Zero-Knowledge

Oleg Koshik

November 18, 2005

Introduction

When concerning any reasonably complex protocol, it is clear, that correct behaivour of different parties plays an important role. Just imagine what could happen if participants misbehave in electronic payment protocol. So becomes obvious the need to enforce correct behaivour. But how? One idea is to make participants prove that they behave correctly with verifying the proof after each message of the protocol.

Considering proofs, it is natural to ask "What does one learn from a proof?" Very often in the protocols it is essential also to preserve privacy, i.e. the proof must not reveal any extra knowledge on the secrets of a participant to another one. By definition, upon verifying a proof, one should be convinced that the assertion being proven is true. But a proof can actually reveal much more than that. Indeed, proofs in mathematics are valued for providing a great deal of insight in addition to validating a particular theorem. And, at a minimum, it seems inherent in the notion of a proof that after verifying a proof, one leaves not just with confidence that the assertion is true, but also with the ability to present the same proof to others and convince them of the assertion.

Zero-knowledge proofs, introduced by Goldwasser, Micali, and Rackoff in 1985, are fascinating constructs which somehow escape the confines of this intuition – they are proofs which are convincing but reveal nothing other than the validity of the assertion being proven. In particular, after verifying a zero-knowledge proof, one does not gain the ability to convince someone else of the same statement!

The applicability of zero-knowledge proofs in the domain of cryptography is vast; as already mentioned, they are typically used to force malicious parties to behave according to a predetermined protocol. Also zero-knowledge proofs serve as a good bench-mark for the study of various problems regarding cryptographic protocols (e.g. the preservation of security under various forms of protocol composition). In addition to their applicability in Cryptography, zero-knowledge has served as a source of inspiration for complexity theory.

Preliminary Background

General problem statement is the following: let L be some language (some set of words) and x some (encrypted value). The task is to prove that $x \in L$. For example, x is prime, x is quadratic residue, x is a private key, corresponding to your public key g.

We say that language L is in **P** iff there is a Turing machine (algorithm) that for each x runs in time at most polynomial in $|x|$ and accepts if and only if $x \in L$.

Randomized algorithms are obtained by allowing our algorithms the ability to flip an unbiased "coin" upon request. We say that algorithm runs in probabilistic polynomial time, if it can flip coins runs in time polynomial in the length of the input. **BPP** is a class of languages L for which there exists a probabilistic polynomial time algorithm, that for each x correctly decides whether $x \in L$ with probability at least $2/3$ over the coin tosses of the algorithm.

Polynomial and probabilistic polynomial time algorithms are usually considered as "efficient algorithms" and the classes of languages **P** and **BPP** are called "easily computable".

The complexity class **NP** consists of those languages possessing efficiently verifiable "classical" proofs. That is, a language L is in **NP** iff there is an efficient proof-verification algorithm (called a verifier) satisfying the following two conditions:

- *Completeness*: For every valid assertion (i.e., every string in L), there exists a proof (that we call **NP**-witness) that the verifier will accept.
- *Soundness*: For every invalid assertion (i.e., every string not in L), no "proof" can make the verifier accept.

We consider such proofs "classical" because the proof is a fixed, written string given in its entirety to the verification algorithm which checks it deterministically.

We know that $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{P} \subseteq \mathbf{BPP}$. We do not know if these containments are strict although it is often conjectured to be the case.

Interactive Proofs

Interactive proofs, introduced by Goldwasser, Micali, and Rackoff serve the same purpose as classical proofs – to convince a verifier with limited computational power that some assertion is true. However, as mentioned above, this is no longer accomplished by giving the verifier a fixed, written proof, but rather by having the verifier to interact with a prover that has unbounded computational power. After the parties exchange messages for some number of rounds, the verifier decides whether to accept or reject. We still require that the verifier's computation time be polynomial in the length of the assertion, but now both the prover and verifier may be randomized. The following two relaxations of the classical notions of completeness and soundness guarantee that an interactive proof is "convincing":

- *Completeness*: For every valid assertion, there is a prover strategy that will make the verifier accept with high probability.
- *Soundness*: For every invalid assertion, the verifier will reject with high probability, no matter what strategy the prover follows.

The complexity class **IP** is the class of languages possessing interactive proofs. Clearly, every language that possesses a classical proof also possesses an interactive proof (in which the prover simply sends the verifier the classical proof). But the converse is not clear; interactive proofs are potentially much more expressive than classical ones. In fact, it has been shown that many more languages possess interactive proofs than classical ones. That is, **IP** is much larger than **NP** (given widely believed complexity-theoretic assumptions).

Note that in general case there is no restriction to prover's computational power. One can replace the general soundness condition by the weaker computational soundness condition, that only requires that it is infeasible (computationally hard) to convince the verifier of the false statement. Protocols that satisfy the computational soundness condition are called arguments. We mention that argument systems may be more efficient than interactive proofs.

Now consider one simple example. We name *Graph Isomorphism* (or GI) a language, that consists of pairs of isomorphic graphs. *Graph Nonisomorphism* (or GNI) is the complement of GI.

It is easy to see that Graph Isomorphism is in **NP**; an easily verifiable proof that two graphs are isomorphic is an isomorphism between them. In contrast, no classical proofs are known for Graph Nonisomorphism. Nevertheless, Graph Nonisomorphism does possess a very efficient interactive proof. The interactive proof is based on two observations. First, if two graphs are nonisomorphic, then their sets of isomorphic copies are disjoint. Second, if two graphs are isomorphic, then a uniformly selected isomorphic copy of one graph is indistinguishable from a uniformly selected isomorphic copy of the other. Thus, the interactive proof, given in Protocol 1, tests whether the (omnipotent) prover can distinguish uniformly selected isomorphic copies of the two graphs.

Protocol 1: Interactive proof $(P; V)$ for Graph Nonisomorphism

Input: Graphs $G_0 = (V_0; E_0)$ and $G_1 = (V_1; E_1)$

1. V : Uniformly select $b \in \{0; 1\}$. Uniformly select a permutation π on V_b. Let $H = \pi(G_b)$. Send H to P.

2. P : If $G_0 \cong H$, let $c = 0$. Else let $c = 1$. Send c to V.

3. V : If $c = b$, accept. Otherwise, reject.

Let's convince in correctness of this protocol. If G_0 and G_1 are nonisomorphic, then $G_0 \cong H$ if and only if $b = 0$. So the prover strategy specified above will make the verifier accept with probability 1. Thus, completeness is satisfied.

On the other hand, if G_0 and G_1 are isomorphic, then H has the same distribution when $b = 0$ as it does when $b = 1$. Thus, b is independent of H and the prover has at most probability at most $\frac{1}{2}$ of guessing b correctly no matter what strategy it follows. This shows that the protocol is sound. Note that we can reduce the cheating probability to $(\frac{1}{2})^k$ by repeating this protocol k times and requiring that the prover succeeds in all k repetitions.

Zero-Knowledge Proofs

A *zero-knowledge proof* is an interactive proof in which the verifier learns nothing from the interaction with the prover, other than the fact that the assertion

being proven is true. This is guaranteed by requiring that whatever the verifier sees in the interaction with the prover is something it could have efficiently generated on its own. That is, there should be a polynomial-time algorithm, called a simulator, that "simulates" the verifier's view of the interaction with the prover (i.e., concatenation of all the messages exchanged between the two parties, prefixed with all random coin tosses of verifier). Recall that the interaction between the prover and verifier is probabilistic. Thus, the simulator is also probabilistic, and we require that it generates an output distribution that is "close" to what the verifier sees when interacting with the prover (when the assertion being proven is true). Intuitively, this means that the verifier learns nothing because it can run the simulator instead of interacting with the prover. Three different interpretations of "close" were suggested and these lead to the three forms of zero-knowledge commonly considered in the literature:

- *Perfect zero-knowledge*: Requires that the distributions are identical.[1]
- *Statistical zero-knowledge*: Requires that the distributions are statistically close, i.e. statistical distance between two distributions is negligible. Even omnipotent verifier cannot distinguish them.
- *Computational zero-knowledge*: Requires that the distributions cannot be distinguished by any probabilistic polynomial-time algorithm.

PZK, **SZK**, and **CZK** are the classes of languages possessing perfect, statistical, and computational zero-knowledge proofs, respectively. Perfect and statistical zero-knowledge capture much stronger requirements than computational zero-knowledge, in that the zero-knowledge condition is meaningful regardless of the computational power of the verifier.

Different classes of languages are related as follows:

$$\mathbf{BPP} \subseteq_{\text{Believed that } \neq} \mathbf{PZK} \subseteq \mathbf{SZK} \subseteq_{\text{Believed that } \neq} \mathbf{CZK} = \mathbf{IP}$$

It is easy to see that $\mathbf{BPP} \subseteq \mathbf{PZK}$, because verifier can verify himself whether $x \in L$ and thus no interaction at all is needed.

Honest Verifier Zero-Knowledge

We call a party *honest* (nonmalicious) when he follows the protocol (though tries to deduce new information from it). However, in general case we may not expect the parties to follow the entire protocol correctly and thus general zero-knowledge protocol must consider also (feasible) adversary strategies, that means there must exist corresponding simulator for malicious verifier strategies. We call the interactive proof $(P; V)$ a *honest-verifier zero-knowledge* if it is zero-knowledge with respect only to honest verifier.

Consider once again Protocol 1. Note that it is not zero-knowledge in its general notion. There is nothing to force a cheating verifier to select H by first picking

[1] Actually the simulator is allowed to fail (outputting a special symbol) with probability $\leq 1/2$, and the output distribution is conditioned on its not failing.

one of the two input graphs and then permuting its vertices. So we have no reason to believe that a cheating verifier "already knows" whether H is isomorphic to G_0 or G_1, thus verifier may gain new knowledge.

Now we will prove, that if the verifier follows the specified protocol, then he can learn nothing, i.e. this proof system is honest verifier zero-knowledge. The only message sent from the prover to the verifier is the guess c. We have already shown that, when the graphs are nonisomorphic, the prover guesses correctly with probability 1. That means that, with probability 1, c is simply equal to b, which is a value the verifier already knows (since it chooses b itself)!

To formalize this intuitive proof, we must exhibit a simulator, as required by the definition of zero-knowledge. The simulator must be an efficient probabilistic algorithm whose output is similar to the verifier's view of the interaction, when given a pair of nonisomorphic graphs as input. The verifier's view of the interaction includes not just the messages exchanged between the verifier and prover (H and c), but also includes the verifier's random coin tosses (the permutation π and the bit b). The simulator, given in Algorithm 2 simply mimics the verifier's protocol and assumes that the prover guesses correctly.

Algorithm 2: Simulator for Graph Nonisomorphism Proof System

Input: Graphs $G_0 = (V_0; E_0)$ and $G_1 = (V_1; E_1)$

1. Uniformly select $b \in \{0; 1\}$. Uniformly select a permutation π on V_b. Let $H = \pi(G_b)$.

2. Let $c = b$.

3. Output $(b; H; c; \pi)$

From the fact that the prover guesses correctly with probability 1 in the protocol, it follows immediately that the output distribution of the simulator is identical to the verifier's view of the interaction (when the input graphs are nonisomorphic and the verifier follows the protocol). Thus, we have that Protocol 1 is an honest-verifier perfect zero-knowledge proof system for Graph Nonisomorphism.

As mentioned before, the probability that the prover can convince the verifier to accept when the graphs are isomorphic can be reduced by repeating the proof system many times. Luckily, both forms repetition (parallel and sequential) preserve honest-verifier perfect zero-knowledge; a simulator for the repeated proof system can be obtained by running the original simulator many times. With cheating verifiers, however, things are more subtle. Sequential repetition preserves zero-knowledge against cheating verifiers, but parallel repetition does not (in general case).

Although it could seem that respect only to honest verifier is quite a strong restriction, there exist efficient transformation for turning certain classes of honest verifier zero-knowledge protocols into zero-knowledge ones. One such class is **SZK**, another one contains *public-coin proofs*, i.e. such proofs that verifier tosses its coins publicly (note that in general case verifier doesn't have to tell prover his results of coin tosses).

Honest verifier zero-knowledge is for many applications sufficient. These protocols are also useful in terms of communication efficiency: for example, every problem possessing an (honest verifier) statistical zero-knowledge proof also has one, in which only two messages are exchanged and the error parameters are

exponentially small. General zero-knowledge protocols require more than three rounds unless the underlying language is trivial (in **BPP**), although it is known that for every language in **IP** there exists constant-round zero-knowledge protocol.

NP is in CZK

One important **NP** problem is Graph 3-Colorability, that means wether there exists an coloring of the vertices of Graph G with 3 colors so that for no edge, the vertices connected to this edge are colored with the same color.

It turns out that the language 3COL containing all 3-colorable graphs is **NP**-complete. That means every NP-language can be reduced to 3COL (using some standard reductions) i.e. in order to prove that *every* **NP**-language has computational zero-knowledge proof it would be sufficient to prove that 3COL has computational zero-knowledge proof.

The following protocol represents the corresponding proof:

Protocol 3: Computational zero-knowledge proof $(P; V)$ for Graph 3-Colorability

Common Input: A graph G(V; E). Suppose that $V \equiv \{1, \ldots, n\}$ for $n := |V|$.

Auxiliary Input (to the prover): A 3-coloring ϕ: $V \to \{1, 2, 3\}$.

The following 4 steps are repeated $t \cdot |E|$ many times so to obtain soundness error exp(-t).

1. P : Select uniformly a permutation π over $\{1, 2, 3\}$. For $i = 1$ to n, send the verifier an encrypted (using a probabilistic public-key cryptosystem) value $\pi(\phi(i))$. For each vertex use different public key.

2. V : Select uniformly an edge $e \in E$ and send it to the prover.

3. P : Upon receiving $e = (i, j) \in E$, send to V the decryption keys to the i-th and j-th values sent in Step 1.

4. V : Check whether or not the decrypted values are different elements of $\{1, 2, 3\}$ and whether or not they match the encryptions received in Step 1.

Analyzing the Protocol 3. Let us consider a single execution of the main loop. Clearly, the prescribed prover is implemented in probabilistic polynomial-time, and always con- vinces the verifier (provided that it is given a valid 3-coloring of the common input graph). In case the graph is not 3-colorable then, no matter how the prover behaves, the verifier will reject with probability at least $1/|E|$ (because at least one of the edges must be improperly colored by the prover). We stress that the verifier selects uniformly which edge to inspect after the prover has committed to the colors of all vertices. Thus, Protocol 3 depicts an interactive proof system for Graph 3-Colorability. As can be expected, the zero-knowledge property is the hardest to establish and we won't bring it in this short overview.

So, we can formulate the following theorem:

*Every language L in **NP** has a computational zero-knowledge interactive proof. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given as auxiliary-input an **NP**-witness for membership of the common input in L.*

Equality **CZK** = **IP** implies that containment **NP** $\in CZK$ is strict.

It is not known, whether analogous containment for **SZK** holds (that **NP** $\in$ **SZK**) but due to strong security guarantee of statistical zero-knowledge this is believed not to be the case. However, **SZK** contains many important computational problems, such like Quadratic Residuosity and Nonresiduosity, Graph Isomorphism and Nonisomorphism, a problem equivalent to Discrete Logarithm. Statistical zero-knowledge proofs can and have been used in specific cryptographic protocols.

Cryptographic Applications

In a typical cryptographic setting, a user referred to as U, has a secret and is supposed to take some action depending on its secret. The question is how can other users verify that U indeed took the correct action (as determined by U's secret and the publicly known information). Indeed, if U discloses its secret then anybody can verify that U took the correct action. However, U does not want to reveal its secret. Using zero-knowledge proofs we can satisfy both conflicting requirements (i.e., having other users verify that U took the correct action without violating U's interest in not revealing its secrets). That is, U can prove in zero-knowledge that it took the correct action. Note that U's claim to having taken the correct action is an **NP**-assertion (since U's legal action is determined as a polynomial-time function of its secret and the public information), and that U has an **NP**-witness to its validity (i.e., the secret is an **NP**-witness to the claim that the action fits the public information). Thus it is possible for U to efficiently prove the correctness of its action without yielding anything about its secret. Consequently, it is fair to ask U to prove (in zero-knowledge) that it behaves properly, and so to force U to behave properly. Indeed, "forcing proper behavior" is the canonical application of zero-knowledge proofs.

Another possible utility of zero-knowledge is the construction of identification schemes, introduced by Feige, Fiat, and Shamir. The premise is that one party, Alice, should be able to identify herself repeatedly to a second party, Bob. For example, Bob can be thought of as an internet service provider or a remote computer network on which Alice has an account. The most common solution for this problem is for Alice to choose a password that Bob keeps stored in a secure password file. When Alice wishes to identify herself to Bob, she simply sends her password to Bob, who checks it against the file. The difficulty with this solution is that an adversary can, by impersonating Bob, obtain Alice's password and later use this to misrepresent himself as Alice.

Zero-knowledge proofs provide an elegant solution to this problem. Instead of choosing a password, Alice generates a true mathematical statement S for which only she knows the proof (and such that it is difficult for an adversary to come up with a proof for S). Bob stores this statement. When Alice wishes to identify herself to Bob, she gives Bob a zero-knowledge proof that S is true. This identifies Alice as the one who knows a proof for S, while Bob (or an adversary impersonating Bob) does not learn the proof for S and hence cannot later misrepresent himself as Alice. Some practical identification protocols have been developed using this idea.