Senior Research Thesis Extended Abstract
Direct Zero-Knowledge Proofs

Jeremiah Blocki
Advisor: Manuel Blum

April 9, 2009

Abstract

Definition A Zero-Knowledge Proof is an interactive protocol which allows one party (the prover) to prove a statement (S) to another party (the verifier) without revealing anything beyond the truth of S. If S is true then the prover should always be able to convince the verifier that the statement is true without revealing anything else. For example, Goldreich, Micali, and Wigderson [1991] found a protocol which allowed the prover to convince a verifier that a graph G is k-colorable without revealing anything else. Consequently, all languages in NP are known to have Zero-Knowledge Proofs because they can be reduced to Graph Coloring. However, there are no known direct Zero-Knowledge Proofs for many NP-Complete languages. I present direct Zero-Knowledge Proof protocols for Subset Sum, Clique, SAT and Integer Programming.

1 Introduction

1.1 Problem Description

A Zero-Knowledge Proof must satisfy three properties: completeness, soundness and zero-knowledge. Completeness means that when S is true then the prover can always convince the verifier of this by following the protocol. Soundness means that when S is false then the protocol guarantees that the verifier will catch the prover cheating with high probability. Finally, the Zero-Knowledge property means that the verifier does not learn anything more than “S is true.” In particular the verifier will not know why S is true and will not be able to prove to anyone that S is true. Typically, a protocol
is shown to satisfy the zero-knowledge property by producing a simulator which can be run without the prover. The verifier can run the simulator independently to produce ‘interactions’ which cannot be distinguished from the actual interactions produced by protocol.

1.2 Problem History

Goldreich et al. [1991] gave the first Zero Knowledge Proof scheme for an NP-Complete language, Graph Coloring. Therefore, every language in NP has a Zero-Knowledge Proof scheme by reduction to Graph Coloring. This is nice in theory, but in practice it may be extremely difficult to reduce a problem to Graph Coloring.

SAT was the first language shown to be NP-Complete language. Other NP-Complete have been established by reductions either from SAT or another NP-Complete language. Cook showed that SAT is NP-Complete by considering the Tableau of a Turing Machine. Therefore, the first step in the known reduction from Subset-Sum to Graph Coloring involves building a Turing Machine to verify Subset-Sum solutions. In theory this is fine, but in practice no one would ever build a Turing Machine to verify Subset-Sum solutions!

1.3 Results

Zero-Knowledge Proofs can be used in authentication protocols and to achieve secure multiparty computation. In practice, it would be nice to have direct Zero-Knowledge Proof schemes. I will present Zero-Knowledge Proof protocols for Subset Sum, Clique, SAT and Integer Programming.

2 Subset Sum

2.1 Problem Definition

**Definition** A *Subset Sum* instance \( I \leftarrow S, m \) consists of a set \( S \) of integers and an integer \( m \). \( I \) is a yes instance if and only if

\[ \exists X \subseteq S, \sum_{x \in X} x = m \]

The *Subset Sum* problem is well known to be NP-Complete [Karp, 1972]. For simplicity, I present a protocol for the Equal Partition Problem, a special
case of Subset Sum which is also known to be NP-Complete. Given a set $S = \{v_1, ..., v_n\}$ find a disjoint partition $S = S_1 \cup S_2$ such that:

1. $||S_1|| = ||S_2||$
2. $S_1 \cap S_2 = \emptyset$
3. $\sum_{x \in S_1} x = \sum_{x \in S_2} x$

### 2.2 Protocol

**Prover:** Assume that the prover knows such set $X$ with

$$||X|| = \frac{n}{2}$$

Define $M = \sum_{x \in X} x$

1. Generate $r_1, ..., r_n$, $r_i$ is chosen uniformly at random from $[0, ..., M]$ at random under the constraint

   $$\sum_{i: v_i \in X} r_i = 0 \mod M + 1$$

2. Compute $R_i = r_i + v_i \mod M + 1$

3. Hide a permuted version of the following table:

<table>
<thead>
<tr>
<th>$v$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$r_1$</td>
<td>$r_2$</td>
<td>...</td>
</tr>
<tr>
<td>$R$</td>
<td>$R_1$</td>
<td>$R_2$</td>
<td>...</td>
</tr>
<tr>
<td>$v_i \in X? $</td>
<td>$b_1$</td>
<td>$b_2$</td>
<td>...</td>
</tr>
</tbody>
</table>

Where $b_i = 1$ indicates that $v_i \in X$ and $b_i = 0$ indicates $v_i \notin X$.

**Verifier:**

Now the verifier can ask to see exactly one of the following:

1. $\frac{n}{2} - 1$, of the triples $(v_1, r_1, R_1)$ (expecting $v_1 + r_1 = R_1 \mod M + 1$)
2. $R_1, ..., R_n, b_1, ..., b_n$ (expecting $\sum_{i=1}^{n} b_i R_i = k \mod M + 1$)
3. $r_1, ..., r_n, b_1, ..., b_n$ (expecting $\sum_{i=1}^{n} b_i r_i = 0 \mod M + 1$)

**Claim 2.1** The above protocol is a Zero-Knowledge Proof scheme
Proof We must show that the protocol satisfies the Zero-Knowledge, Completeness and Soundness conditions.

Zero-Knowledge The key intuition is that $r_i$ by itself is just a random number. Similarly, $R_i$ is just a random number without $r_i$. Thus, at each step, the verifier is shown numbers which he could have generated randomly. Formally, the verifier could easily simulate choice 1 by himself by picking the $(\frac{n}{2} - 1)$, $v_i$ values for the triples he wants to see, generating $(\frac{n}{2} - 1)$ of the corresponding $r_i$ values at random and then setting the $R_i$ variables accordingly. Similarly, the verifier could simulate choice 3 by picking picking $b_1, ..., b_n$ such that

$$||X'| = \frac{n}{2}$$

where $X' = \{i : b_i = 1\}$, and then picking $r_1, ..., r_n$ the same way the prover does...uniformly at random from $[0, ..., M]$ under the constraint

$$\Sigma_i: v_i \in X \sum r_i = 0 \mod M + 1$$

Finally, the verifier can simulate choice 2 by himself by picking the $R_1, ..., R_n$ and $b_1, ..., b_n$ values uniformly at random from $[0, ..., M]$ under the constraint(s)

$$\Sigma^n_{i=1} b_i R_i = k$$

and

$$\sum |i : b_i = 1| = \frac{n}{2}$$

Completeness If the prover knows of a set $X$ then it is easy to verify there is no way for him to be caught if he follows the protocol.

Soundness Suppose that one of the following was true

1. $\exists i \text{ s.t. } v_1 + r_1 \not\equiv R_1 \mod M + 1$

2. $\Sigma^n_{i=1} b_i R_i \not\equiv k \mod M + 1$

3. $\Sigma^n_{i=1} b_i r_i \not\equiv 0 \mod M + 1$

Then a verifier who selects from the three options uniformly at random has at least a $\frac{1}{3}(\frac{n-2}{2n}) > \frac{1}{8}$ chance of catching the prover. Now suppose that all three statements were always false, so that it is impossible for the verifier to ever catch the prover. Then set

$$X = \{v_i, b_i = 1\} \subset S$$
\[ \sum_{x \in X} \equiv \sum_{i=1}^{n} b_i v_i \mod M + 1 \quad (1) \]
\[ \equiv \sum_{i=1}^{n} b_i v_i + \sum_{i=1}^{n} b_i r_i \mod M + 1 \quad (2) \]
\[ \equiv \sum_{i=1}^{n} b_i (v_i + r_i) \mod M + 1 \quad (3) \]
\[ \equiv \sum_{i=1}^{n} b_i R_i \mod M + 1 \quad (4) \]
\[ \equiv k \mod M + 1 \quad (5) \]

2.3 Example

1. \( S = \{59, 32, 23, 60, 85, 90, 60\} \)

2. \( k = 248 \)

3. \( M = \sum_{x \in S} = 453 \)

4. \( X = \{44, 60, 85, 59\} \)

2.3.1 Peggy

Peggy knows \( X \) and generates the following table, but hides all of the cells of the table from Victor.

\[
\begin{array}{c|cccccccc}
 v & 44 & 32 & 60 & 85 & 59 & 23 & 90 & 60 \\
 r & 94 & 314 & 103 & 0 & 257 & 387 & 27 & 433 \\
v_i \in X? & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

2.3.2 Victor

Victor sees the following table

\[
\begin{array}{c|cccccccc}
 v & * & * & * & * & * & * & * & * \\
r & * & * & * & * & * & * & * & * \\
R & * & * & * & * & * & * & * & * \\
v_i \in X? & * & * & * & * & * & * & * & * \\
\end{array}
\]

Victor now has three choices:

1. Victor asks for proof that \( v_i + r_i \equiv R_i \mod M + 1 \) by pointing at \( \frac{n}{2} - 1 = 3 \), of the columns. In response Peggy reveals this part of the table
2. Victor asks for proof that $\sum_{i=1}^{n} b_i R_i \equiv k \mod M + 1$. In response Peggy reveals another part of the table:

\[
\begin{array}{c|cccccccc}
\nu & * & * & * & * & * & * & * & * \\
\nu & 44 & 32 & 60 & * & * & * & * & * \\
\nu & 94 & 314 & 103 & * & * & * & * & * \\
\nu & 138 & 346 & 163 & * & * & * & * & * \\
\alpha_i \in X? & * & * & * & * & * & * & * & * \\
\end{array}
\]

3. Victor asks for proof that $\sum_{i=1}^{n} b_i r_i \equiv 0 \mod M + 1$. In response Peggy reveals part of the table:

\[
\begin{array}{c|cccccccc}
\nu & * & * & * & * & * & * & * & * \\
\nu & * & * & * & * & * & * & * & * \\
\nu & 138 & 346 & 163 & 85 & 316 & 410 & 117 & 39 \\
\nu & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

2.4 Extending the Protocol to regular Subset Sum

The protocol can be adapted to the Subset Sum problem, though there is one major technicality. Our protocol cannot reveal the size ($|X|$) of our Subset Sum solution $X \subset S$. We no longer guarantee that $|X| = \frac{|S|}{2}$. To fix this we create a new set $S'$ by padding the set $S$ with $|S|$ zero entries. Clearly, we cannot generate any new Subset Sums with $S'$ because we just added 0 a bunch of times. However, if there was a solution $X \subset S$ of size $|X| < \frac{|S|}{2}$, we can create $X' \subset S'$ of size $|S|$ by padding $X$ with zeros. Thus we may assume without loss of generality that our Subset Sum solutions in $S'$ have size $\frac{|S'|}{2}$.

3 Clique

3.1 Problem Definition

Definition A Clique instance $I = (G, k)$ is a undirected graph $G$ and an integer $k$. $I$ is a yes instance if and only if $G$ contains a clique of size $\geq k$.

Clique was one of Richard Karp’s original 21 NP-complete problems [Karp, 1972].

Let $M_G$ denote the adjacency matrix of $G$. 

6
3.2 Protocol

Prover:

1. Generate an isomorphic graph \( G' \)

2. Write down (but hide) \( M_{G'} \)

Verifier:

1. Ask to see that \( G' \) is isomorphic to \( G \) (the prover just reveals \( M_{G'} \) and the permutation).

2. Ask to see that \( G' \) contains a \( k \)-clique (the prover reveals which nodes are in the \( k \)-clique and reveals the corresponding cells in the adjacency matrix ... these should all be 1).

Claim 3.1 The above protocol is a Zero-Knowledge Proof scheme

Proof We must show that the protocol satisfies the Zero-Knowledge, Completeness and Soundness conditions.

Zero-Knowledge Victor could simulate option 1 by simply generating an isomorphic graph and writing down \( M_{G'} \). Victor can simulate option 2 by just picking \( k \) nodes at random and writing down some adjacency matrix where all of the edges between these nodes is 1.

Completeness If the prover knows of a \( k \)-clique in the graph then the prover can always write down an isomorphic graph \( G' \) for which he knows a \( k \)-clique and never get caught.

Soundness If the prover does not know of a \( k \)-clique then either \( G' \) is not isomorphic or the prover does not know of a \( k \)-clique in \( G' \) either. In either case the verifier can catch the cheat with probability \( \frac{1}{2} \) by selecting from the two options randomly.

Note: Essentially the same protocol works for many graph problems such as: Hamiltonian Path/Cycle, Subgraph Isomorphism, Independent Set and Vertex Cover
4 SAT

4.1 Definition of Problem

SAT is the original NP-Complete problem Cook [1971]. For simplicity, I present a Zero-Knowledge Proof for the Exactly One in 3 - SAT problem, which is also well known to be NP-Complete Schaefer [1978]. We are given a 3 – SAT formula \( \phi \) with variables \( x_1, ..., x_n \) and clauses \( C_1, ..., C_m \).

\[
C_i = \{ \ell_{i,1}, \ell_{i,2}, \ell_{i,3} \}
\]

4.2 Protocol

Prover:

1. For each variable \( x_i \) (replace \( x_i \) with \( \bar{x}_i \) and replace \( \bar{x}_i \) with \( x_i \)) with probability \( \frac{1}{2} \). Notice that the formula remains equivalent when we do this.

2. Permute the variables and the clauses. Let \( x_i' \) and \( C_j' \) denote the permuted clauses and variables.

3. Write Down (but hide) the new list of clauses

4. Write Down (but hide) the satisfying assignment to the permuted formula

Verifier: The verifier can ask for

1. A Proof that the new formula is equivalent (the prover can just reveal the new list of clauses and the permutations of the variables and clauses used to generate the new formula, the satisfying assignment)

2. Proof that a particular clause is satisfied by the hidden assignment (The prover reveals the assignment for those particular three variables, all the other clauses remain hidden as well as the permutations)

Claim 4.1 The above protocol is a Zero-Knowledge Proof Protocol for 3-SAT

Proof We must show that the protocol satisfies the Zero-Knowledge, Completeness and Soundness conditions.

Zero Knowledge: The verifier could simulate this protocol by himself.
1. If he chooses option 1 then randomly permute the variables and clauses and ‘show’ himself the new formula.

2. If he chooses option 2 then randomly permute the variables and clauses then randomly select a clause, literal and make up an assignment such that the clause is true (in the assignment exactly one of the literals will be true).

**Completeness:** If Peggy knows of a satisfying assignment, she can never be caught by Victor if she just uses the actual assignment.

**Soundness:** If the Prover does not know of a satisfying assignment then she can either

1. Write down some other nonequivalent formula (getting caught by option 1)

2. Write down some assignment which doesn’t satisfy all clauses (getting caught in option 1 with probability at least $\frac{1}{m}$)

With some work the protocol can be extended to regular SAT.

## 5 Integer Programming

### 5.1 Introduction to Problem

Without loss of generality linear program is a set of equations of the form

$$a_1x_1 + \ldots + a_nx_n = d$$

where $d, a_1, \ldots, a_n$ are constants and $x_1, \ldots, x_n$ are variables. 0-1 Integer Programming is also one of Karp’s 21 NP-Complete problems [Karp, 1972]. It adds the constraint $x_i \in \{0, 1\}$. This problem is very similar to the Subset Sum problem (with $S = \{a_1, \ldots, a_n\}$ and $d = m$) except that we may now have multiple constraints. The Zero-Knowledge Proof techniques are also very similar.

### 5.2 Protocol

Given constraints

$$a_1,jx_1 + \ldots + a_n,jx_n = d_j$$
for \( i = 1, \ldots, m \). Set \( M = \max_i \sum_{j=1}^n a_{ij} \).

**Prover:**

1. Create dummy variables \( x_{n+1}, \ldots, x_{2n} \) such that \( x_{n+i} = \bar{x}_i \).
2. Pick \( y_1, \ldots, y_{2n} \) uniformly at random from \([0, \ldots, M]\). For the first row, write down (but hide) \( y_1, \ldots, y_{2n} \).
3. Compute \( z_1, \ldots, z_{2n} \) such that
   \[
   y_i + z_i \equiv x_i \mod M
   \]
   For the second row, write down (but hide) \( z_1, \ldots, z_{2n} \).
4. For the other rows, write down (but hide) \( a_{1,j}, \ldots, a_{n,j}, a_{1,i}, \ldots, a_{n,i} \).
5. (In practice the prover also permutes the columns, for ease of analysis and explanation we pretend that everything is written in the natural order)
6. For each equation, compute and write down (but hide)
   \[
   v_{1,i} = \langle z_1, \ldots, z_n \rangle \cdot \langle a_{1,i}, \ldots, a_{n,i} \rangle \mod M + 1
   \]
7. Compute and write down (but hide)
   \[
   v_{2,i} = \langle y_1, \ldots, y_n \rangle \cdot \langle a_{1,i}, \ldots, a_{n,i} \rangle \mod M + 1
   \]

**Verifier:**

1. Proof that the permutated 0-1 Integer Program is equivalent (the prover just shows the permutation of the columns)
2. Show that \( z_i + y_i \equiv \{0, 1\} \mod M + 1 \) for all \( 0 < i \leq 2n \) (in fact for exactly \( n \) values of \( i \), \( z_i + y_i \equiv 1 \mod M + 1 \))
3. Show that
   \[
   v_{1,i} = \langle z_1, \ldots, z_n \rangle \cdot \langle a_{1,j}, \ldots, a_{n,j} \rangle \mod M + 1
   \]
   was computed correctly
4. Show that
\[ v_{2,i} = < y_1, ..., y_n > \cdot < a_{1,i}, ..., a_{n,i} > \mod M + 1 \]
was computed correctly for each equation.

5. Show that
\[ v_{1,i} + v_{2,i} \equiv d_i \mod M + 1 \]
for each equation.

Note: The technique can be extended to regular Integer Programming where we have constraints of the form \( x_i' \in \{0, ..., 2^m\} \). Simply replace \( x_i' \) with \( x_{i,1}, ..., x_{i,m} \in \{0, 1\} \) to represent each bit of \( x_i' \).

**Claim 5.1** *The above protocol is Zero-Knowledge*

**Proof** We must show that the protocol satisfies the Zero-Knowledge, Completeness and Soundness conditions.

**Zero Knowledge:** The verifier could simulate this protocol by himself. Notice that \( y_i, z_i \) are independently random numbers when viewed separately. So any step where we only see \( y_i \) or \( z_i \) does not reveal anything. Victor could also generate \( v_{1,i}, v_{2,i} \) in the last step by picking all of the \( y_i \) values at random to obtain \( v_{i,1} \) and then setting \( v_{2,1} \) to guarantee that
\[ v_{1,i} + v_{2,i} \equiv d_i \mod M + 1 \]
In fact the step where Victor sees the \( y_i, z_i \) values and nothing else is ok. There should be exactly \( n \) values of \( i \) s.t \( y_i + z_i \equiv 1 \mod M + 1 \) and exactly \( n \) values of \( i \) s.t \( y_i + z_i \equiv 1 \mod M + 1 \). Victor could have simply picked random \( y_i \) values and then picked the \( z_i \) values so that both statements are true.

**Completeness:** Clearly, if Peggy follows the protocol she cannot get caught cheating.

**Soundness:** Suppose Peggy is trying to cheat. To not be caught she must write down an equivalent 0−1 Integer Program and pick \( y_i, z_i \) values such that:

1. For \( n \) values of \( i \)
\[ z_i + y_i \equiv 1 \mod M + 1 \]
2. For the other \( n \) values of \( i \)

\[
z_i + y_i \equiv 0 \quad \text{mod } M + 1
\]

3. For all \( j \),

\[
< z_1, ..., z_n > \cdot < a_{1,j}, ..., a_{n,j} > + < y_1, ..., y_n > \cdot < a_{1,j}, ..., a_{n,j} > \equiv d_j \quad \text{mod } M + 1
\]

But then we can set

\[
x_i \equiv z_i + y_i \equiv 1 \quad \text{mod } M + 1
\]

Noting that \( x_i \in \{0, 1\} \) and for all \( j \)

\[
\Sigma_i x_i a_{i,j} \equiv d_j \quad \text{mod } M + 1
\]

But

\[
\Sigma_i x_i a_{i,j} < M + 1
\]

Therefore, for all \( j \)

\[
\Sigma_i x_i a_{i,j} = d_j
\]

So if Peggy cheats she will get caught with reasonable probability.

References


