Zero Knowledge Compilers

John McCall

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

December 7, 2013

Overview

- Zero knowledge protocols are used to prove the validity of a statement, without revealing anything other than the correctness of the claim.
- They have practical applications in cryptography.
- They are difficult to design and to implement.
- Zero knowledge compilers help to ease this burden.

Outline

Outline

Zero Knowledge Proofs

- Consist of a prover and a verifier.
- Can be used to prove knowledge of a statement.
- Are a type of interactive proof.

Interactive Proof

Must satisfy:

- *Completeness:* If the statement being proven is true, an honest verifier, a verifier correctly following the protocol, will be convinced after interacting with an honest prover.
- *Soundness:* If the statement is false, no prover, either honest or dishonest, will be able to convince an honest verifier, except with some small probability.

Zero Knowledge Proof

Must satisfy:

- *Zero knowledge:* Any knowledge known by the prover or the verifier before performing the proof is the same as the knowledge known by either party after performing the proof.

Setup

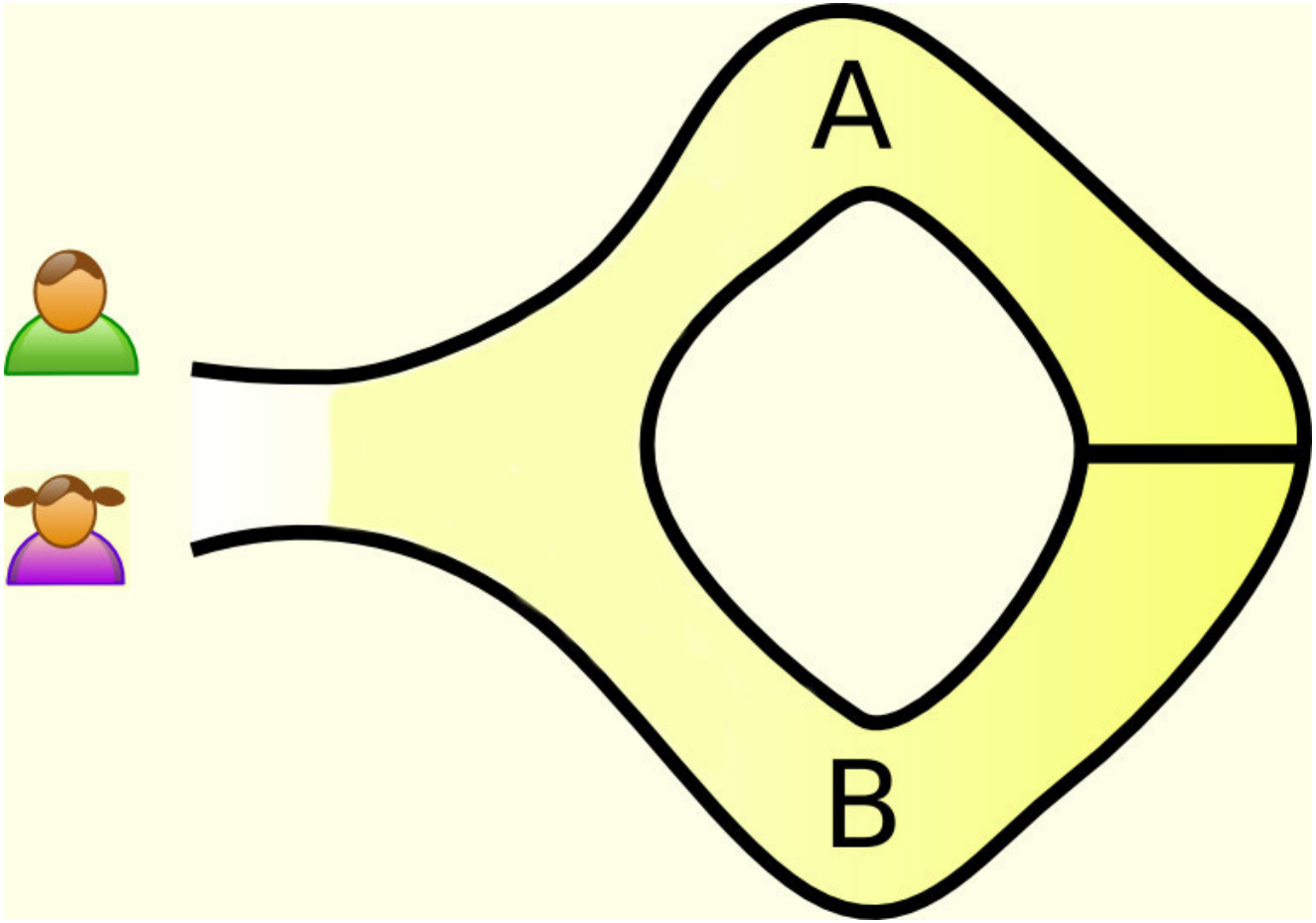

Figure : An image of the cave.
Modified from: `wikipedia.org/wiki/Zero-knowledge_proof`

Protocol

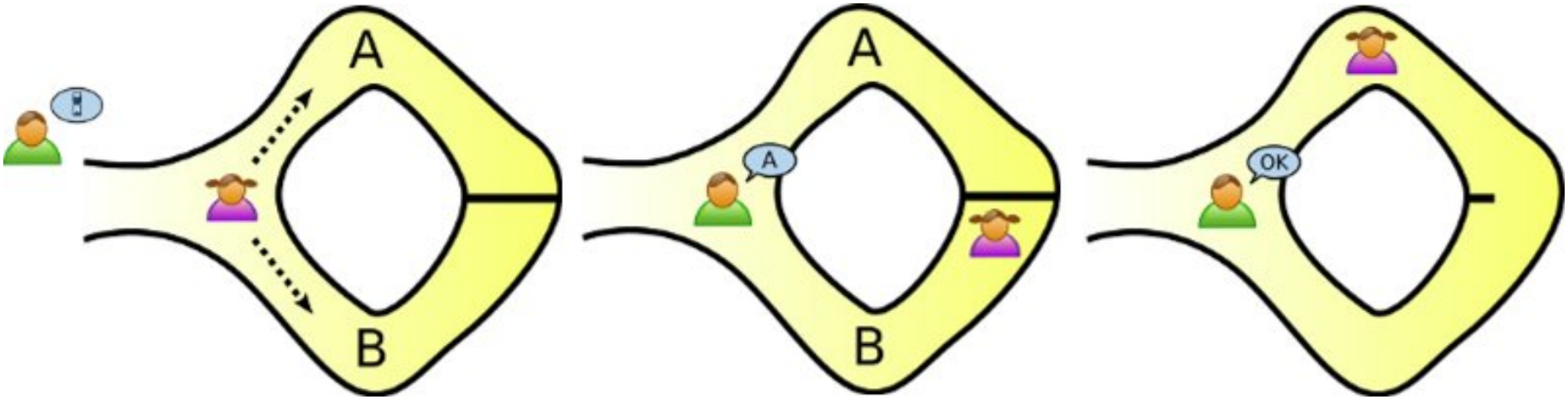

Figure : Illustration of the magic cave protocol.
Taken from: `wikipedia.org/wiki/Zero-knowledge_proof`

Definitions

- A *cycle* is a list of connected vertices of a graph which starts and ends at the same vertex.
- A *Hamiltonian path* is a list of connected vertices of a graph which includes each vertex exactly once.
- A *Hamiltonian cycle* is a Hamiltonian path which is also a cycle.

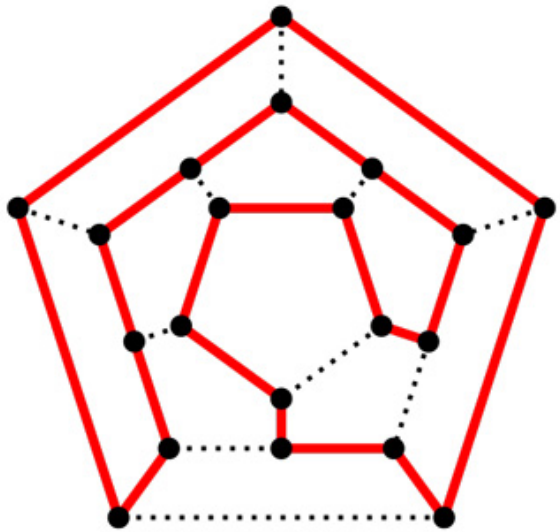

Figure : An example of a Hamiltonian cycle. The solid line marks the path. Taken from: `wikipedia.org/wiki/Hamiltonian_path`

Definitions

- An *isomorphism*, $f : V(G) \to V(H)$, of graphs G and H is a bijection between the vertex sets of G and H such that any two vertices u and v are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

One-way Functions

- Difficult to reverse.
- Low probability of collisions.
- Used when the prover needs to choose a value which remains a secret at first, but will be revealed later.

Setup

- Want to prove knowledge of a Hamiltonian cycle in a graph without revealing the cycle.
- Peggy knows a Hamiltonian Cycle for a graph, G. Victor has knowledge of G but not the cycle.

Protocol

- Peggy constructs H, a graph which is isomorphic to G. Since Peggy knows a Hamiltonian cycle for G she must know one for H as well.
- Peggy commits to H, using a one-way function. Doing this means that Peggy cannot change H without Victor finding out.
- Victor then randomly asks Peggy to do one of two things. Either show the isomorphism between H and G, or show a Hamiltonian cycle in H.

Protocol

- If Peggy was asked to show that the two graphs are isomorphic, she starts by revealing H to Victor. She also reveals the isomorphism between G and H. Victor can then verify that the two graphs are isomorphic.
- If Peggy was asked to show a Hamiltonian cycle in H, she first translates the cycle from G onto H. She then reveals to Victor the Hamiltonian cycle in H.
- In both cases Victor must also verify that H is the same graph that Peggy committed to by using the same one-way function and comparing the outputs.

Outline

Compilers

- Translates one language into another.
- Many different types of compilers: single-pass compilers, multi-pass, optimizing compilers, and many others.
- The first compilers started to appear in the 1950s.
- Notoriously difficult to implement, at the time.
- There are two parts to compilation, analysis and synthesis.

Zero Knowledge Compilers

- Take, as input, an abstract proof specification or proof-goal, written in languages designed specifically for this problem.
- Output an implementation of the given specification in a high-level language, usually C++ or Java.

Outline

Definitions

A *group*, in a mathematical sense, is a set, G paired with an operation, $\odot$. Written as $(G, \odot)$

- The set must be closed under that operation.
  - If $a, b \in G$ then $a \odot b \in G$.
- The operation must be associative.
  - If $a, b, c \in G$ then $(a \odot b) \odot c = a \odot (b \odot c)$.
- There must be an identity element.
  - There exists an $e \in G$ such that for all $a \in G$, $a \odot e = a$.
- There must be an inverse element.
  - For all $a \in G$, there exists an $a^{-1} \in G$ such that, $a \odot a^{-1} = e$.

Definitions

- The *preimage* of a set, B, under some function, $f : A \to B$, is the set of all elements a in A such that $f(a)$ is in B.
- For example, if $f(x) = x^2$ then the preimage of $\{4\}$ would be $\{-2, 2\}$

Definitions

- A mapping $\phi : G \rightarrow H$ from a group, $(G, \odot)$, to another group, $(H, \oplus)$, is called a *homomorphism* if and only if for all a, b in G the following equation holds: $\phi(a \odot b) = \phi(a) \oplus \phi(b)$

Homomorphism

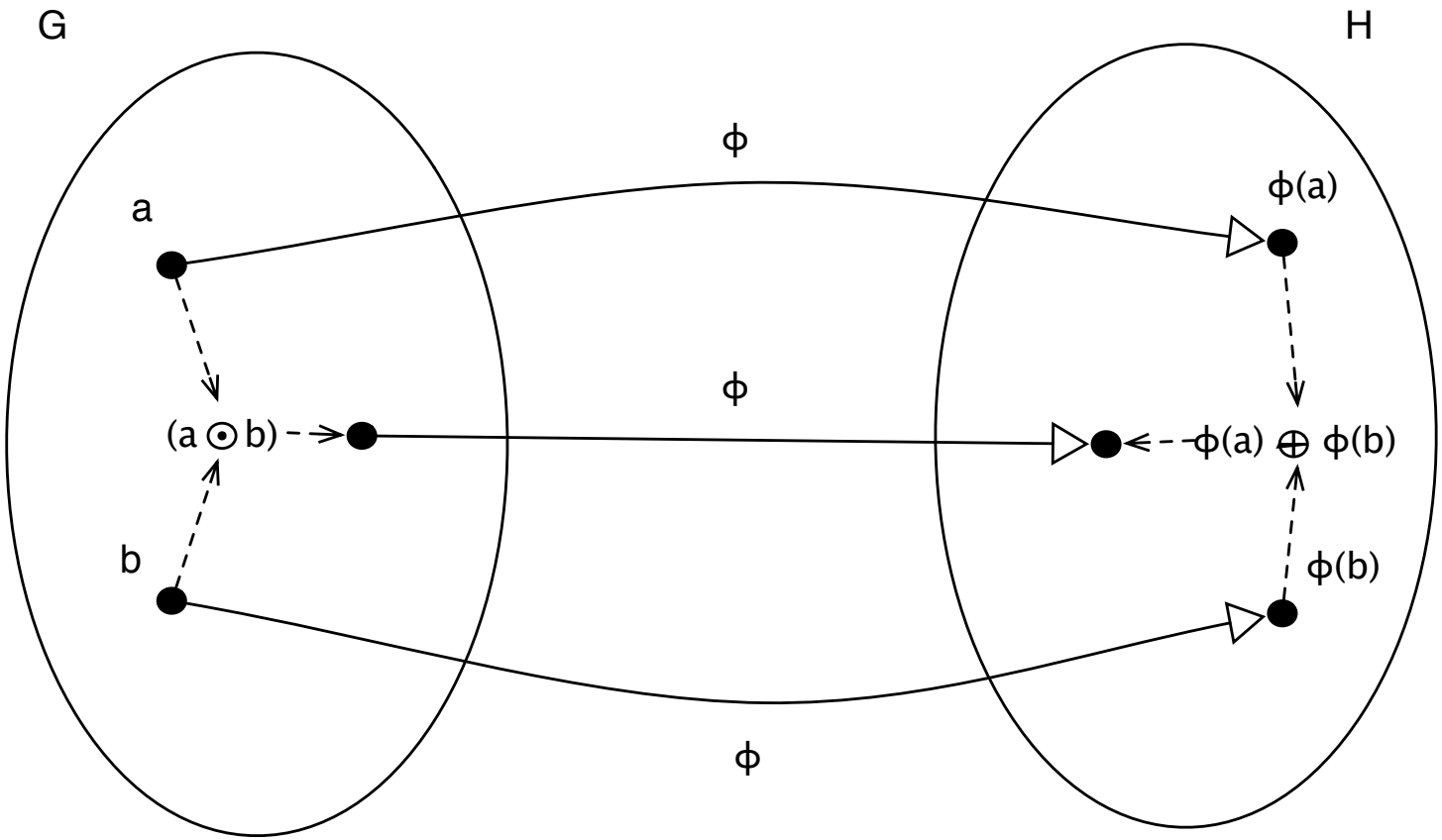

Figure : Diagram showing a homomorphism.

ZKCrypt

- Developed by Almedia et al.
- An optimizing cryptographic compiler.
- Utilizes *verified compilation* and *verifying compilation*.

ZKCrypt

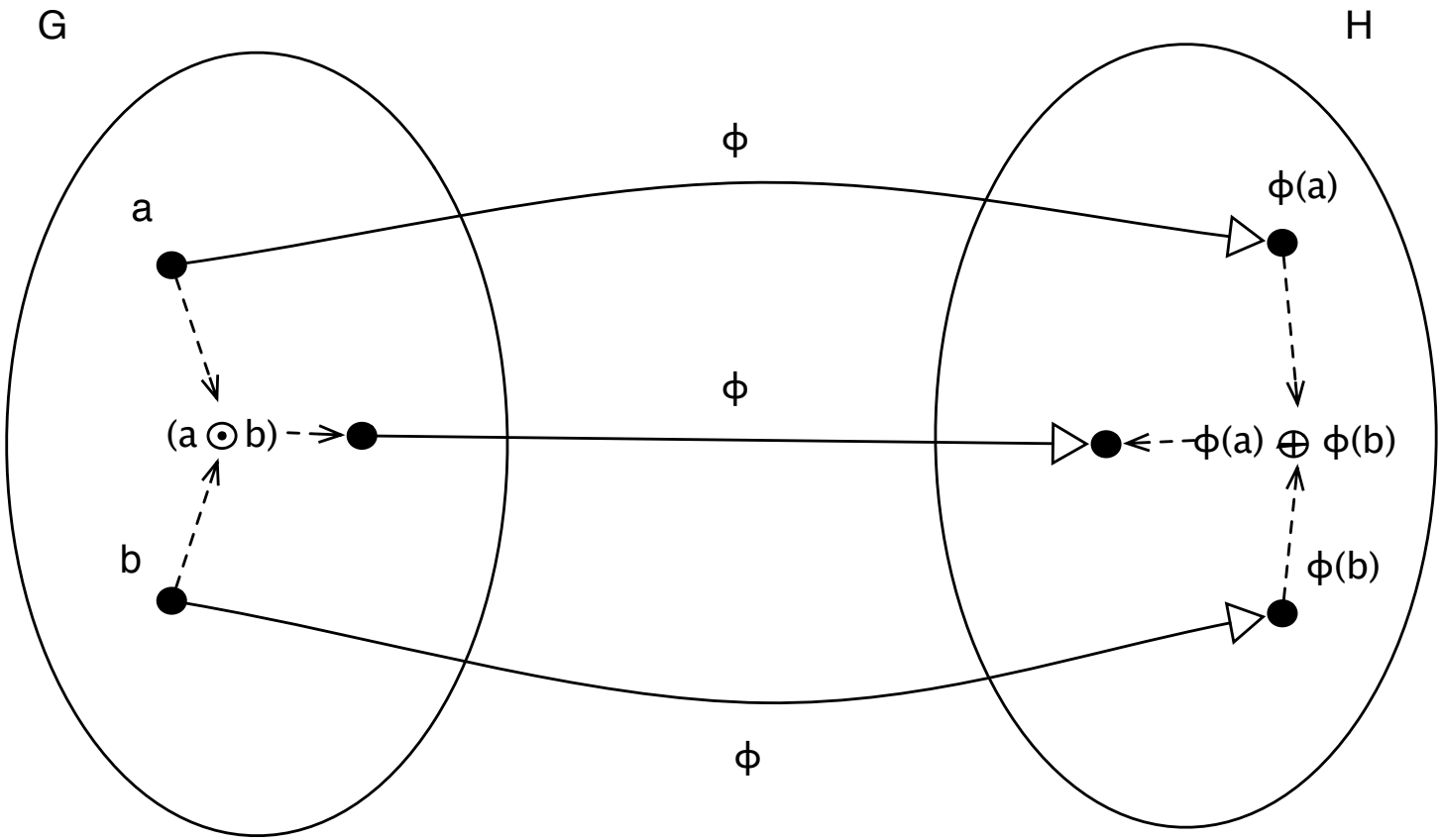

Figure : Diagram showing a homomorphism.

The Input

Consists of:

- Declarations block.
- Inputs block.
- Protocol block:
  - Homomorphism.
  - Relation.

ZKCrypt

- *Verified compilation* is where the correctness of the output is proven once-and-for-all.
- *Verifying compilation* is where the correctness of the output is checked on each run.

ZKPDL

- Developed by Meiklejohn et al.
- Primarily used in electronic cash.
- Split into independent two blocks: *computation* and *proof*.

The Input

- Computation block:
  - Given block.
  - Compute block.
- Proof block:
  - Given block.
  - Prove knowledge of block.
  - Such that block.

Outline

Final Thoughts

- ZKCrypt, more general purpose, performs optimizations.
- ZKPDL, more focused on e-cash.

References


J. B. Almeida, M. Barbosa, E. Bangerter, G. Barthe, S. Krenn, and S. Z. Béguelin.
Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols.
In *Proceedings of the 2012 ACM conference on Computer and communications security*, *CCS '12*, pages 488-500, ACM, New York, New York, USA, 2012.


S. Meiklejohn, C. Erway, A. Küpçü, T. Hinkle,and A. Lysyanskaya.
ZKPDL: a language-based system for efficient zero-knowledge proofs and electronic cash.
In Proceedings of the 19th USENIX conference on Security , *USENIX Security'10*, USENIX Association, Berkeley, CA, USA.

Thank you!

Questions?