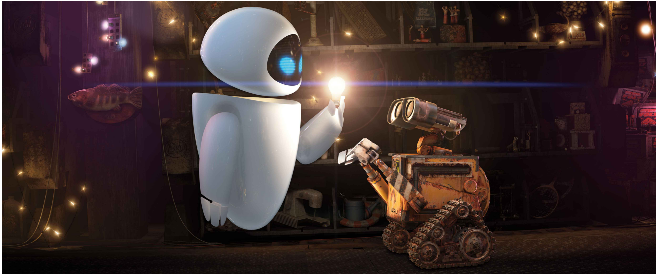
Zero Knowledge Proofs: ZK for all NP

MoL Research Project



Maaike Zwart & Suzanne van Wijk

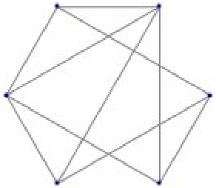
January 2015

Institute for Logic, Language and Computation

Aim: Zero Knowledge proofs for *all* NP-problems

- Find a ZK proof for graph colouring (G3C)
- Need: commitment schemes
- Use this proof to find a ZK proof for all NP-problems
- Do it yourself: find a direct ZK proof for some other NP-complete problems

Graph Colouring



Given a graph $G = \{V, E\}$, want:
$f : V \to \{R, B, G\}$
Such that:
$(u, v) \in E \implies f(u) \neq f(v)$

Finding a ZK proof for Graph Colouring

Recall...

Zero knowledge interactive proof:

- Completeness
- Soundness
- Zero-knowledge

Finding a ZK proof for Graph Colouring

Can you give me the colouring?

Finding a ZK proof for Graph Colouring

That's too much to ask!

Can you give me the colouring?

Finding a ZK proof for Graph Colouring

That's too much to ask!

What about just the colouring of two adjacent vertices?

Finding a ZK proof for Graph Colouring

Hmmm

What about just the colouring of two adjacent vertices?

Finding a ZK proof for Graph Colouring

What about just the colouring of two adjacent vertices?

No, still too much!

Finding a ZK proof for Graph Colouring

But..

What about just the colouring of two adjacent vertices?

Finding a ZK proof for Graph Colouring

What about just the colouring of two adjacent vertices?

I can when I first randomly permute the colours!

Finding a ZK proof for Graph Colouring

I can when I first randomly permute the colours!

Ok :)

Finding a ZK proof for Graph Colouring

I can when I first randomly permute the colours!

Wait! How do I know you don't lie about the colours?

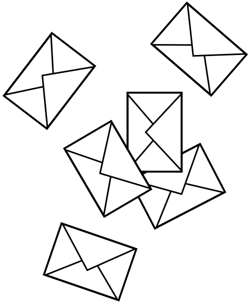
Finding a ZK proof for Graph Colouring

Wait! How do I know you don't lie about the colours?

Commitment schemes!

Commitment Schemes: Digital Envelopes

Prover should *commit* the whole colouring before the verifier asks the colour of two vertices.

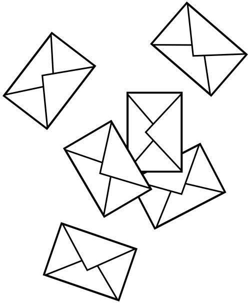


Commitment Schemes: Digital Envelopes

Prover should *commit* the whole colouring before the verifier asks the colour of two vertices.

Two phases: commit and reveal

- Commitment should be *secret* (non-transparent envelopes)
- The revealed information should be *unambiguous*



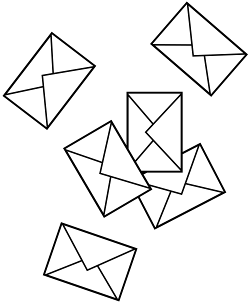
Commitment Schemes: Digital Envelopes

Prover should *commit* the whole colouring before the verifier asks the colour of two vertices.

Two phases: commit and reveal

- Commitment should be *secret* (non-transparent envelopes)
- The revealed information should be *unambiguous*

How secret / unambiguous? Computationally!



Commitment Schemes

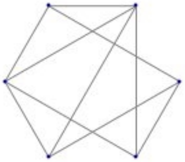
Digital examples:

- One-way functions: easy to compute $f(x)$ given f and x, but hard to compute x given f and $f(x)$.
- Discrete log: easy to compute g^h mod p given g, h, p, but hard to compute h given g, p and g^h

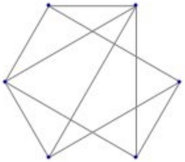
Real-life examples: exercise!

Full Zero-Knowlege proof of Graph Colouring

Prover:



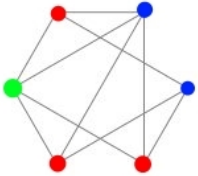
Verifier:



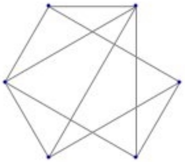
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph

Prover:



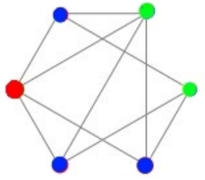
Verifier:



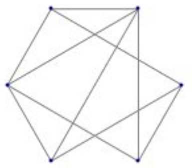
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours

Prover:



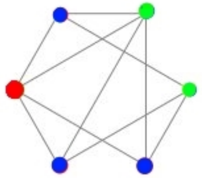
Verifier:



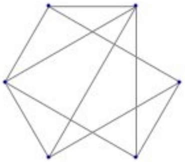
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring

Prover:



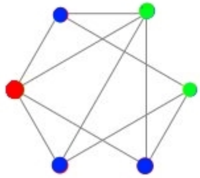
Verifier:



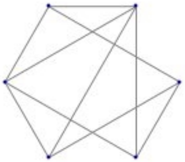
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices

Prover:



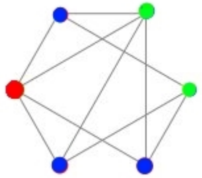
Verifier:



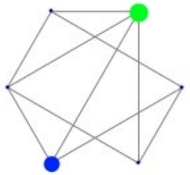
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices

Prover:



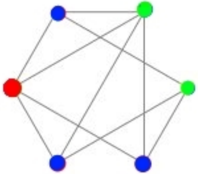
Verifier:



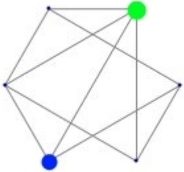
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment

Prover:



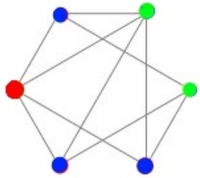
Verifier:



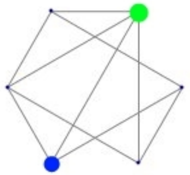
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different

Prover:



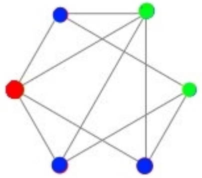
Verifier:



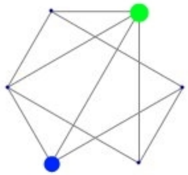
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different
- Verifier accepts/rejects on the outcome of the two checks

Prover:



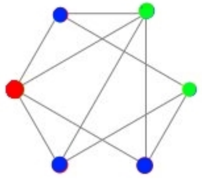
Verifier:



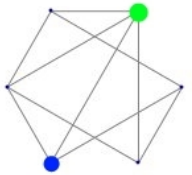
Full Zero-Knowlege proof of Graph Colouring

- Prover colours the graph
- Prover takes a random permutation of the colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different
- Verifier accepts/rejects on the outcome of the two checks
- Repeat until Verifier is fully satisfied

Prover:



Verifier:



ZK proof of G3C: Analysis

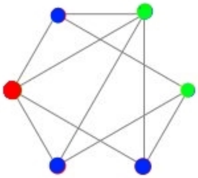
- Completeness error?
- Soundness error?
- Zero-Knowledge?

ZK proof of G3C: Analysis

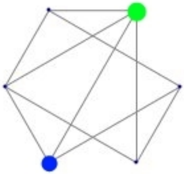
- **Completeness error?**
- Soundness error?
- Zero-Knowledge?

- Prover takes a random permutation of the colours
- Prover colours the graph with the permuted colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different
- Verifier accepts/rejects on the outcome of the two checks
- Repeat until Verifier is fully satisfied

Prover:



Verifier:

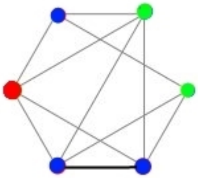


ZK proof of G3C: Analysis

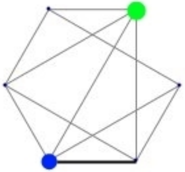
- Completeness error?
- **Soundness error?**
- Zero-Knowledge?

- Prover takes a random permutation of the colours
- Prover colours the graph with the permuted colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different
- Verifier accepts/rejects on the outcome of the two checks
- Repeat until Verifier is fully satisfied

Prover:



Verifier:

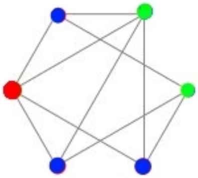


ZK proof of G3C: Analysis

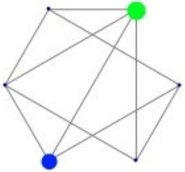
- Completeness error?
- Soundness error?
- **Zero-Knowledge?**

- Prover takes a random permutation of the colours
- Prover colours the graph with the permuted colours
- Prover commits the colouring
- Verifier asks for the colour of two adjacent vertices
- Prover reveals the colour of these vertices
- Verifier checks the commitment
- Verifier checks that the colours are different
- Verifier accepts/rejects on the outcome of the two checks
- Repeat until Verifier is fully satisfied

Prover:



Verifier:



NP-completeness

Note: G3C is NP-complete.

That means, that every NP problem $x \in L$ can be translated to a question $f(x) \in G3C$.
(How? Follow a course on complexity theory! But let me give you a flavour:)

NP-completeness

Note: G3C is NP-complete.

That means, that every NP problem $x \in L$ can be translated to a question $f(x) \in G3C$.
(How? Follow a course on complexity theory! But let me give you a flavour:)

Cook-Levin Theorem: SAT is NP-complete.
Proof: Fiddle with TM.

NP-completeness

Note: G3C is NP-complete.

That means, that every NP problem $x \in L$ can be translated to a question $f(x) \in G3C$.
(How? Follow a course on complexity theory! But let me give you a flavour:)

Cook-Levin Theorem: SAT is NP-complete.
Proof: Fiddle with TM.

Next: Reduce SAT to L to prove L is NP-complete too.

Web of reductions

2.4. The Web of Reductions 51

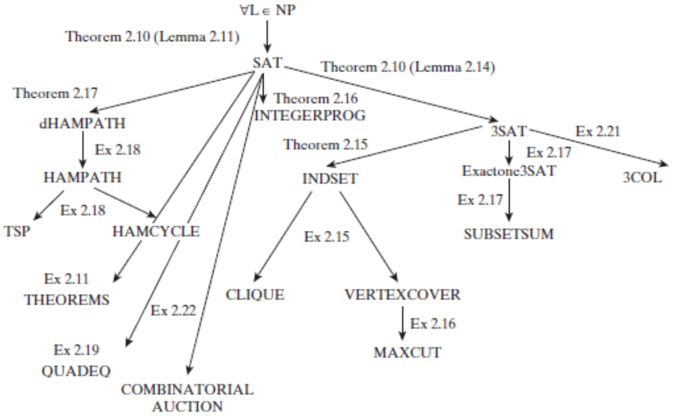


Figure 2.4. Web of reductions between the **NP**-completeness problems described in this chapter and the exercises. Thousands more are known. [1]

[1] image taken from Arora, Barak, Computational Complexity (2009)

ZK proofs for all NP

A Zero-Knowledge proof for any L in NP:

- Reduce L to G3C (find f s.t. $x \in L$ iff $f(x) \in G3C$).
- Check that knowing a witness for $f(x) \in G3C$ implies knowing a witness for $x \in L$ (this usually follows immediately from f)
- Execute the Zero-Knowledge proof for G3C.

ZK proofs for all NP

A Zero-Knowledge proof for any L in NP:

- Reduce L to G3C (find f s.t. $x \in L$ iff $f(x) \in G3C$).
- Check that knowing a witness for $f(x) \in G3C$ implies knowing a witness for $x \in L$ (this usually follows immediately from f)
- Execute the Zero-Knowledge proof for G3C.

Anticlimax?
Final Assignment: Choose an NP-complete problem (out of some given) and find a direct Zero-Knowledge proof for it.