Zero Knowledge Compilers

John T. McCall
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
mcca0798@morris.umn.edu

ABSTRACT

<Insert Abstract Here>

General Terms

Need to figure this out yet

Keywords

Zero Knowledge Protocols, Compilers

1. INTRODUCTION

I will focus on Zero-Knowledge Compilers which are compilers that automatically generate Zero-Knowledge proofs. This is how I plan to use the following sources:

- I expect [**?**, **?**, **?**] to be my core sources, depending on how relevent [**?**] turns out to be I'll replace it with a better source.
- I will use [**?**, **?**, **?**] for background information and examples of Zero-Knowledge Protocols.
- I will need to find some papers for background information on compilers.

As stated above I need to find some sources about compilers. I probably will need to find more papers dealing with ZK-Compilers as well.

1.1 Key Points

What main problems(s) or questions(s) does the research address?

The main problem that the research address is how to create reliable zero knowledge protocols. They can be difficult to define and even harder to verify. Zero knowledge compilers help because they can efficiently generate zero knowledge protocols, and because of how they are constructed the user can trust that they will work.

What are the key contributions of each of your main sources?

Source [**?**] provides a great deal of information about their zero knowledge compiler, ZKCrypt. They go into detail about zero knowledge protocols, how their compiler produces them, and they give a proof verifying that their protocols are valid. They also talk about a few applications of their compiler.

Source [**?**] talks in depth about ZKPDL, which is a language they created for writing zero knowledge protocols. They also created an interpreter for this language, which performs optimizations to lower computational and space overhead. This paper also provides an example dealing with electronic cash.

Source [**?**] uses Σ-Protocols in a compiler to automatically generate sound and efficient zero knowledge proofs of knowledge. The compiler automatically generates the implementation of the protocol in Java, or it can output a description of the protocol in LaTeX.

How are the main sources related to each other?

The main sources all use compilers to generate zero knowledge protocols, but the ways they are implemented are all different so there is some room for comparison. All the compilers are also based off Σ-Protocols, or variations of Σ- Protocols.

What is the state of the research?

The current state is that the compilers have been implemented and tested. They all provided enough data to back up their research. Most of the work they are doing now will extend the applications of their compilers to support other proof types.

What background material will you need to present in order for your audience to understand the research?

I will need to provide background information on zero knowledge protocols and compilers. It's probably more important that I focus on zero knowledge protocols and only give basic compiler background.

2. ZERO KNOWLEDGE PROTOCOLS

Zero knowledge protocols, also referred to as zero knowledge proofs, are a type of protocol in which one party, called the *prover*, tries to convince the other party, called the *verifier*, that a given statement is true.

2.1 Definition

Formally, a zero knowledge proof is a type of interactive proof. Mohr gives a concise definition in [**?**].

DEFINITION 1. *An interactive proof system for a set S is a two-party game between a verifier executing a probabilis-*

UMM CSci Senior Seminar Conference, December 2013 Morris, MN.

tic polynomial-time strategy and a prover which executes a computationally unbound strategy satisfying:

- *Completeness: For every $x \in S$, the verifier always accepts after interacting with the prover on common input x*
- *Soundness: For some polynomial p, it holds that for every $x \notin S$ and every potential strategy P^*, the verifier rejects with probability at least $\frac{1}{p(|x|)}$ after interacting with P^* on common input x.*

To summarize: if an honest verifier is always convinced after interacting with an honest prover, then the proof is complete. A proof is sound if a cheating prover can only convince an honest verifier with some small probability.

For an interactive proof to be a zero knowledge proof it must, unsurprisingly, satisfy the condition of *Zero Knowledge.*

DEFINITION 1. *A proof is zero knowledge on (inputs from) the set S if, for every feasible strategy B^* there exists a feasible computation C^* so that the following two probability ensembles are computationally indistinguishable:*

- *the output of B^* after interacting with A on common input $x \in S$*
- *the output of C^* on input $x \in S$*

In other words, any information, in this case B^*, that can be learned by interacting with A can also be learned without interacting with A. [**?**]

In certain cases the ability to prove a statement requires some secret knowledge by the prover. A *zero knowledge proof of knowledge* is a specific type of zero knowledge protocol in which the prover must convince the verifier of the validity of the statement without revealing that secret knowledge. [**?**]

2.2 Examples

Below are a couple of examples of zero knowledge protocols. The first example is easy to follow and just highlights how a zero knowledge protocol functions. The second is more complex, but is a more real world application.

2.2.1 The Magic Cave

The classic example for zero knowledge protocols is the cave example. First presented in [**?**] and then restated in [**?**] the cave example is the go to example for learning zero knowledge protocols.

Peggy has stumbled across a magical cave. Upon entering the cave there are two paths, one leading to the right and one leading to the left. Both paths eventually lead to a dead end, however Peggy has discovered a secret word that opens up a hidden door in the dead end, connecting both paths.

Victor hears about this, and offers to buy the secret from Peggy for $1,000,000, which Peggy agrees to. Before giving Peggy the money Victor wants to be certain that Peggy actually knows this secret word. How can Peggy (the prover) convince Victor (the verifier) that she knows the word, without revealing what it is?

The two of them come up with the following plan. First, Victor will wait outside the cave while Peggy goes in. She will randomly pick either the right or the left path and go down it. Since Victor was outside he should have no knowledge of which path Peggy took. Then Victor will enter the cave. He will wait by the fork and shout to Peggy which path to return from.

Assuming that Peggy knows the word, she should be able to return down the correct path, regardless of which one she started on. If Victor says to return down the path she started on, she simply walks back. If Victor says to return down the other path, she whipers the magic word, goes through the door, and returns down the other path.

If Peggy doesn't know the word, about there is a 50% chance that Victor will choose the path she did not start down. If this happens there is no way that she can return down the correct path. The experiment should be repeated until Victor either discovers Peggy is a liar becuase she returned down the wrong path, or until he is sufficiently satisfied that she does indeed know the word.

Is this a zero knowledge protocol? It satisfies completeness because if Peggy knows the word she will be able to convince Victor. It is sound because if Peggy doens't know the word she will not be able to convince Victor unless she was very lucky. Finally it is zero knowledge because if Victor follows the protocol he will not be able to learn anything besides whether or not Peggy knows the word. Since this protocol satisfies these three conditions it is in fact a zero knowledge protocol.

2.2.2 Graph Isomorphism

3. COMPILERS

This section will provide some basic background information about compilers.

3.1 Background

4. ZERO KNOWLEDGE COMPILERS

This section will be the main section. Here I will talk about my core sources and how they are using their compilers.

4.1 Sigma-Protocols

Here I'll talk about Sigma-protocols and how they are used in the following compilers.

Note to myself: Look into the Fiat-Shamir heuristic

4.2 ZKCrypt

Here I'll talk about ZKCrypt, it's implementation and verification steps.

4.3 ZKPDL

Similar to ZKCrypt section.

5. APPLICATIONS

Here I'll talk about how those compilers are used in the real world.

5.1 Electronic Cash

5.2 Another Application

6. CONCLUSION

This is where I'll neatly wrap everything up.