
$\hat{P}1$: The prover will prepare boxes for n' parallel independent executions, which means there will be $n' \cdot (2 \cdot n + m \cdot n + 2 \cdot C_2^m)$ black boxes. Each set of $(2 \cdot n + m \cdot n + 2 \cdot C_2^m)$ black boxes is generated independently and contains the different commitment of the same knapsack problem.
$\hat{V}1$: The verifiers decommit to σ and remaining $\{\sigma_{i,j}^{1-r_{i,j}}\}_{i,j=1}^k$, which are not revealed in the first stage.
$\hat{P}2$: The prover checks that the verifier has properly decommitted to the value by checking $\sigma_{i,j}^0 \oplus \sigma_{i,j}^1 = \sigma$. Each bit in σ is the question from the verifier for each execution. If so, the prover opens the appropriate boxes according to each bit value of σ for each execution(either all the boxes within one set which describes the knapsack problem correctly or under one set W is equal some summation of S_i with unique item) .
$\hat{V}2$: The verifier conducts the verification of the prover's proofs based on the result of $\hat{P}2$.

Table 7. PRS Concurrent ZKP for 0-1 Simple Knapsack

$\sigma_{1,1}^{0}$	$\sigma_{1,2}^{0}$	...	$\sigma_{1,k}^{0}$	$\sigma_{1,1}^{1}$	$\sigma_{1,2}^{1}$	...	$\sigma_{1,k}^{1}$
$\sigma_{2,1}^{0}$	$\sigma_{2,2}^{0}$	...	$\sigma_{2,k}^{0}$	$\sigma_{2,1}^{1}$	$\sigma_{2,2}^{1}$	...	$\sigma_{2,k}^{1}$
...	...	...	...	...	...	...	...
$\sigma_{k,1}^{0}$	$\sigma_{k,2}^{0}$	...	$\sigma_{k,k}^{0}$	$\sigma_{k,1}^{1}$	$\sigma_{k,2}^{1}$	...	$\sigma_{k,k}^{1}$

Table 8. $2k^2$ Randomly Generated Strings

$\sigma_{1,1}^{0} \oplus \sigma_{1,1}^{1} = \sigma$	$\sigma_{1,2}^{0} \oplus \sigma_{1,2}^{1} = \sigma$	...	$\sigma_{1,k}^{0} \oplus \sigma_{1,k}^{1} = \sigma$
$\sigma_{2,1}^{0} \oplus \sigma_{2,1}^{1} = \sigma$	$\sigma_{2,1}^{0} \oplus \sigma_{2,2}^{1} = \sigma$	...	$\sigma_{2,k}^{0} \oplus \sigma_{2,k}^{1} = \sigma$
...	...	...	...
$\sigma_{k,1}^{0} \oplus \sigma_{k,1}^{1} = \sigma$	$\sigma_{k,2}^{0} \oplus \sigma_{k,2}^{1} = \sigma$	...	$\sigma_{k,k}^{0} \oplus \sigma_{k,k}^{1} = \sigma$

Table 9. Constraints on Strings

In the second stage, the prover will make independent copies of $P1$ in Table 6 with n' parallel verifiers. The verifier will decommit the σ and remaining k^2 $\sigma_{i,j}^{1-r_{i,j}}$ to the prover. The prover will execute n' copies of $P2$ in Table 6 if the k^2 strings in the first stage and k^2 strings in the second stage satisfy $\sigma_{i,j}^{0} \oplus \sigma_{i,j}^{1} = \sigma$. The verifier will accept only if all the conditions are satisfied.

In the actual execution of the protocol, since the prover does not know the value of σ, the protocol would be a proof system for 0-1 simple knapsack with soundness error $\frac{1}{2^{n'}}$. The sole purpose of the first stage is to allow the simulator to know the value of σ. As long as the simulator makes the verifier reveal both $\sigma_{i,j}^{0}$ and $\sigma_{i,j}^{1}$ for some i, j, it can simulate the rest of the protocol by adjusting the $P1$ in Table 6 according to the value of $\sigma = \sigma_{i,j}^{0} \oplus \sigma_{i,j}^{1}$.

($P1$): The prover will send a random k-bit string $r_1 = r_{1,1}, r_{2,1}..., r_{k,1}$
($V1$): The verifier will decommit $\sigma_{1,1}^{r_{1,1}}, \sigma_{2,1}^{r_{2,1}}, ..., \sigma_{k,1}^{r_{k,1}}$
($P2$): The prover will send a random k-bit string $r_2 = r_{1,2}, r_{2,2}..., r_{k,2}$
($V2$): The verifier will decommit $\sigma_{1,2}^{r_{1,2}}, \sigma_{2,2}^{r_{2,2}}, ..., \sigma_{k,2}^{r_{k,2}}$
(...)
(Pk): The prover will send a random k-bit string $r_k = r_{1,k}, r_{2,k}..., r_{k,k}$
(Vk): The verifier will decommit $\sigma_{1,k}^{r_{1,k}}, \sigma_{2,k}^{r_{2,k}}, ..., \sigma_{k,k}^{r_{k,k}}$

Table 10. Preamble Messages

5.2. Efficiency Considerations

Round-complexity is considered as a very important complexity measure for cryptographic protocols. It indicates the number of message exchanges taking place in the protocol. Typically, a constant number of rounds will be a desirable result. Under the assumptions of the existence of one-way functions, ZKPs can be constructed in constant number of rounds by Feigh and Shamir [13], Brassard, Crepeau and Yung [9], and Goldreich and Kahan [16] [22].

In [16], Goldreich and Kahan presented how to reduce the error probability of interactive proofs that having constant error probability without increasing the round-complexity and while preserving their zero-knowledge property. They are SZKP whose soundness condition requires that nobody, even with unbounded computational ability, can fool the verifier into accept the false statements except with negligible probability.

Recall that commitment schemes are commonly used to construct ZKPs with two phases. The first phase is called commit. The verifier will not know which value it is if the prover commits a value. The second phase is called reveal. The prover will send some auxiliary information that allows the verifier to reveal the uniquely committed value in order to assure that it is the original value. Two properties named secrecy and nonambiguity are involved [16]:

- Secrecy: At the end of the commit phase, the verifier does not gain any information of the prover's value.

- Nonambinguity: In the reveal phase, with the transcript of the interaction in the commit phase, there exists only one value that the verifier may accept as a legal reveal of the commitment.

Two different commitment schemes have been presented to construct constant-round ZKPs [16].

- Standard commitment scheme: the nonambiguity requirement is absolute (makes no reference to the computational power of the adversary) whereas the secrecy requirement is computational (refers only to probabilistic polynomial-time adversaries).

- Perfect commitment schemes: the secrecy requirement is absolute, while the nonambiguity requirement is computational.

Goldreich and Kahan presented how to construct constant-round zero knowledge proofs for every set in NP, assuming the strong intractability assumptions (the existence of claw-free collections). The claw-free collection implies the existence of one-way functions, but the converse might not be true. Generally, four steps are involved in constructing a round-efficient ZKP [16]:

1. The verifier commits to a challenge. This is usually implemented by two rounds/messages.
2. The prover commits to a sequence of values.
3. The verifier decommits either properly or not.
4. Depending on the verifier's proper decommitment, the prover decommits to the corresponding values.

Based on this construction of constant-round ZKP for NP, Goldreich proved that security will be preserved in the two extreme schedulings of concurrent executions [15]: one is the case of parallel execution, and the other is of concurrent execution under the timing model. The blackbox simulator technique is used for zero-knowledge property verification.

5.3. Knowledge Complexity

Knowledge complexity is introduced by Goldwasser, Micali and Rackoff [24] to measure the computational advantage obtained by interaction. Anything obtained during the interaction will be considered as knowledge. How to quantify the amount of knowledge obtained by interaction is more challenging. One definitional approach is trying to bound the amount of knowledge by the number of bits that are communicated in an alternative interaction that allows to simulate the original interaction. To be more concrete, one party is said to yield at most $k(|x|)$ bits of knowledge if the variation distance between interactive proof system denoted as $(P, V)(x)$ and a simulator denoted as is bounded above by $1 - 2^{-k(|x|)} + |x|^{-c}$. x is the input string. $k(|x|) = \frac{log(|x|)}{O(1)}$ [21].

5.4. Non-interactive Zero-Knowledge

The non-interactive zero-knowledge proof system was defined by Blum, Feldman and Micali. It consists of three entities: a prover, a verifier and a uniformly selected reference string. The reference string is not selected by either parties, but is selected by a trusted third party [16]. The reference string is public to the prover as well as the verifier. The interaction between the two parties is only a single message sent from the prover to the verifier. Then it is left for the verifier to reach the final decision.

Non-interactive zero-knowledge proof systems have many applications such as the construction of public-key encryption and signature schemes. What's more, ZKP can be derived by combining a secure coin-flipping protocol with a non-interactive zero-knowledge proof system [16]. Thus, the round-complexity of this ZKP depends on the round-complexity of the coin-flipping protocol and on whether it can be securely performed in parallel many times.

6. Conclusion

This chapter presented a survey on ZKPs with backgrounds, important concepts, existing applications for NP problems, composition operations and efficiency considerations. It constructed ZKPs for two NP problems: exact cover and 0-1 simple knapsack based on Blum's protocol for Hamiltonicity. Applying the PRS protocol, a concurrent ZKP for 0-1 simple knapsack is provided and explained in detail.

The notion of zero-knowledge originated with the efforts to formalize problems during the design of cryptographic protocols. ZKPs have been shown in many applications of the cryptographic area such as identity verification, authentication, key exchange and enforcing the honest behavior while maintaining privacy. In an authentication system, one party wants to prove its identity to a second party via some secret information but doesn't want the second party to learn anything about this secret. One of the classic authentications based on ZKPs is the Feige-Fiat-Shamir proof of identity [11].

The discovery of ZKPs for all of NP has played an important role in secure computation where several parties engage in a protocol to jointly compute a function on their private inputs in a way that no party learns anything other than the output of the protocol. Since ZKPs can be viewed as a special case of secure two-party computation, every party can use ZKPs to convince each other that it is following the specific protocol without revealing its private input. This application was first developed by Goldreich, Micali and Vigderson in [18]. A general construction of a ZKP for an NP relation, which makes a black-box use of a secure protocol for a related multi-party functionality, was presented in [27].

ZKP can also be a good solution to some interesting problems. For example, in [28], the author proposed a new ZKP of identity scheme based on visual cryptography, visual zero knowledge proof of identity, that has been built with only Boolean OR operations. In [25], the author introduced cryptographic and physical ZKPs systems for solutions of Sudoku Puzzles, which are also known as NP problem.

Existing ZKPs are iterative in nature; their protocols require multiple communication rounds. Most of ZKPs are based on some complexity assumptions (e.g. the existence of one-way function such as quadratic residuosity, factoring, discrete log, etc.). Researchers still try to find a model in which ZKPs for all of NP could be obtained without any assumptions. In secure computation, it is realized that the complexity assumptions could be removed by working in a model with private communication channels. The further work of ZKPs could go between cryptography and complexity theory. Non-black box ZKP has begun to inspire complexity theorists to reexamine whether known limitations of black-box reduction can be bypassed with various types of non-black box reductions. Another direction is to find common variants of ZKPs [42], such as noninteractive zero knowledge, proofs and arguments of knowledge and witness indistinguishable protocols.

References

[1] Petersen Graph http://en.wikipedia.org/wiki/Petersen_graph accessed on Oct 12nd, 2011.

[2] Zero Knowledge Proof http://en.wikipedia.org/wiki/Zero_knowledge_proof accessed on Oct 12nd, 2011.

[3] S. Almuhammadi and C. Neuman. Security and Privacy Using One-Round Zero Knowledge Proofs. *Proceedings of the seventh IEEE international Conference on E-Commerce Technology*, 2005.

[4] L. Babai. Trading Group Theory for Randomness. *In 17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[5] B. Barak. How to go Beyond the Black-Box Simulation Barrier. *In 42nd IEEE Symposium on Foundations of Computer Science*, pages 106–115, 2001.

[6] G. Barssard, D. Chaum, and C. Crepeau. Minumum Disclosure Proofs of Knowledge. *Computer and System Sciences*, 37(2):156–189, 1988.

[7] M. Blum. How to Prove a Theorem So No One Else can Claim It. *Proceedings of the International Congress of Mathematicians*, 1(2):1444–1451, 1987.

[8] G. Brassard and C. Crepeau. Zero-knowledge Simulation of Boolean Circuits. *In Cypto86, Springer-Verlag Lecture Notes in Computer Science*, 263:223–233, 1987.

[9] G. Brassard, C. Crepeau, and M. Yung. Everything in NP can be Argued in Perfect Zero-knowledge in a Bounded Number of Rounds. *Lecture Notes in Computer Science Advances in Cryptology EUROCRYPT 89*, pages 192–195, 1990.

[10] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. *30th ACM symposium on the Theory of Computing*, pages 409–418, 1998.

[11] U. Feige, A. Fiat, and A. Shamir. Zero Knowledge Proof of Identity. *19th STOC*, pages 210–217, 1986.

[12] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. *In 22nd ACM Symposium on the Theory of Computing*, pages 416–426, 1990.

[13] U. Feige and A. Shamir. Zero Knowledge Proof of Knowledge in Two Rounds. *Lecture Notes of Computer Science Advances in CryptologyCRYPTO 89*, 435:526–544, 1990.

[14] U. Feigh, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *Journal on Computing*, 29(1):1–28, 1999.

[15] O. Goldreich. Concurrent Zero Knowledge with Timing, Revisited. *34th ACM Symposium on the Theory of Computing*, pages 332–340, 2002.

[16] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[17] O. Goldreich and H. Krawczyk. On the Composition of Zero Knowledge Proof Systems. *SIAM Journal on Computing*, 25(1):169–192, 1990.

[18] O. Goldreich, S. Micali, and A. Sahai. How to Play Any Mental Game (extended abstract). *Proc. of 19th STOC*, pages 218–229, 1987.

[19] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity or All Languages in NP have Zero-Knowledge Proof Systems. *Journal of the ACM*, 38(3):691–729, 1991.

[20] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptoology*, 7(1):1–32, 1994.

[21] O. Goldreich and E. Petrank. Quantifying Knowledge Complexity. *32nd FOCS*, pages 59–68, 1995.

[22] O. Goldriech. Zero-knowledge: Twenty Years after Its Invention. *Technical Report*, 2004.

[23] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[24] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof System. *SIAM Journal on Computing*, 18:186–208, 1989.

[25] R. Gradwohl, M. N. B. Pinkas, and G. N. Rothblum. Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions for Sudoku Puzzles. 2007.

[26] R. Impagliazzo and M. Yung. Direct Minimum-knowledge Computations. *Proc. Of STOC*, pages 40–51, 1987.

[27] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-Knowledge from Secure Multiparty Computation. *STOC 07*, pages 21–30, 2007.

[28] A. M. Jaafar and A. Samsudin. Visual Zero-knowledge Proof of Identity Scheme: A New Approach. *Second International Conference on Computer Research and Development*, 2010.

[29] G. Jain. Zero Knowledge Proofs: A Survey. *Technical Report University of Pennsylvania*, 2008.

[30] W. Jiang and C. Clifton. A Secure Distributed Framework for Achieving k-anonymity. *Special Issue of the VLDB Journal on Privacy-Preserving Data Management*, pages 180–187, 2006.

[31] A. Juels and M. Sudank. A Fuzzy Vault Scheme. *International Symposium of Information Theory*, pages 408–425, 2002.

[32] R. M. Karp. Reducibility among Combinatorial Problems. *In Complexity of Computer Computations, Proc. Sympos*, pages 85–103, 1972.

[33] J. McLean. Security Models and Information Flow. *In Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, 1980.

[34] R. C. Merkle, S. Member, Ieee, M. E. Hellman, and S. Member. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions On Information Theory*, 24:525–530, 1978.

[35] A. Mohr. A Survey of Zero Knowledge Proofs with Applications to Cryptography. *Southern University at Carbondale*, 2007.

[36] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. *In 43rd FOCS*, pages 366–375, 2002.

[37] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. *In EuroCrypt99, Springer Lecture Notes in Computer Science*, 1592:415–431, 1999.

[38] A. Rosen and O. Goldreich. *Concurrent zero-knowledge*. Springer, 2006.

[39] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.

[40] G. I. Simari. A Primer on Zero Knowledge Protocols. *http://cs.uns.edu.ar/ gis/publications/zkp-simari2002.pdf accessed on Oct. 12nd, 2011.*

[41] C. Tang and Z. Hao. Perfect Zero-Knowledge Argument of Knowledge with Negligible Error Probability in Two-Round for NP from Any One-way Permutation. *2010 International Conference on Communications and Mobile Computing*, 2010.

[42] S. Vadhan. The Complexity of Zero Knowledge. *Proceeding of FSTTCS'07*, 22:612–613, 2007.

[43] A. C. Yao. Theory and application of trapdoor functions. *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 80–91, 1982.

[44] A. C. Yao. How to generate and exchange secrets. *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167, 1986.