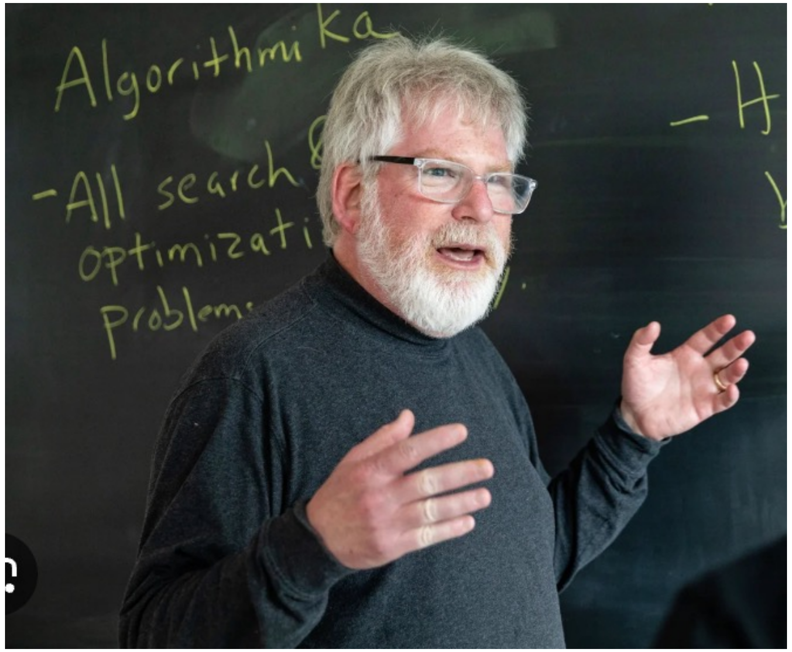
Foundations of Cryptography. Lecture 2: Cryptomania

Anna Lysyanskaya

Last Time: One-Way Functions and Minicrypt



- Definitions of security for
  - Symmetric encryption
  - One-way functions
  - Pseudorandom generators
  - Pseudorandom functions
  - Block ciphers
- Concepts: indistinguishability
- Theorems: Existence of OWF is necessary and sufficient for symmetric encryption, PRGs, PRFs, and block ciphers.
- Minicrypt: everything you can construct from a one-way function
  - One of five of Impagliazzo’s possible worlds

Today: Cryptomania

- Cryptomania = world in which more sophisticated cryptography is possible
- OWFs exist, and more
- Example of a cryptomania resident: public-key encryption
  - Impagliazzo and Rudich showed that you cannot build public-key encryption from a OWF.
- What do we need to achieve public-key encryption?
  - Definition of security
  - Construction – it will use OWFs enhanced with a trapdoor, and zero-knowledge proofs
  - Proof of security of the construction

Today: Cryptomania

- Zero-knowledge proofs
  - Definition (high level)
  - Construction for an NP-complete language
  - Another flavor: non-interactive zero-knowledge proof (NIZK)

- Public-key encryption: definition

- Trapdoor permutation (aka OWP with a trapdoor)
  - Definition
  - Examples

- Construct public-key encryption from NIZK and TDPs
  - Very theoretical construction, don’t use it in practice! But helps understand proofs of security.

- Look at practical constructions and try to make sense of them using our theoretical tools

Zero-Knowledge Proof: Idea

- Two parties: a Prover and a Verifier
- Prover’s input is a theorem X and its proof W
- Verifier just has the theorem X
- How does the Prover convince the Verifier that the theorem holds?
    - Obvious idea: reveal the proof W
    - But what’s the fun in that? You don’t want to give away your proof, you want your friend to find it herself!

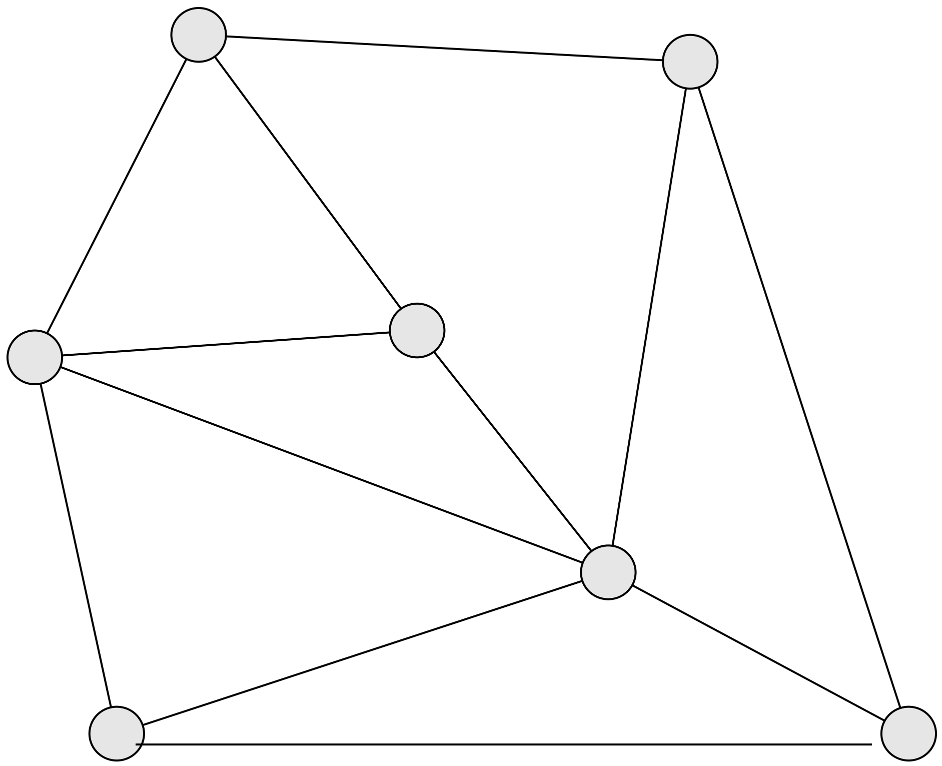
- How does the Prover convince the Verifier that the theorem holds without revealing anything about the proof?
    - Use a zero-knowledge proof!

Zero-knowledge proofs:
a crash course

Can you 3-color a graph?

1. Each vertex colored red, green or blue

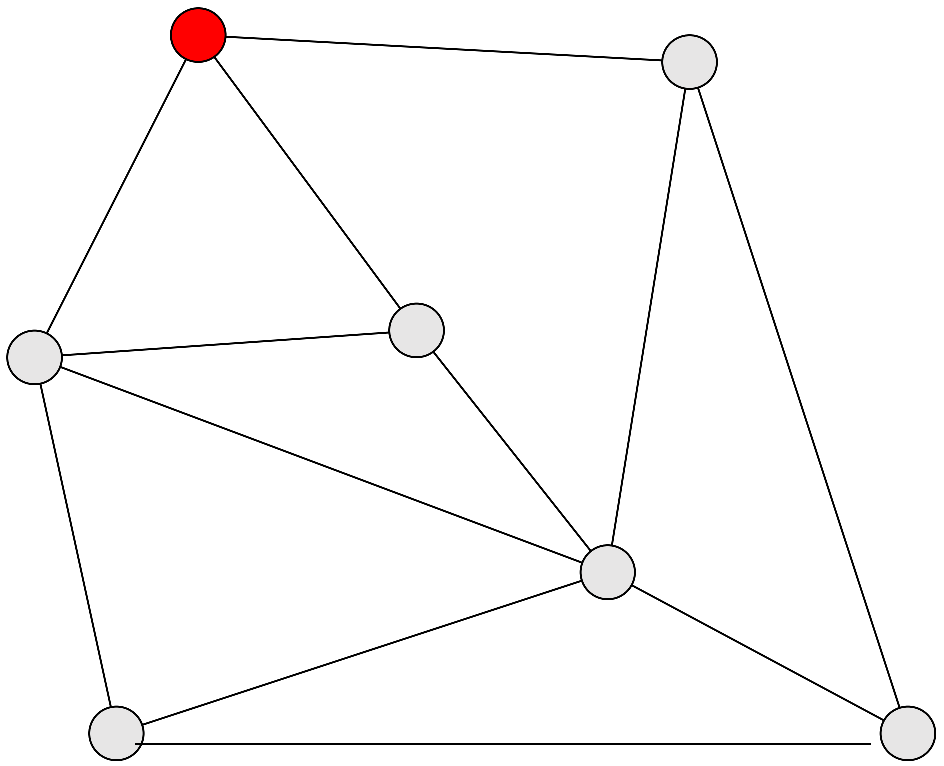
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

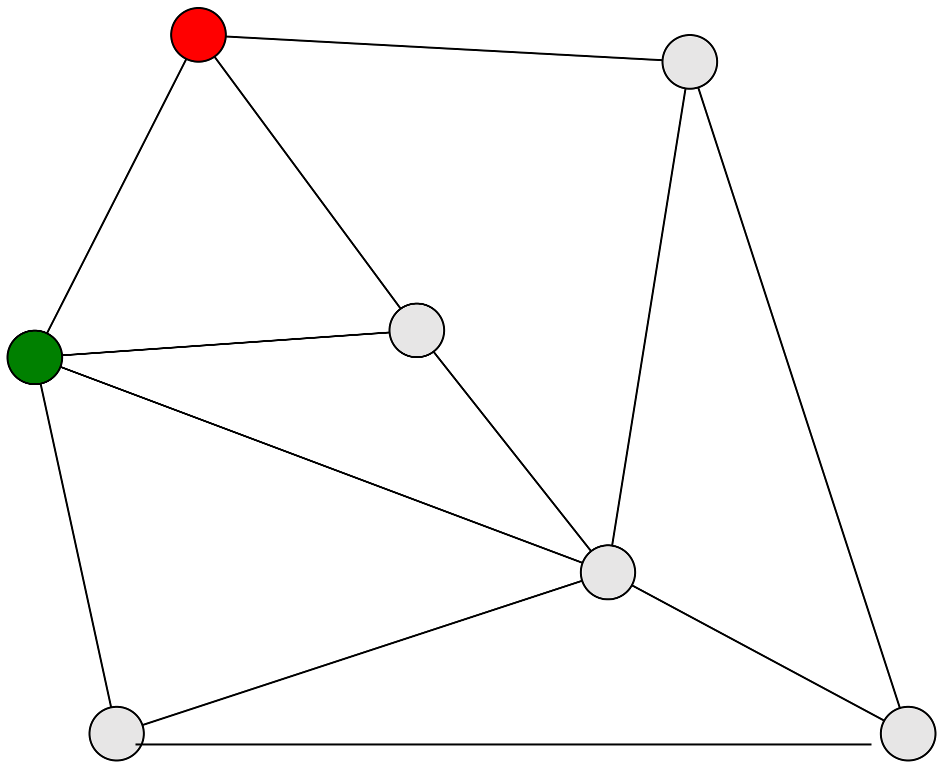
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

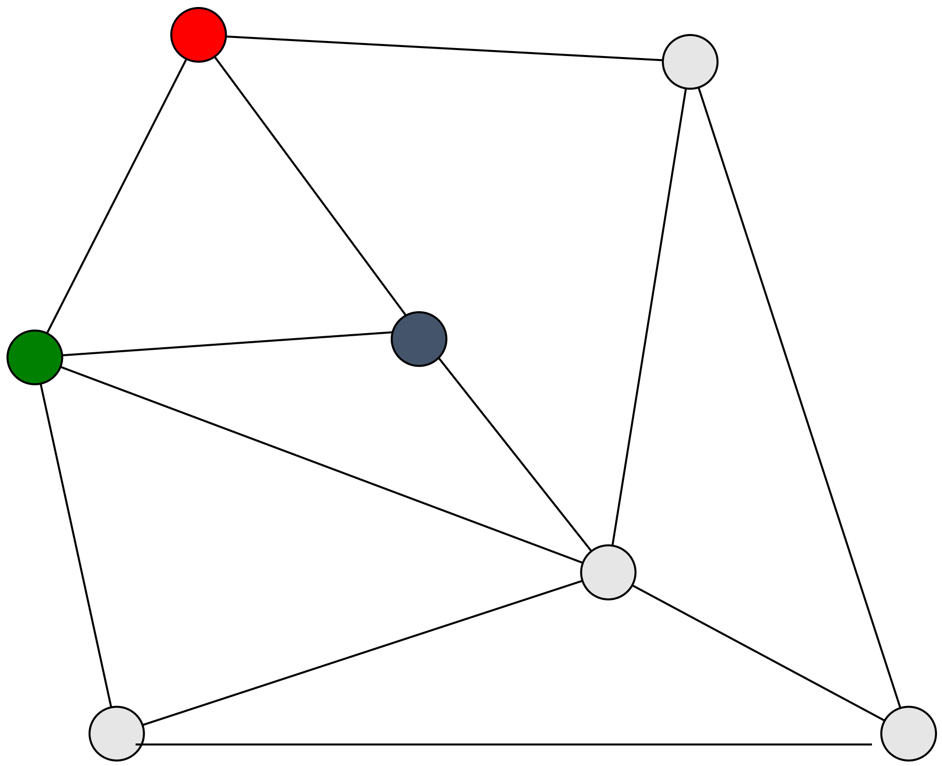
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

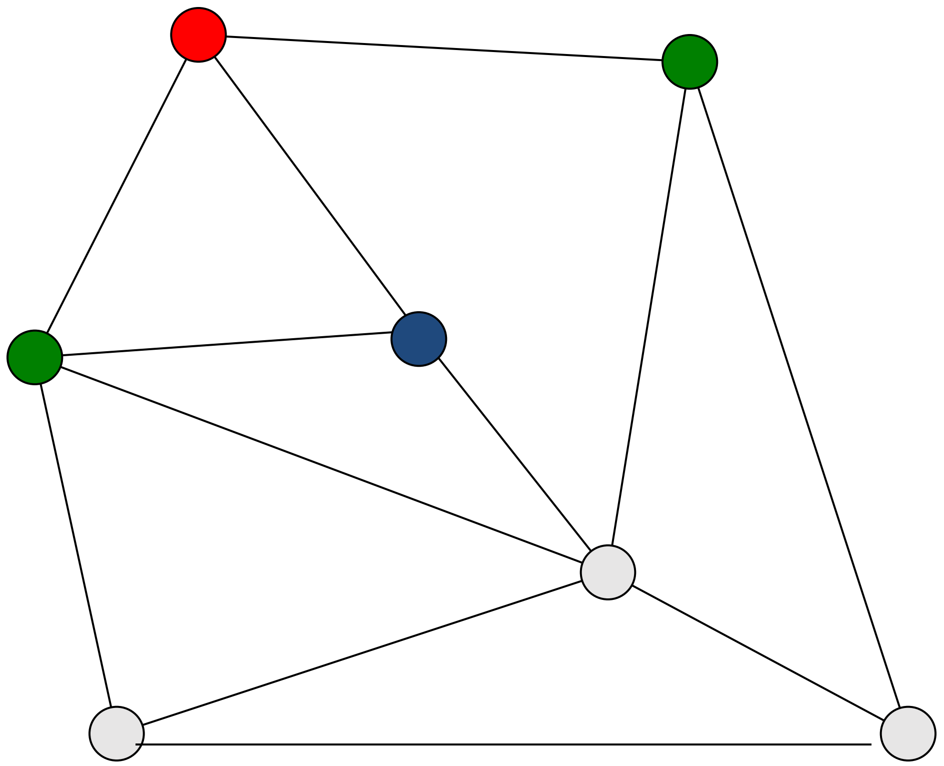
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

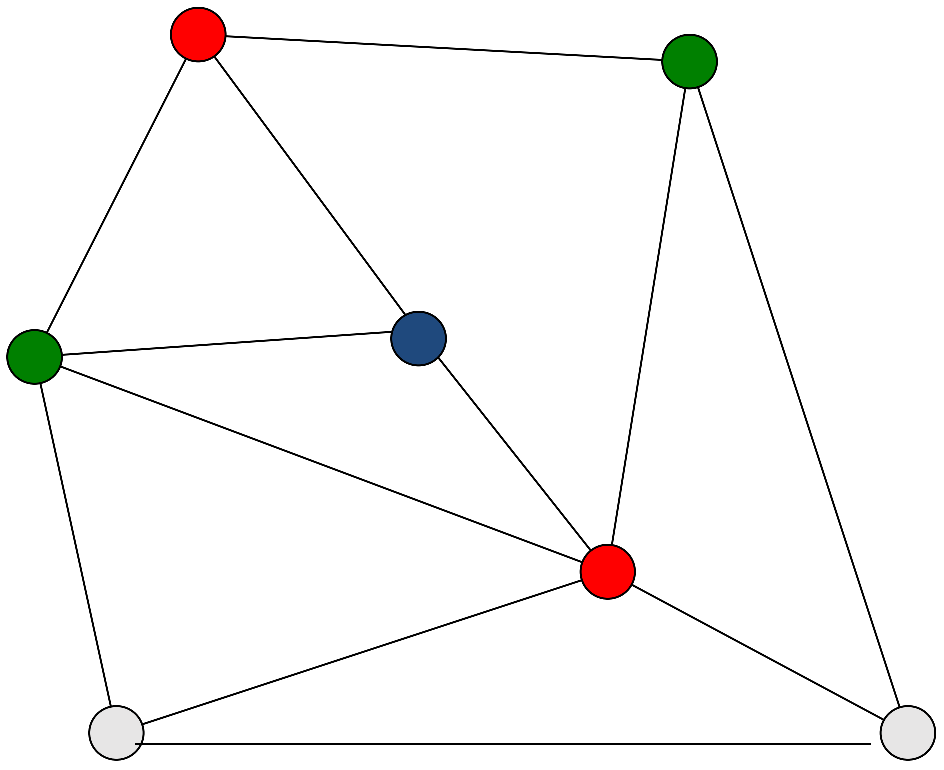
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

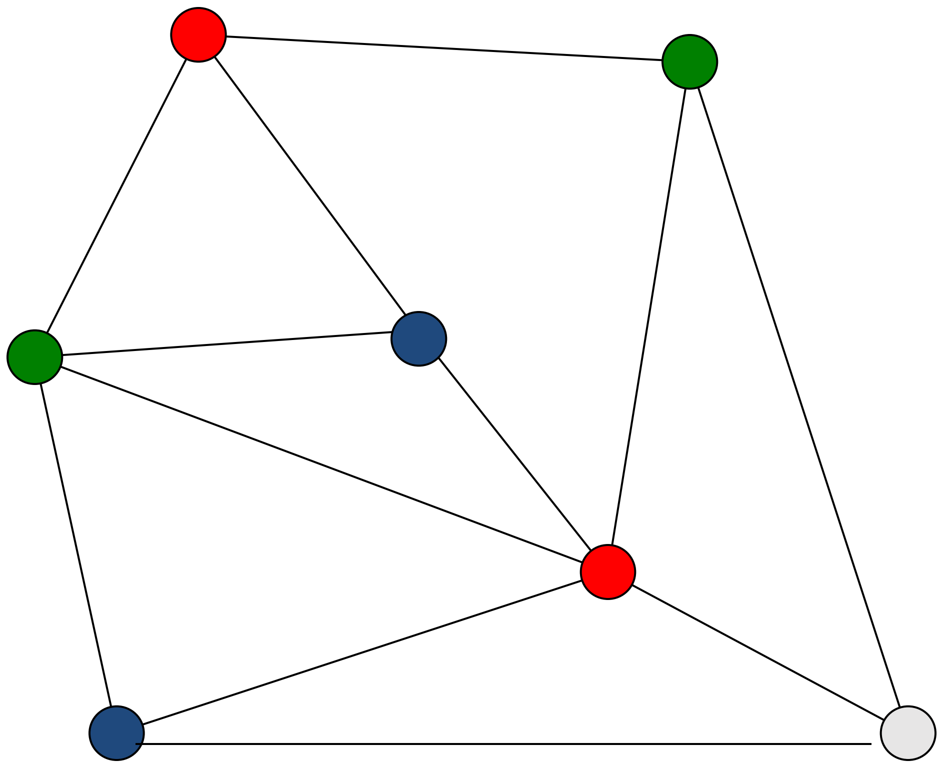
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

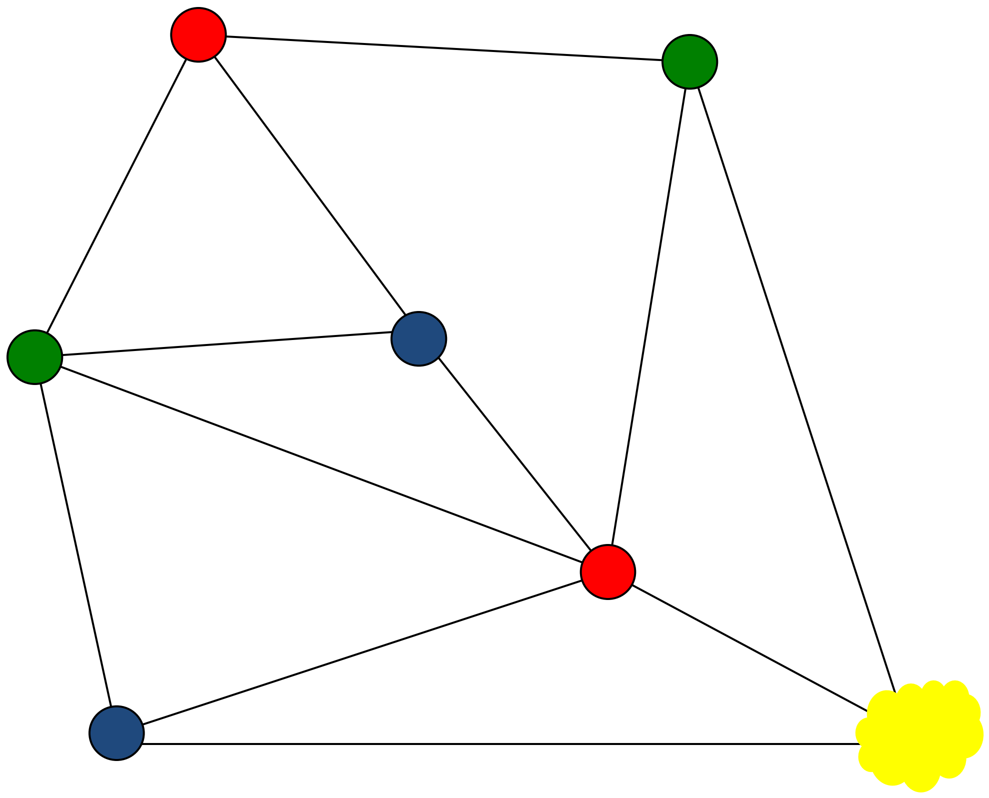
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

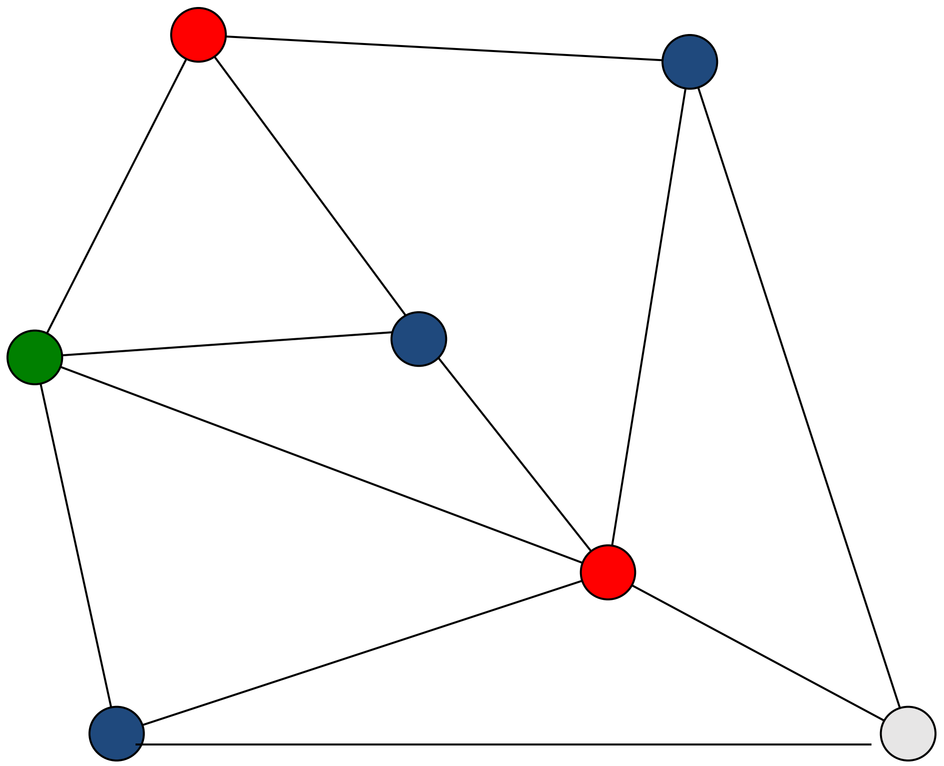
2. No monochromatic edges



Can you 3-color a graph?

1. Each vertex colored red, green or blue

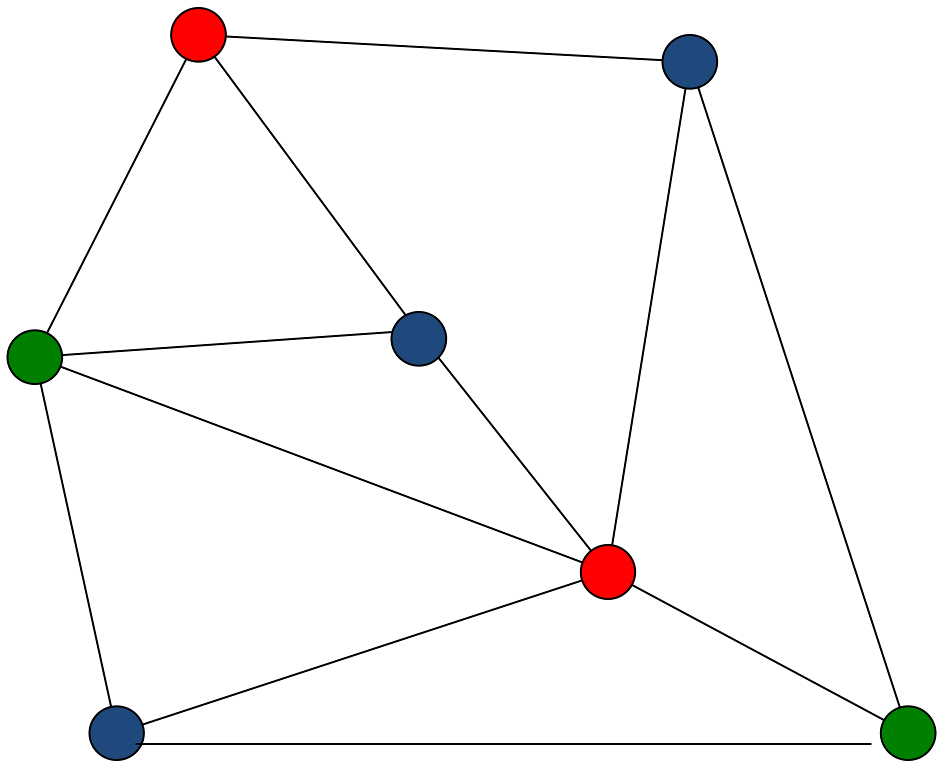
2. No monochromatic edges



Can you 3-color a graph?

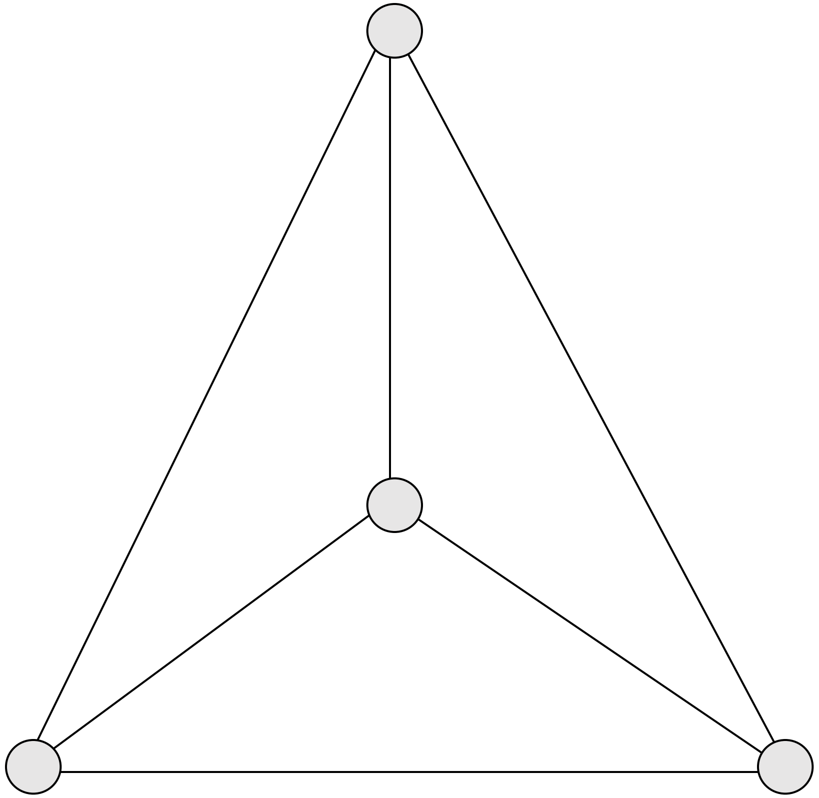
1. Each vertex colored red, green or blue

2. No monochromatic edges

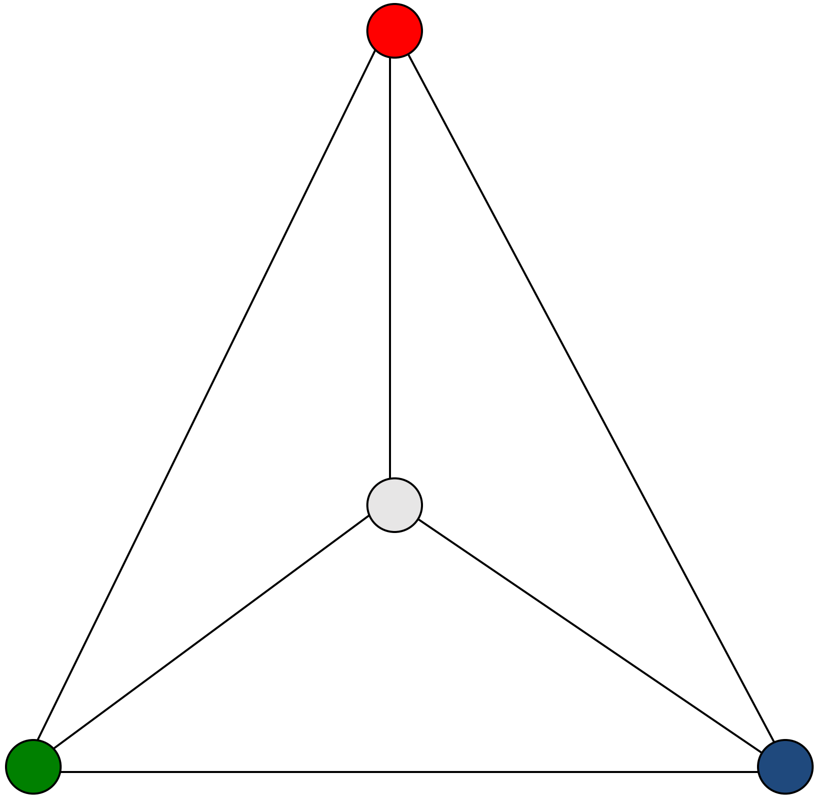


Is every graph 3-colorable?

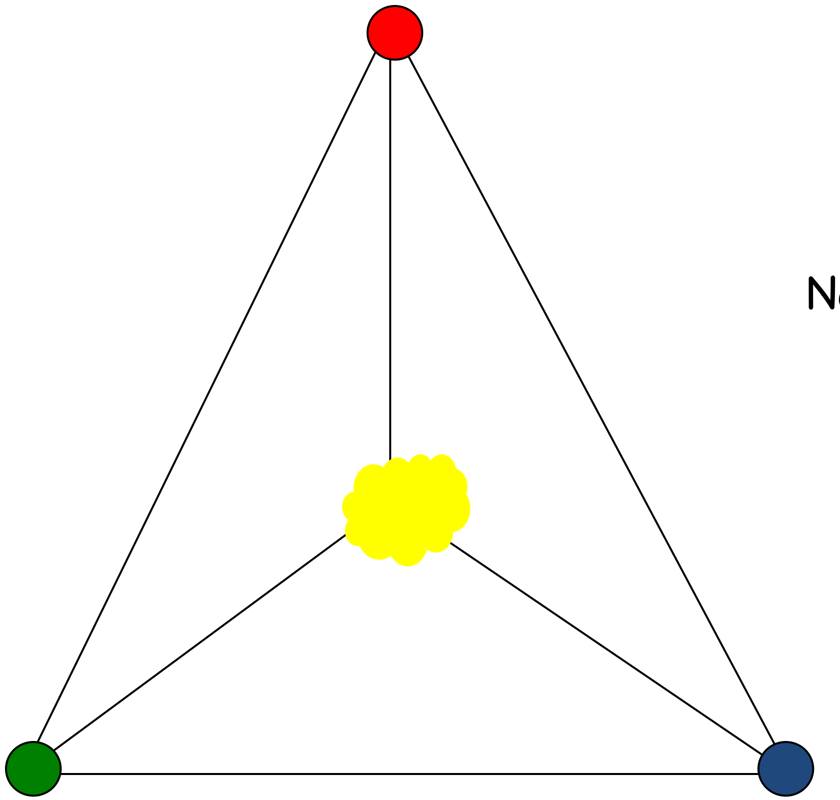
Is every graph 3-colorable?



Is every graph 3-colorable?

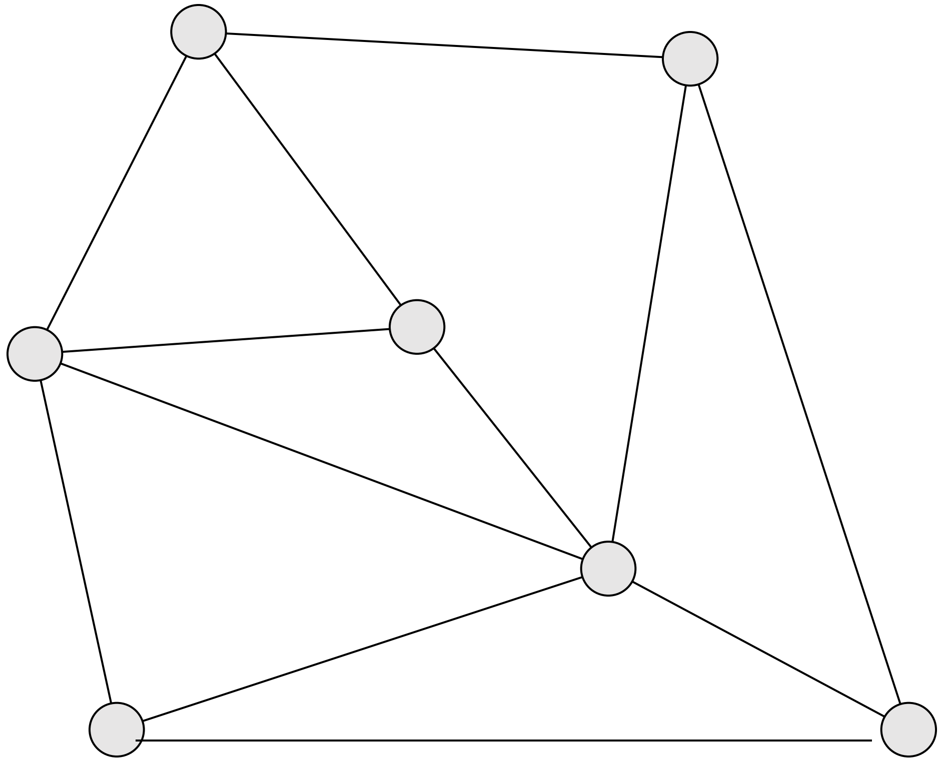


Is every graph 3-colorable?



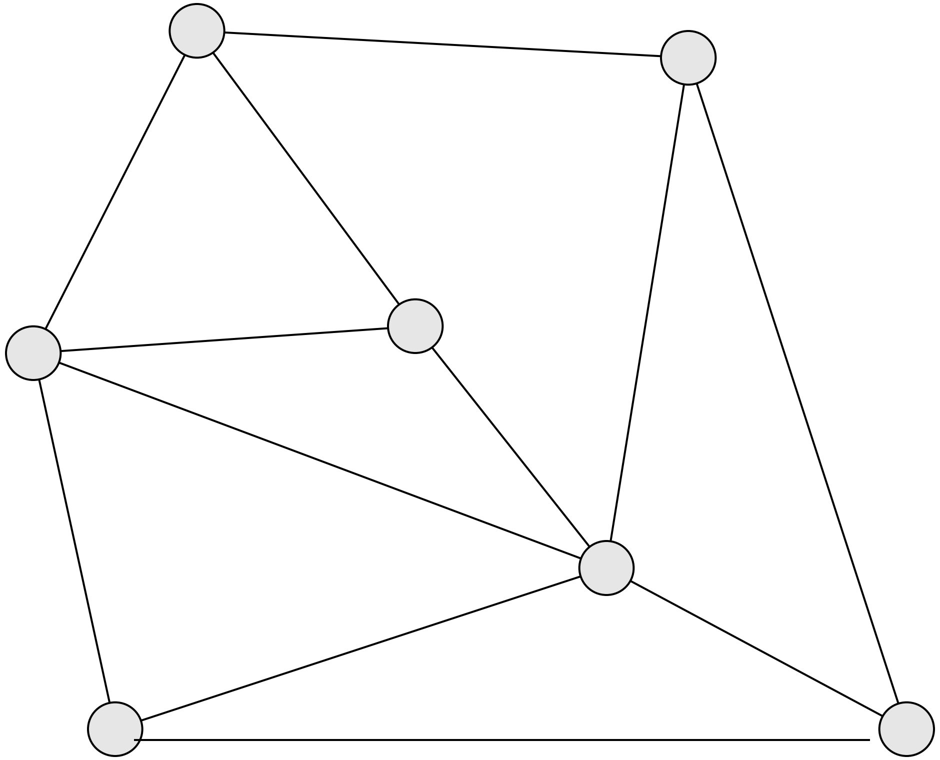
No…

Zero-knowledge proof of 3-colorability

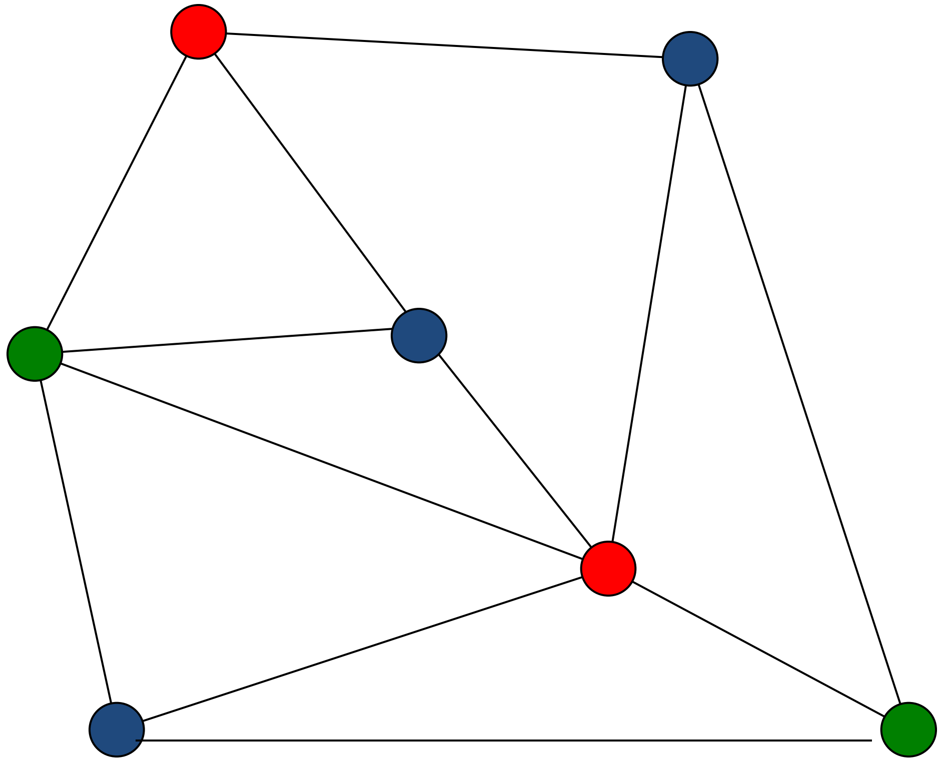


Zero-knowledge proof of 3-colorability

Prover

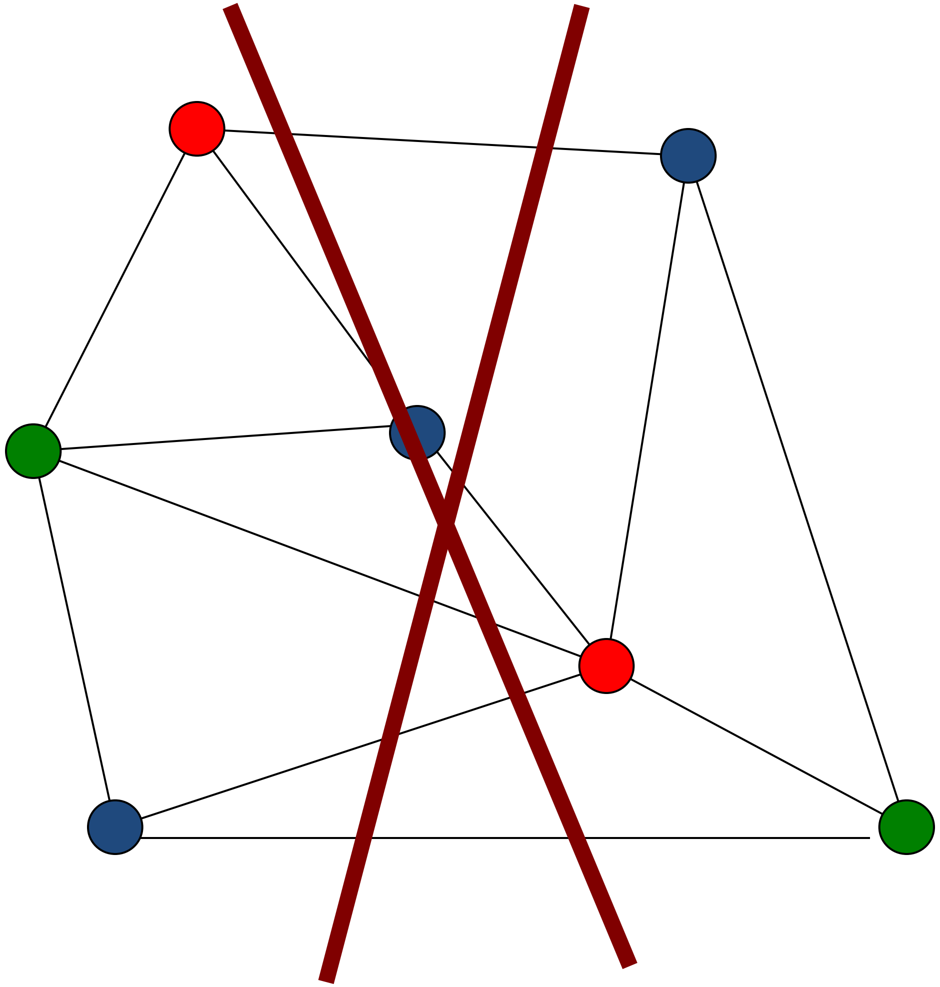


Zero-knowledge proof of 3-colorability



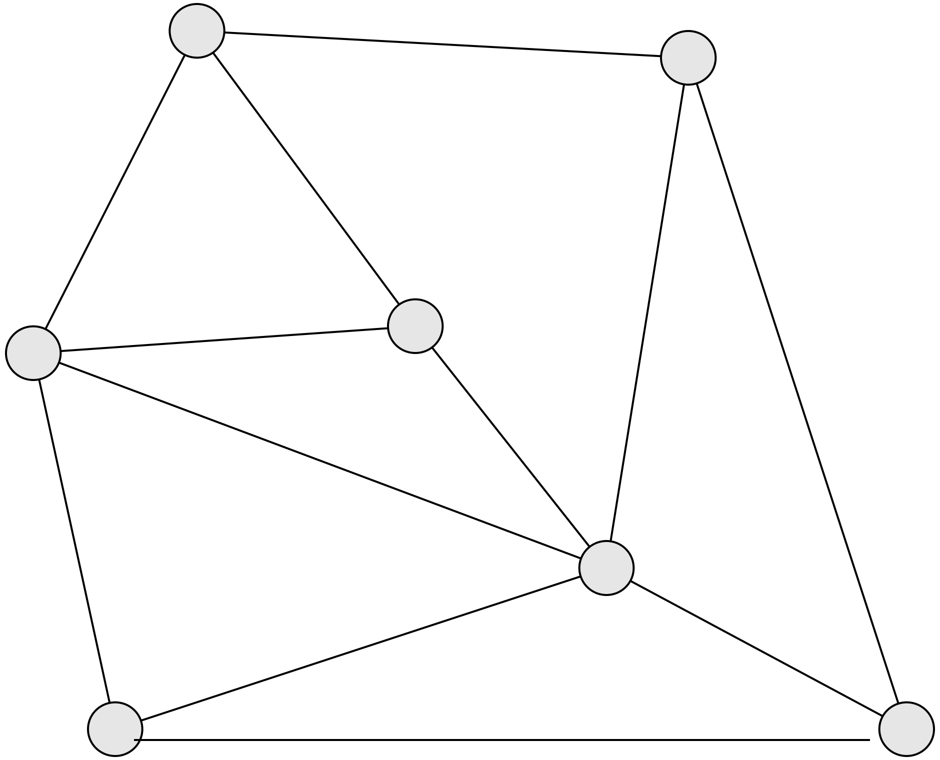
Prover

Zero-knowledge proof of 3-colorability

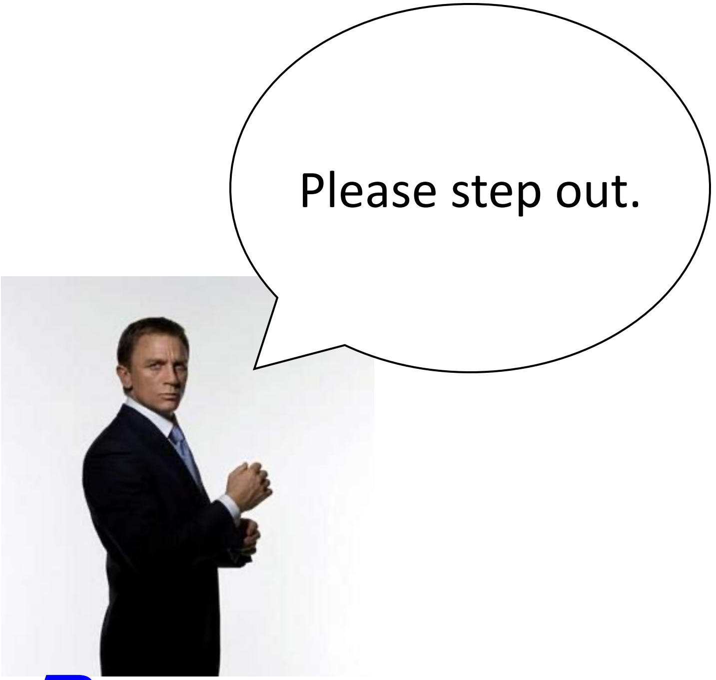


Zero-knowledge proof of 3-colorability

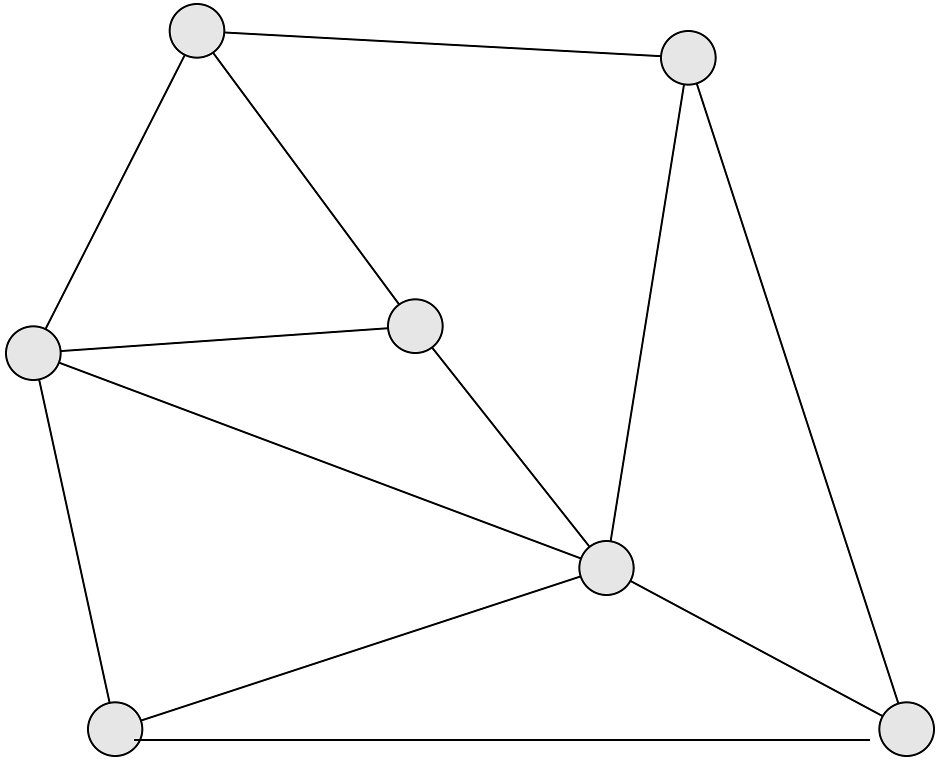
Prover



Zero-knowledge proof of 3-colorability

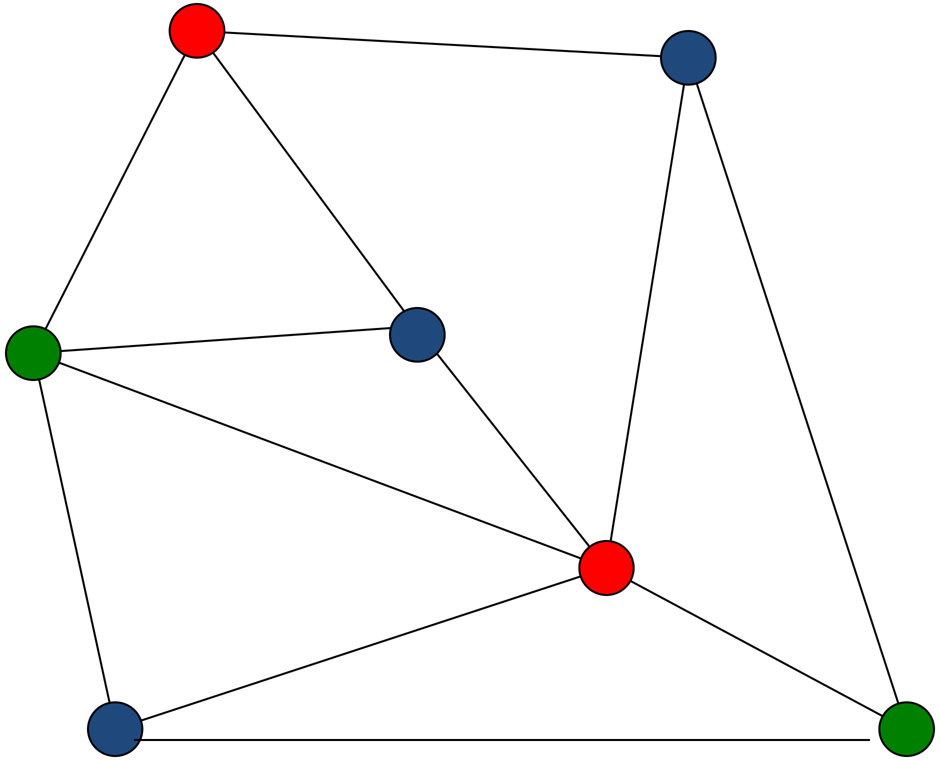


Prover

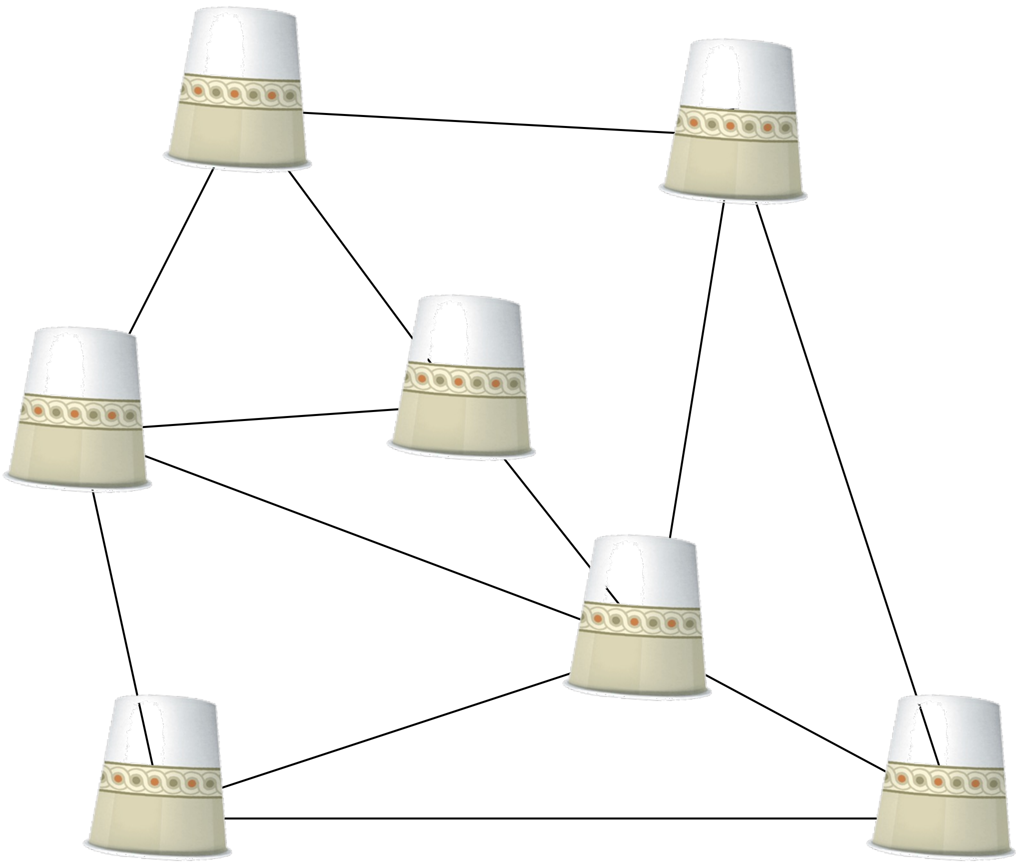


Zero-knowledge proof of 3-colorability

Prover

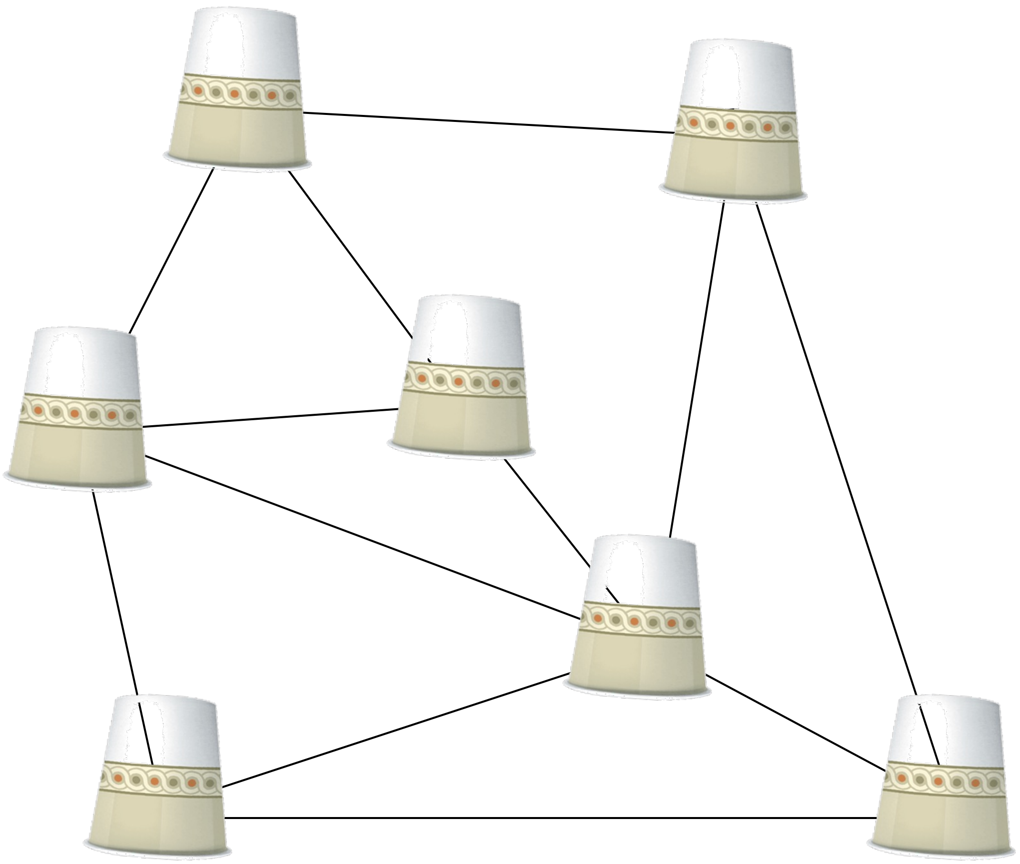


Zero-knowledge proof of 3-colorability

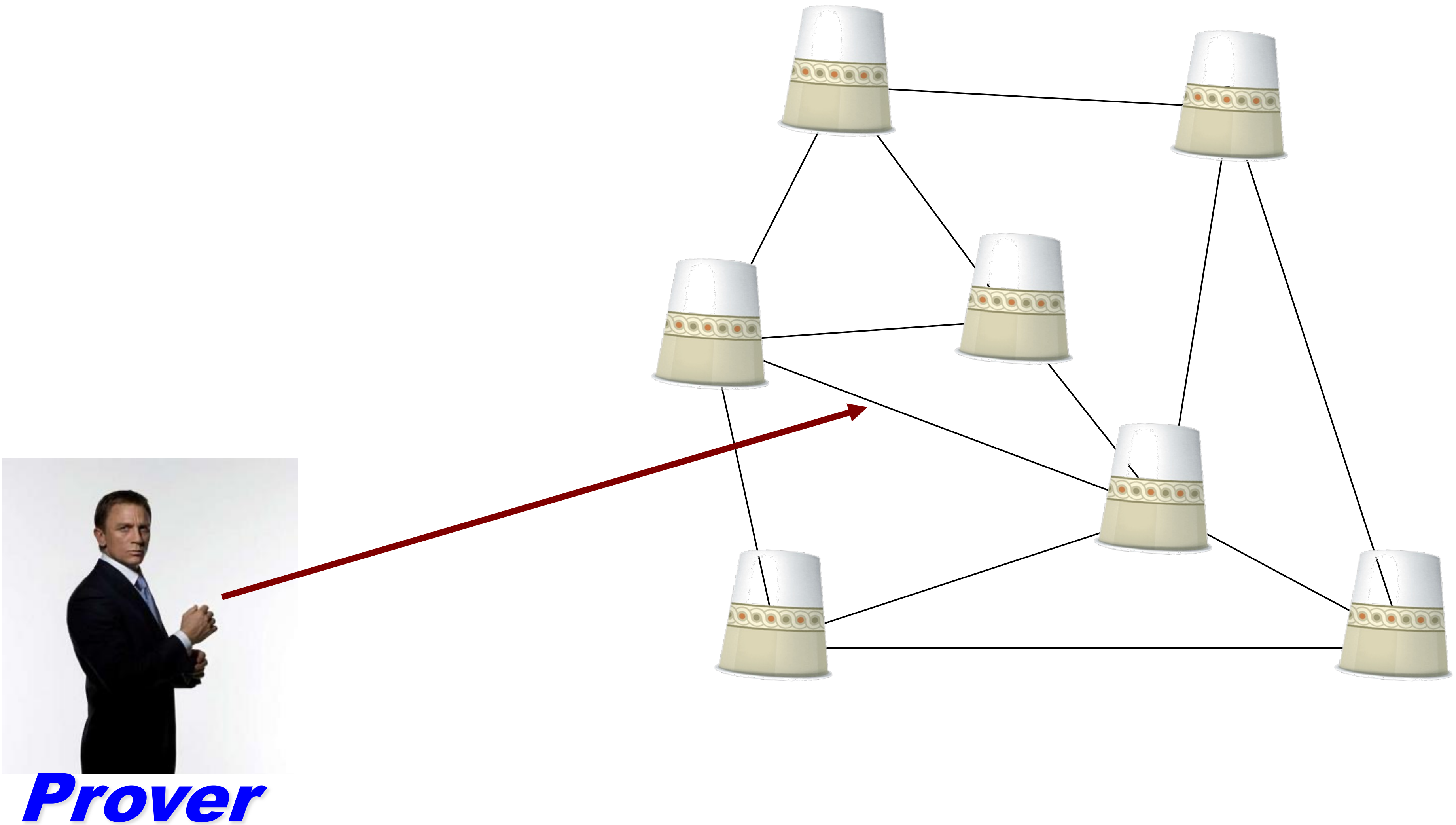


Prover

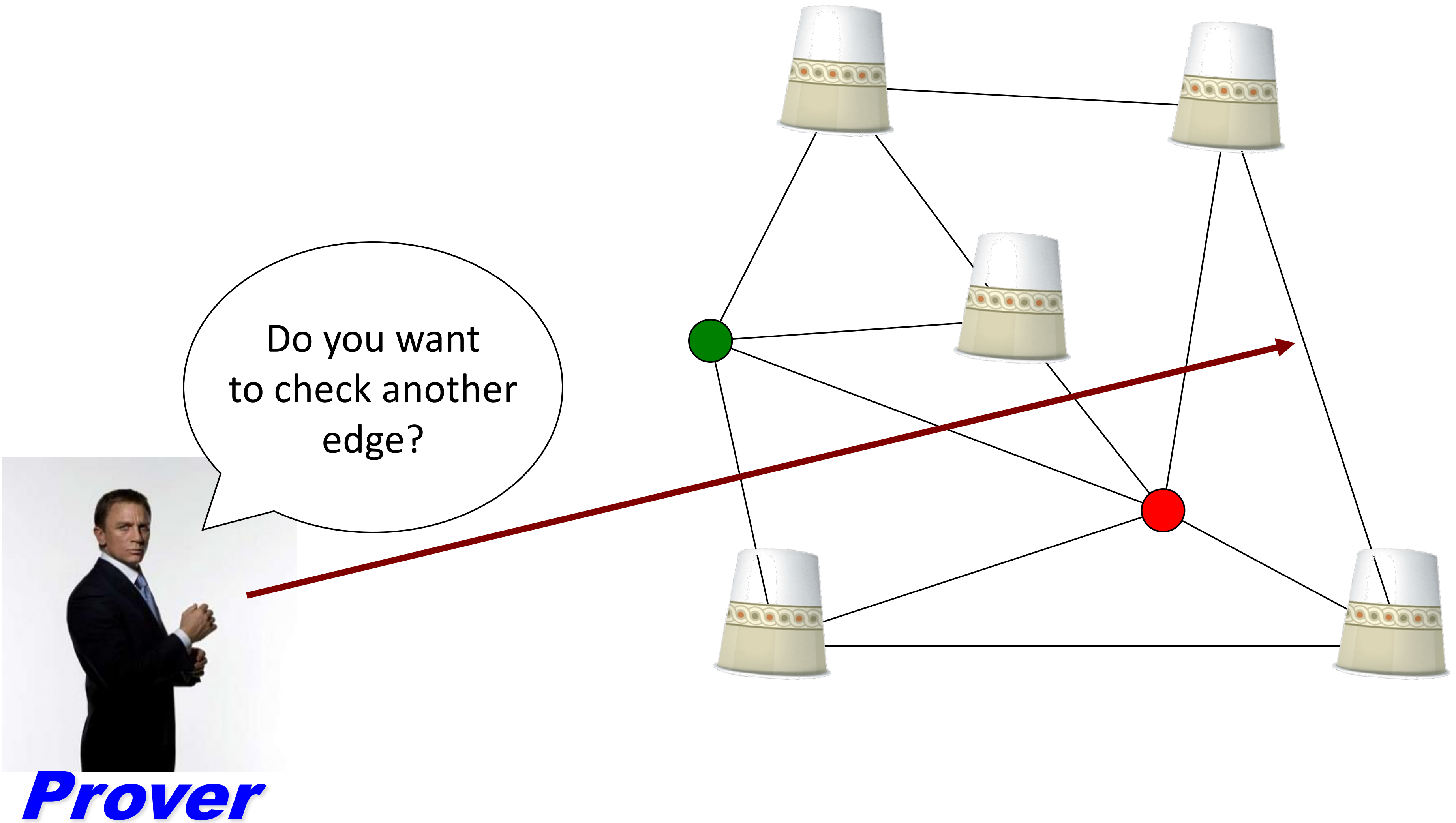
Zero-knowledge proof of 3-colorability



Zero-knowledge proof of 3-colorability

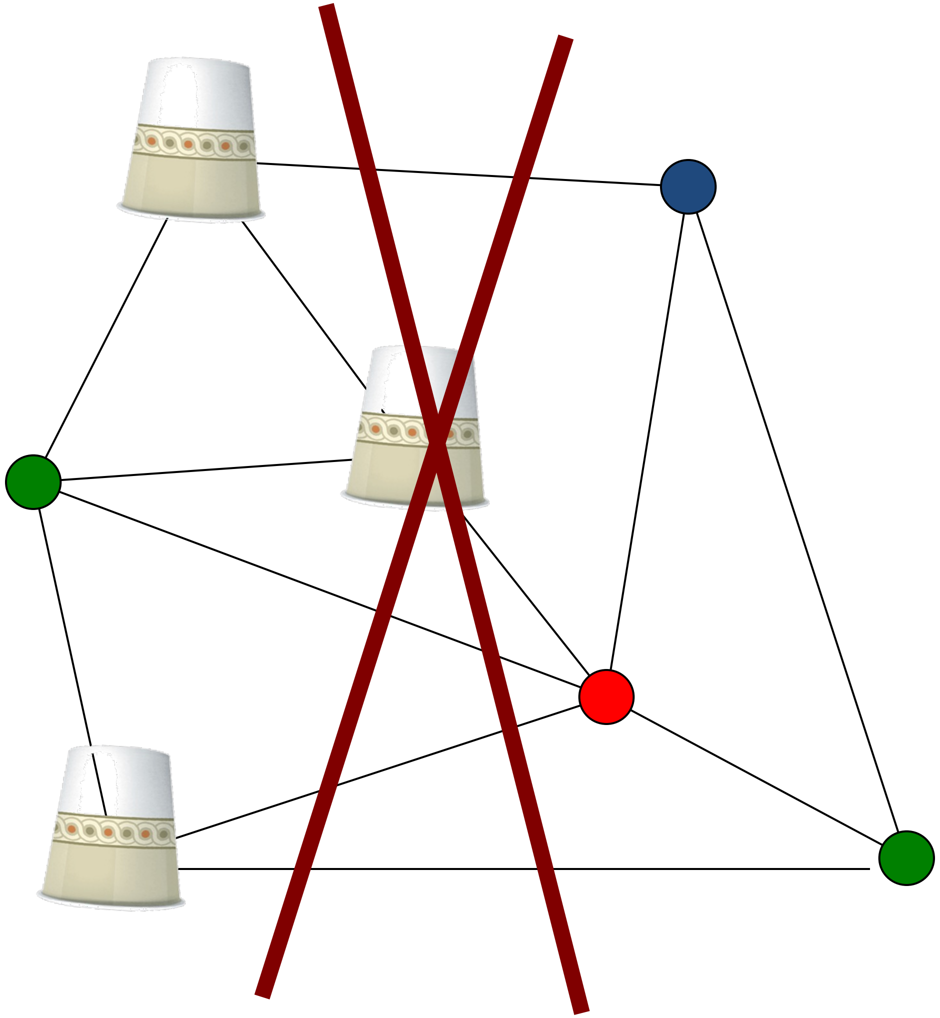


Zero-knowledge proof of 3-colorability



Zero-knowledge proof of 3-colorability

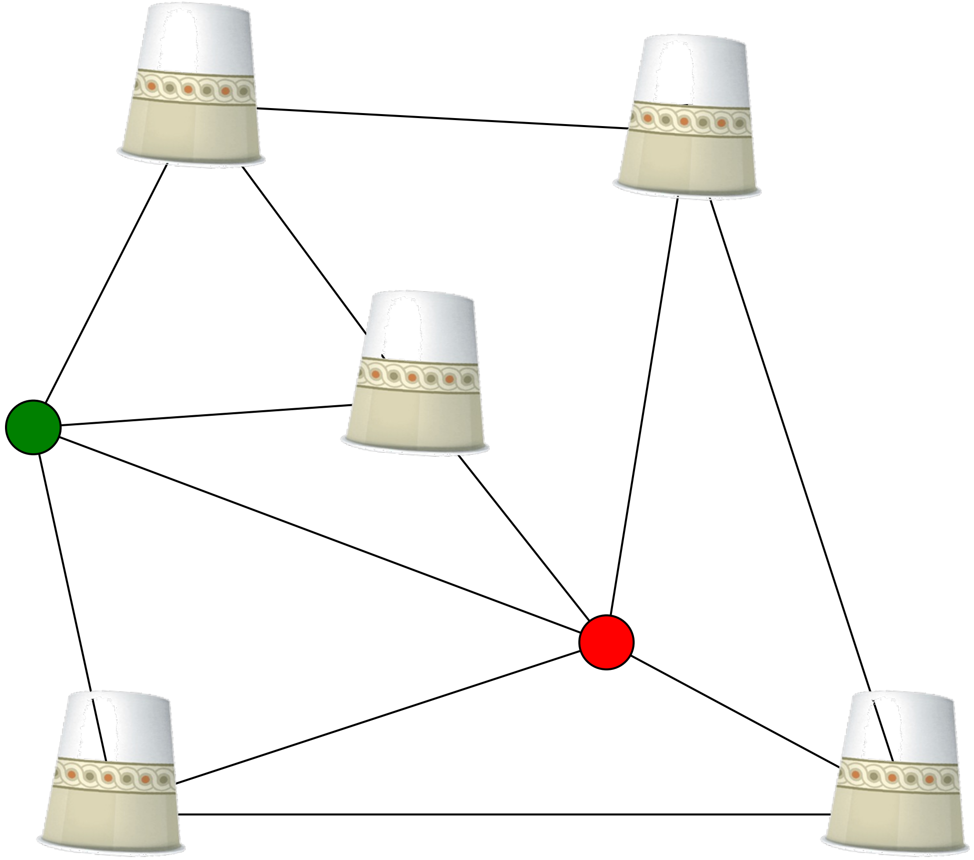
Prover



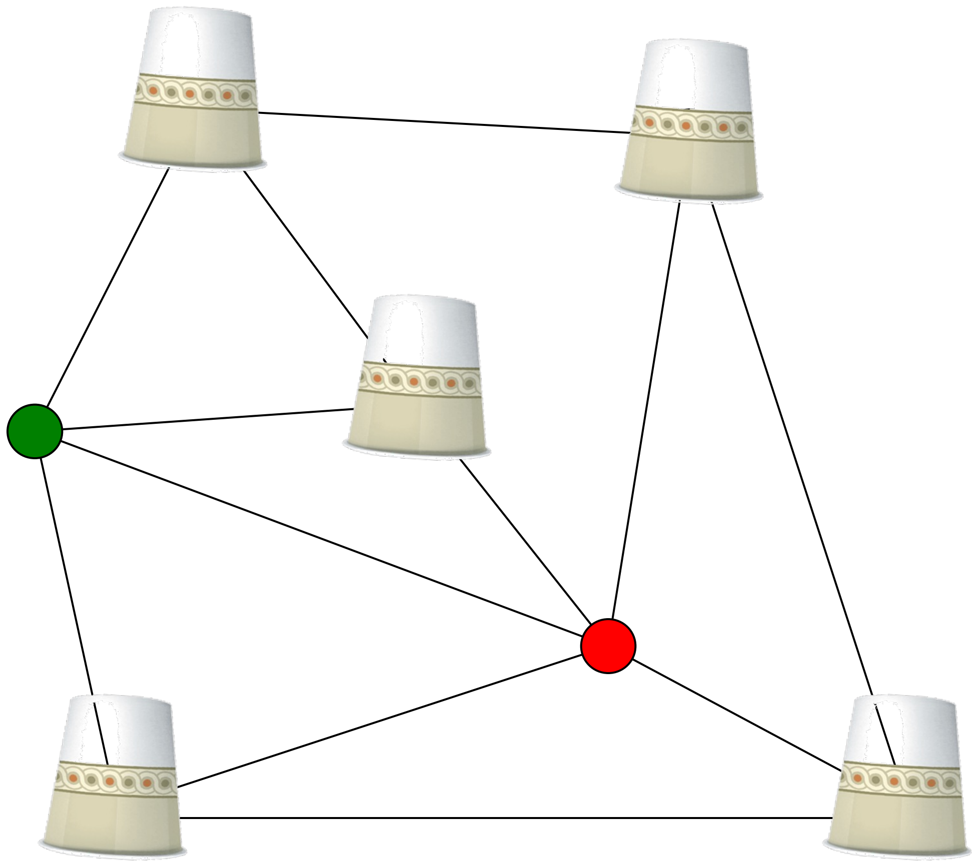
Zero-knowledge proof of 3-colorability

Please step out.

Prover

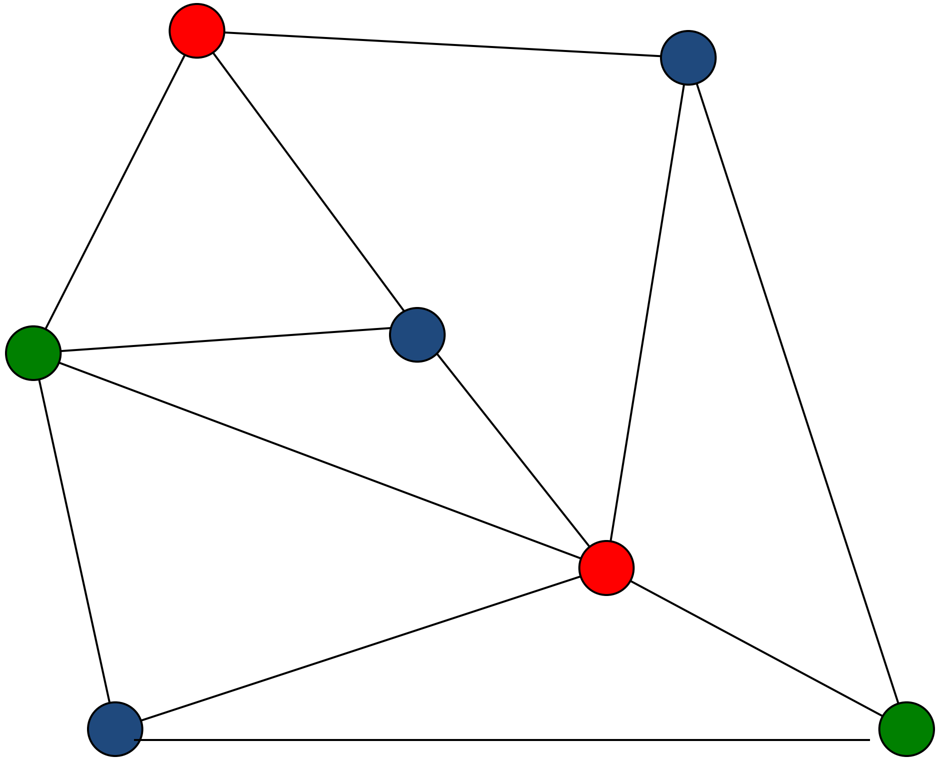


Zero-knowledge proof of 3-colorability



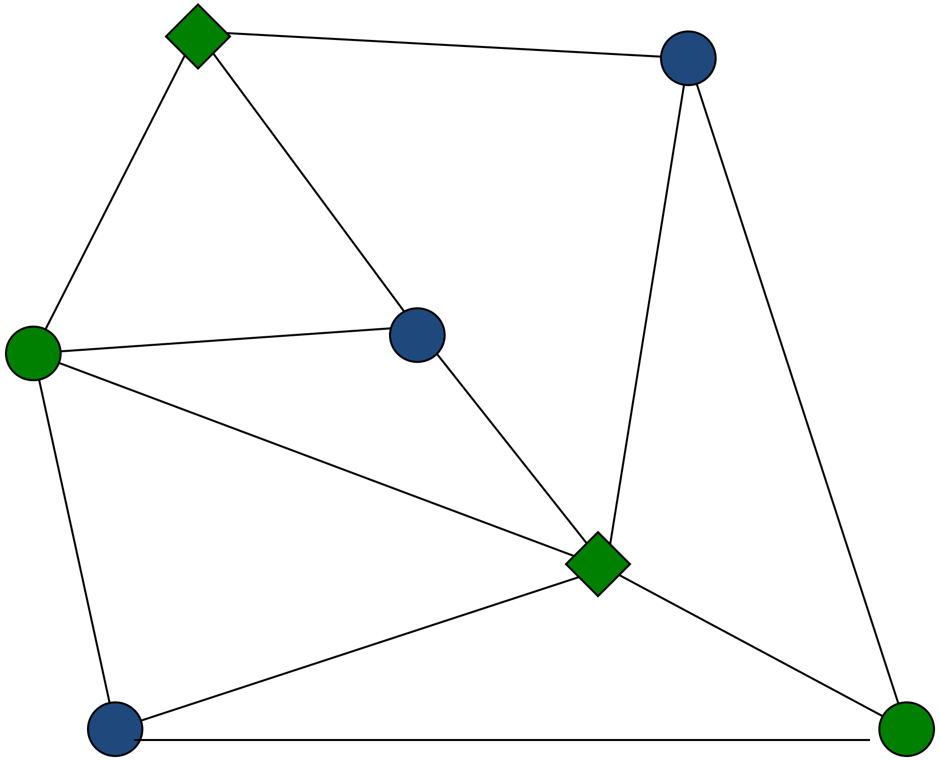
Zero-knowledge proof of 3-colorability

Prover

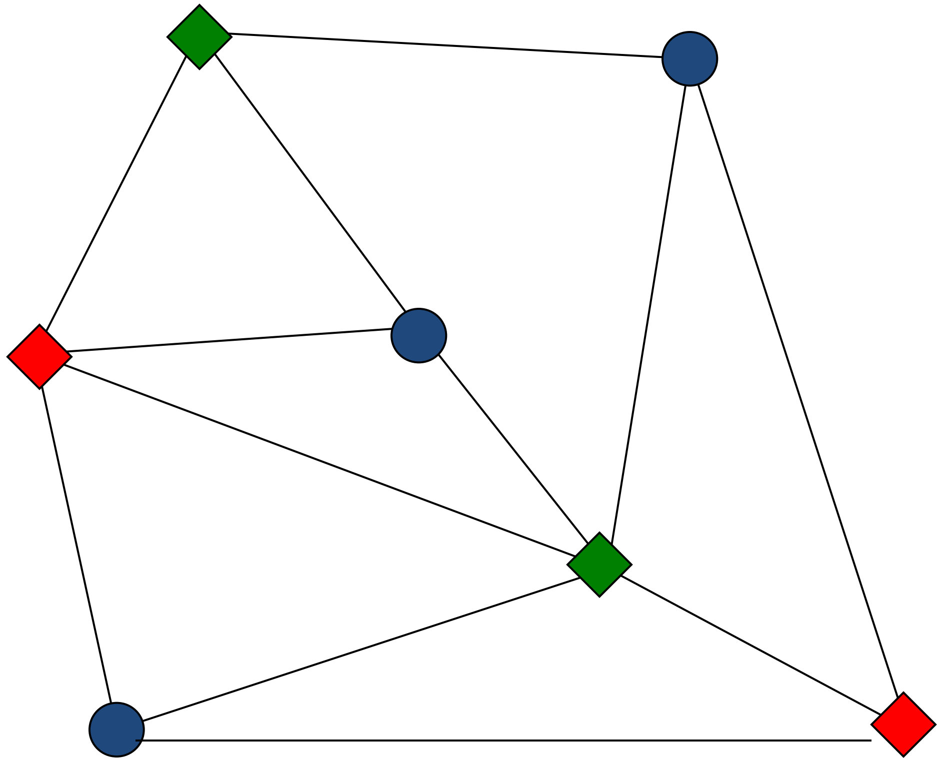


Zero-knowledge proof of 3-colorability


Prover

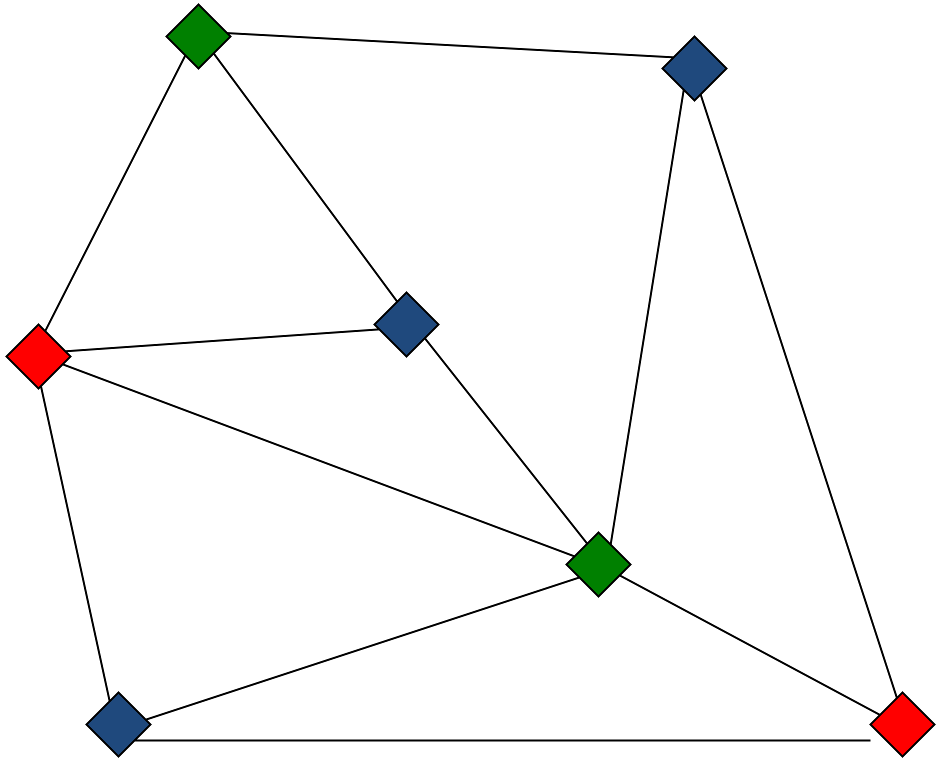


Zero-knowledge proof of 3-colorability

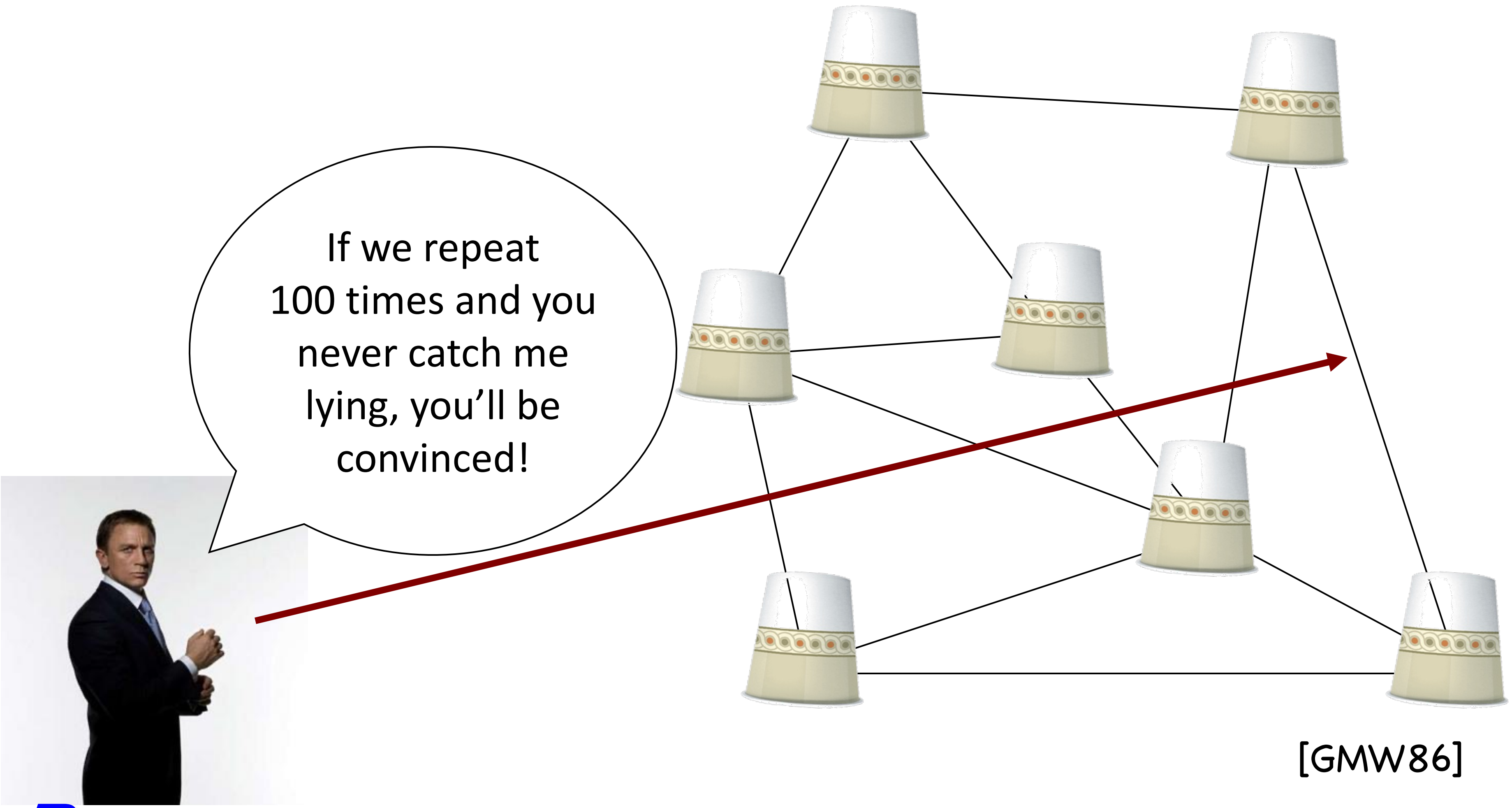


Prover

Zero-knowledge proof of 3-colorability



Zero-knowledge proof of 3-colorability



Prover

[GMW86]

Do you need paper cups?

NO. In the actual protocol, the prover “commits” to the colors.

For example, let f be a OWP, and let B be its hardcore bit.

To color the vertex v with the color(v) $\in$ {00, 01, 10}, pick random x_1 and x2 from $\{0,1\}^\lambda$
compute $y_1 = f(x_1)$, $y_2=f(x_2)$, mask = $B(x_1)B(x_2)$, masked_color=mask $\oplus$ color(v)

Instead of coloring v and covering it with a paper cup, announce “my commitment to the color of v corresponds to the unmasking of masked_color for y_1, y_2”

To ``open,” reveal x_1, x_2. The verifier (1) checks that $y_1 = f(x_1)$, $y_2=f(x_2)$ and
(2) sets color(v) = masked_color $\oplus$ $B(x_1)B(x_2)$

This commitment hides the color (because B is a hardcore bit), but the prover cannot change his mind about it.

Zero-Knowledge Proof: More Formally

- First, recall what a “language” L in NP looks like:

  L = {x | ∃ witness w such that WitnessVerification(x,w) = Accept}

  - For example:

    3-Colorability = {graph G | ∃ a way to color vertices of G into three colors so that for each (u,v) in E(G), color(u) different from color(v) }

- Let (A,B) be a pair of interactive algorithms. Notation: let Output(A(x)<->B(x)) denote the output of A(x) after interacting with B(y).

- A pair of algorithms (Prover, Verifier) constitute a zero-knowledge proof system for a language L if:
  - Running time: Verifier is a probabilistic polynomial-time algorithm. (Often we also need Prover to be ppt)
  - Completeness: if $x \in L$ and w is the ``witness” to that, then Output(Verifier(1^λ,x) <->Prover(1^λ,x,w)) = Accept
  - Soundness ε: if $x \notin L$, then for any adversarial Prover*, Pr[Output(Verifier(1^λ,x) <->Prover(1^λ,x,w)) = Accept] $\leq \varepsilon$
  - Zero knowledge: ∀ adversarial verifier V*, ∃ a ppt “simulator” algorithm SimV* such that ∀ $x \in L$, the output of SimV*(1^λ,x) is indistinguishable from Output(V*(1^λ,x) <->Prover(1^λ,x,w)).

    The meaning of this simulator: whatever the verifier V* learns from Prover(1^λ,x,w), it can learn by just running SimV*(1^λ,x) without any access to the Prover at all.

    Why does this even make sense? The simulator can see “inside” the verifier, reset it to a previous state, etc.

Zero-Knowledge Proof: More Formally

- A pair of algorithms (Prover, Verifier) constitute a zero-knowledge proof system for a language L if:
  - Running time: Verifier is a probabilistic polynomial-time algorithm. (Often we also need Prover to be ppt)
  - Completeness: if $x \in L$ and w is the ``witness" to that, then Output(Verifier(1^λ,x) <->Prover(1^λ,x,w)) = Accept
  - Soundness ε: if $x \notin L$, then for any adversarial Prover*, Pr[Output(Verifier(1^λ,x) <->Prover(1^λ,x,w)) = Accept] $\leq \varepsilon$
  - Zero knowledge: $\forall$ adversarial verifier V*, $\exists$ a ppt "simulator" algorithm SimV* such that $\forall$ $x \in L$, the output of SimV*(1^λ,x) is indistinguishable from Output(V*(1^λ,x) <->Prover(1^λ,x,w)).

    The meaning of this simulator: whatever the verifier V* learns from Prover(1^λ,x,w), it can learn by just running SimV*(1^λ,x) without any access to the Prover at all.

    Why does this even make sense? The simulator can see "inside" the verifier, reset it to a previous state, etc.

- Theorem: the protocol we just saw for 3-colorability is a ZK proof system
  - Running time: yes
  - Completeness: yes
  - Soundness: already argued
  - ZK property: need to come up with a simulator

Zero-Knowledge Proof: More Formally

- Theorem: the protocol we just saw for 3-colorability is a ZK proof system
  - Running time: yes
  - Completeness: yes
  - Soundness: already argued
  - ZK property: need to come up with a simulator

- Simulator:

  (1) guess which edge e = (u,v) the verifier will check
  (2) pick two random distinct colors (e.g. "red" and "green") and color u and v in those
  (3) color all the other vertices "red"
  (4) commit to all this coloring of the graph, send the commitments to V*
  (5) V* responds with an edge e*.

  If $e^* \neq e$:

  reset V* to its state before it received the commitments, and go back to step (1)

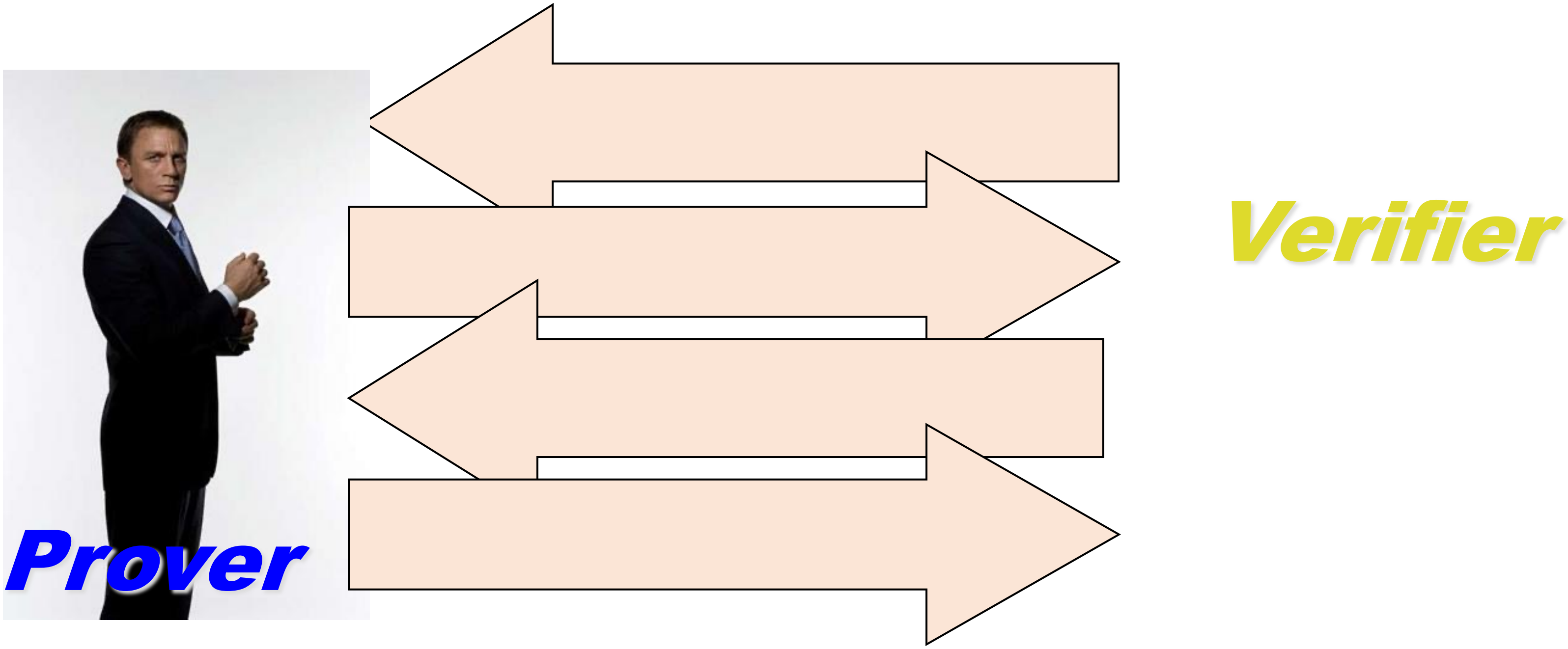
  Else: open the commitments to the colors picked in (2)

  (6) output whatever V* outputs

ZK Proofs for Other Things

Theorem: Everything provable is provable in zero-knowledge.
[GMR85,GMW86,BGGHKMR88]

(Easy to see that any L ∈ NP has a ZK proof system, because 3-colorability is NP-complete.)



- Prover convinces Verifier that the statement is true
- Verifier learned nothing about the solution

Non-Interactive ZK Proof System (NIZK) [BDMP FLS]

Note: this definition of ZK is a simplification. Many additional subtleties that we won't get into.
Important: soundness must still hold even when A has seen a simulated proof

- Setup(1^λ), Prove(params,x,w), Verify(params,x,π) : non-interactive algorithms
- Completeness: if x $\in$ L, w is a witness, params <- Setup(1^λ), π <- Prove(params,x,w),
  Verify(params,x,π) accepts
- Soundness:
  for all ppt A, Pr[params <- Setup(1^λ); (x,π) <- A(params) : x $\notin$ L and Verify(params,x,π) = Accept] = negligible(λ)
- Zero knowledge: there exists simulator algorithms SimSetup(1^λ) and SimProve(simparams,td,x)
  such that the following experiments' outputs are indistinguishable for all x $\in$ L, its witness w:

Real proof: { params <- Setup(1^λ); π <- Prove(params,x,w) : (params,π) }
Simulation: {(simparams,td) <- SimSetup(1^λ); π <- SimProve(simparams,td,x) : (simparams,π) }

Steps of the experiment

Output of the experiment

I haven't yet told you what trapdoor permutations are

- Theorem [FLS]: If trapdoor permutations exist, then NIZK proof systems exist.

More on NIZKs

- I haven't shown you how they work. And I don't have time to do so. ☹
- There is a lot of research, discussion, excitement around NIZK right now.
- There are efficient and provably secure NIZKs for languages that are interesting and important in practice (we will talk about them on Friday).

- And now we will see how they help us achieve public-key encryption.

Public-Key Encryption: Algorithms and Correctness

- KeyGen(1^λ) outputs two keys: public key PK and secret key SK
- Encrypt(PK,m) only needs the public key to output a ciphertext c
- Decrypt(SK,c) outputs the message m

- Correctness: for all m, if (PK,SK) <- KeyGen (1^λ) and c <- Encrypt(PK,m), then Decrypt(SK,c) = m.

Public-Key Encryption: Security

Recall the symmetric-key case:

- How does the adversary interact with other system participants?

black boxes/oracles for encryption and decryption
чорні скриньки/оракули
для шифрування та дешифрування

Encrypt(K, 1^λ, ▯)

Decrypt(K, 1^λ, ▯)

Recall the symmetric-key case:

- How does the adversary interact with other system participants?

Only need the decryption oracle:
A can encrypt by itself

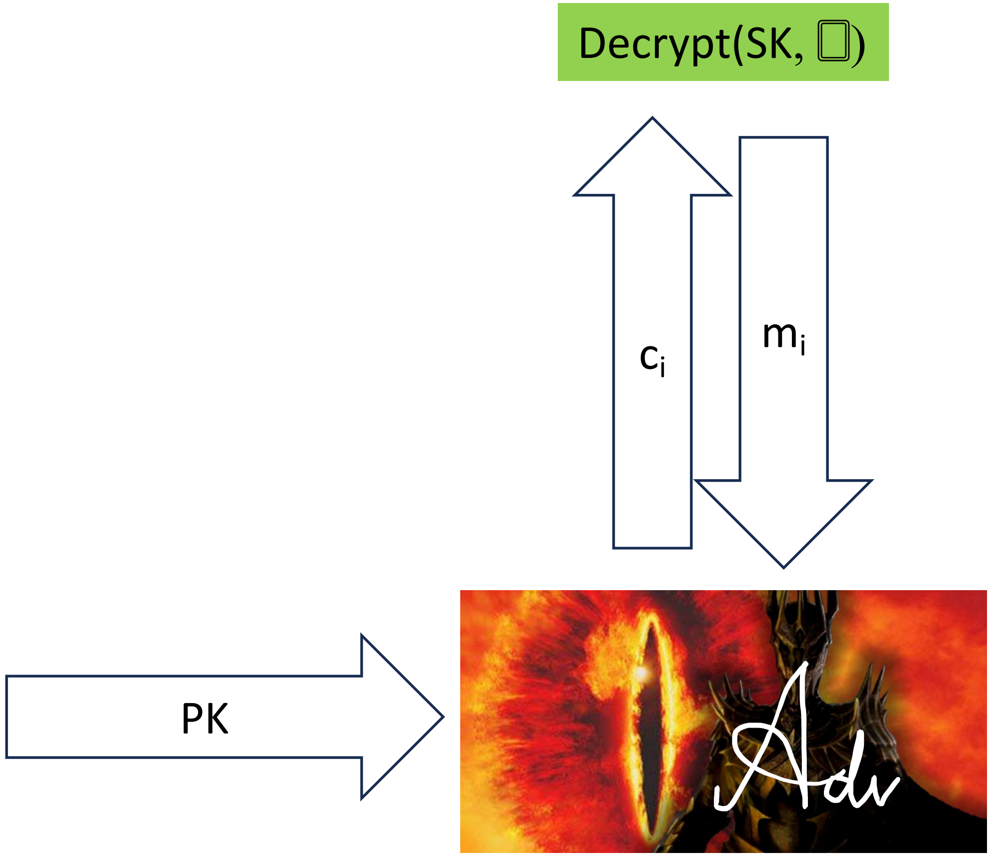
~~Encrypt(K, 1^λ, □)~~

Decrypt(K, 1^λ, □)

The Adversary receives PK as input and has access the decryption oracle:

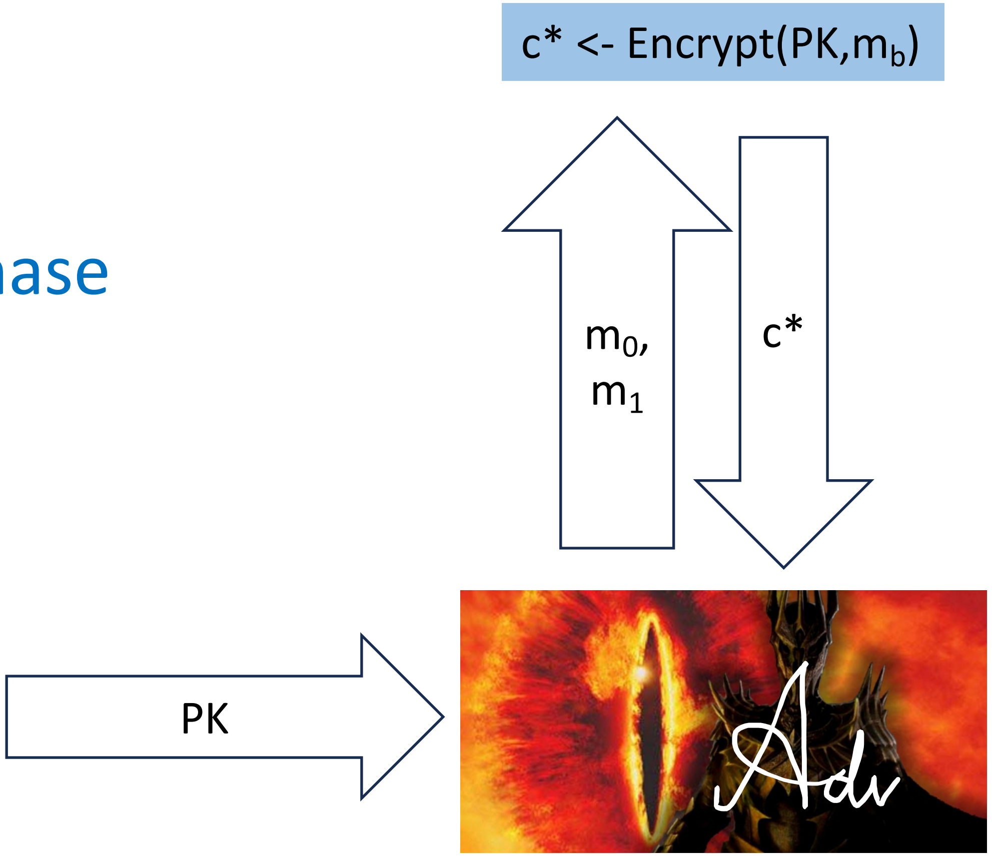
Query phase

Фаза запиту



Then the Adversary receives a challenge ciphertext

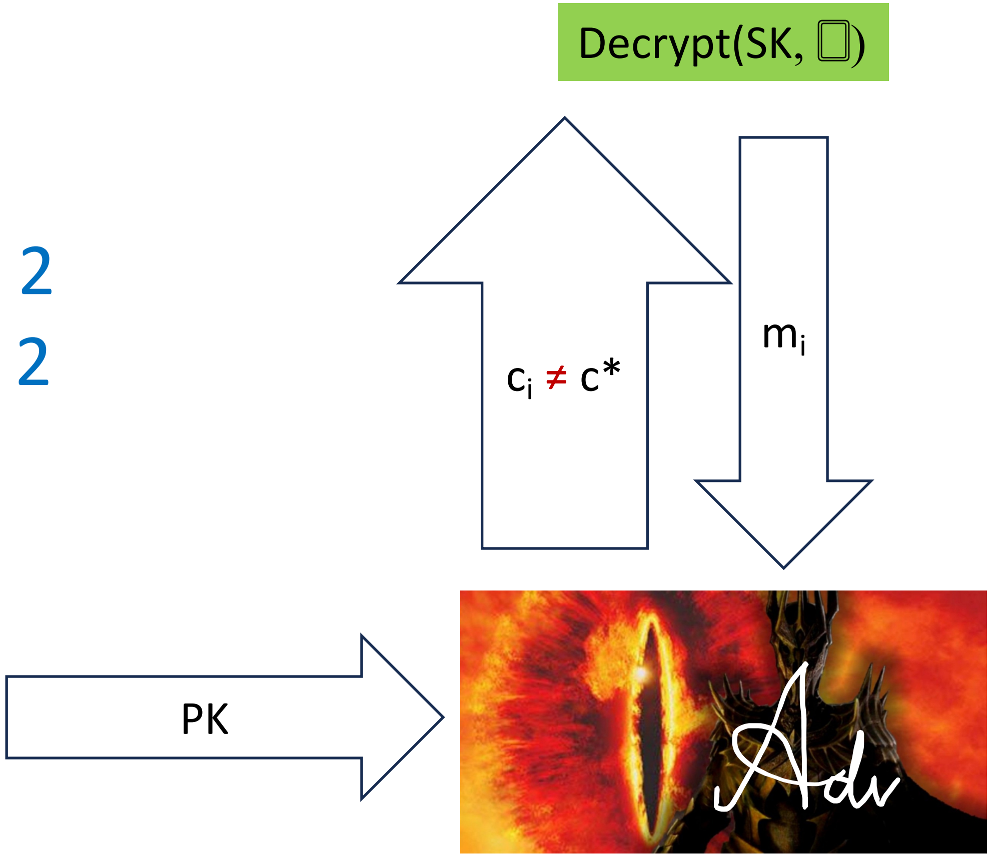
Challenge phase



The Adversary queries the decryption oracle again:

Query phase 2

Фаза запиту 2



The Adversary produces an output:

Decrypt(SK, □)

Output phase

Just as in the symmetric case:

- Let

  p_0 = Pr[A outputs 0 when b=0]
  p_1 = Pr[A outputs 0 when b=1]

(KeyGen, Encrypt, Decrypt) constitute a secure public-key encryption scheme if $|p_0 - p_1|$ = negligible(λ)

Public-Key Encryption: Construction, Try1

That's what a trapdoor permutation is! For example, RSA.

- KeyGen(1^λ) outputs PK = one-way permutation f with hardcore bit B
  SK = trapdoor, i.e. an efficient way to compute f^{-1}
- Encrypt(PK,m) for the simplified case where m is just one bit:
  pick a random x <- Domain(f), let c = $(f(x), B(x) \oplus m)$
- Decrypt(SK,c) : let c = (y, masked_message)
  recover x = $f^{-1}(y)$, recover m = masked_message $\oplus$ B(x)

- Correctness: easy to see.

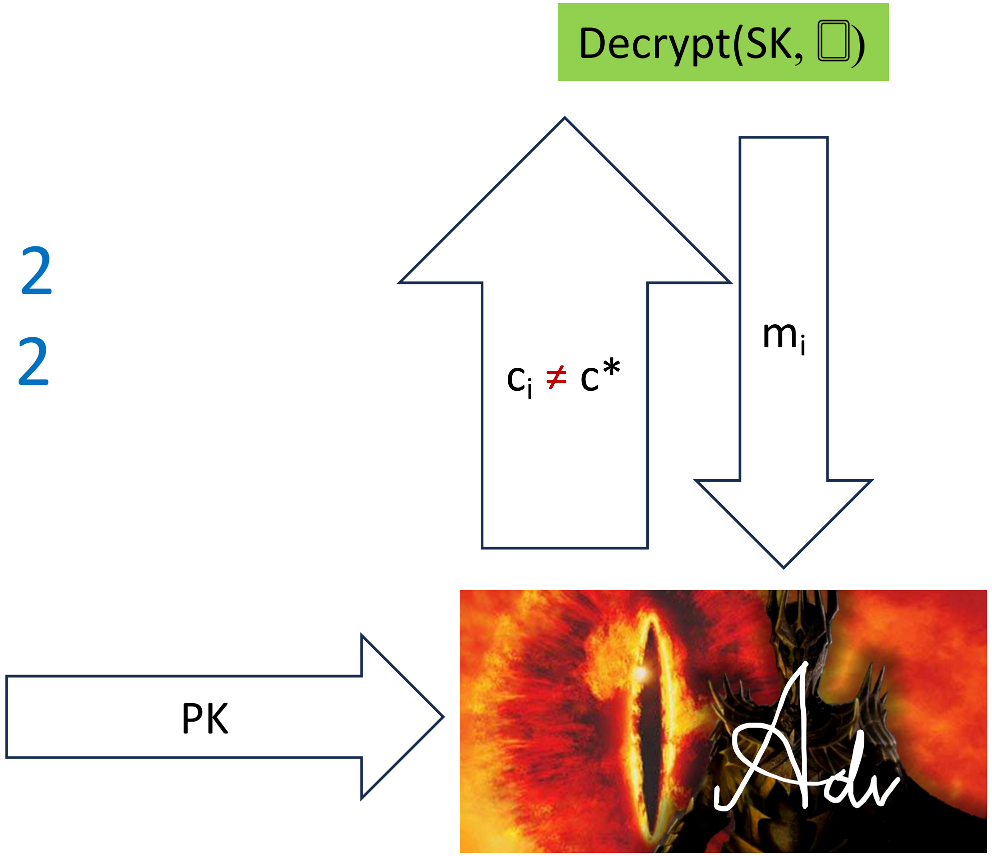
Public-Key Encryption: Construction, Try1

- Is it secure?
- If A does not have access to the decryption oracle, then it is secure (follows from the security of the trapdoor permutation)
- What if A has access to the decryption oracle?

Public-Key Encryption: Attack on Try1

Query phase 2

Фаза запиту 2



Let $c^* = (y^*, u^*)$

Form query $c = (y^*, 1 \oplus u^*)$,

receive decryption m.

Output $m^* = m \oplus 1$

Public-Key Encryption: Fix Using NIZK

- KeyGen(1^λ) outputs PK = (params,f_1,f_2), where params are for NIZK, and
  f_1, f_2 are OWPs with hardcore bit B
  SK = trapdoor for f_1

- Encrypt(PK,m) for the simplified case where m is just one bit:
  pick a random x_1 <- Domain(f_1), let $c_1 = (f_1(x_1), B(x_1) \oplus m) = (y_1, u_1)$
  pick a random x_2 <- Domain(f_2), let $c_2 = (f_2(x_2), B(x_2) \oplus m) = (y_2, u_2)$
  compute NIZK proof π that c_1 and c_2 were computed from same m
  output ciphertext $c = (c_1, c_2, \pi)$

- Decrypt(SK,c) : let $c = (c_1, c_2, \pi)$.
  Verify the proof π, reject if if doesn't verify.
  Else let $c_1 = (y_1, u_1)$. Recover $x_1 = f^{-1}(y_1)$, recover $m = u_1 \oplus B(x_1)$

- Correctness: easy to see.

Proof of Security

- Roadmap for the proof:
  - Define games that are different from the security experiments
  - Show that all the games are indistinguishable

Proof of Security

- Game 1: Security experiment when m = 0.

Challenger uses f_1^{-1} in decryption queries
Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus m)$, $c_2 = (f_2(x_2), B(x_2) \oplus m)$,
and proof π

Proof of Security

- Game 1: Security experiment when m = 0.

  Challenger uses f_1^{-1} in decryption queries
  Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 0)$, $c_2 = (f_2(x_2), B(x_2) \oplus 0)$,
  and proof π

Proof of Security

- Game 2: Security experiment with params output by SimSetup, m=0

  Challenger uses f_1^{-1} in decryption queries
  Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 0)$, $c_2 = (f_2(x_2), B(x_2) \oplus 0)$, and SIMULATED proof π

Indistinguishable from Game 1 because of the security of NIZK

Proof of Security

- Game 3: Security experiment with params output by SimSetup and a mismatched challenge ciphertext

Challenger uses f_1^{-1} in decryption queries
Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 0)$, $c_2 = (f_2(x_2), B(x_2) \oplus 1)$, and SIMULATED proof π

Indistinguishable from Game 2 because of the security of OWP and its hardcore bit B

Proof of Security

- Game 4: Security experiment with params output by SimSetup and a mismatched challenge ciphertext

Challenger uses f_2^{-1} in decryption queries
Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 0)$, $c_2 = (f_2(x_2), B(x_2) \oplus 1)$, and SIMULATED proof π

Indistinguishable from Game 3 because of the soundness of NIZK even after seeing a simulated proof.

Proof of Security

- Game 5: Security experiment with params output by SimSetup and m=1

  Challenger uses f_2^{-1} in decryption queries
  Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 1)$, $c_2 = (f_2(x_2), B(x_2) \oplus 1)$,
  and SIMULATED proof π

Indistinguishable from Game 4 because of the security of OWP and its hardcore bit B

Proof of Security

- Game 6: Security experiment with params output by Setup and m=1

  Challenger uses f_2^{-1} in decryption queries
  Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 1)$, $c_2 = (f_2(x_2), B(x_2) \oplus 1)$, and proof π

  Indistinguishable from Game 5 because of the zero-knowledge property of NIZK

Proof of Security

- Game 7: Security experiment with params output by Setup and m=1

  Challenger uses f_1^{-1} in decryption queries
  Challenge ciphertext is $c_1 = (f_1(x_1), B(x_1) \oplus 1)$, $c_2 = (f_2(x_2), B(x_2) \oplus 1)$, and proof π

Indistinguishable from Game 6 because of the soundness property of NIZK

Today: Cryptomania

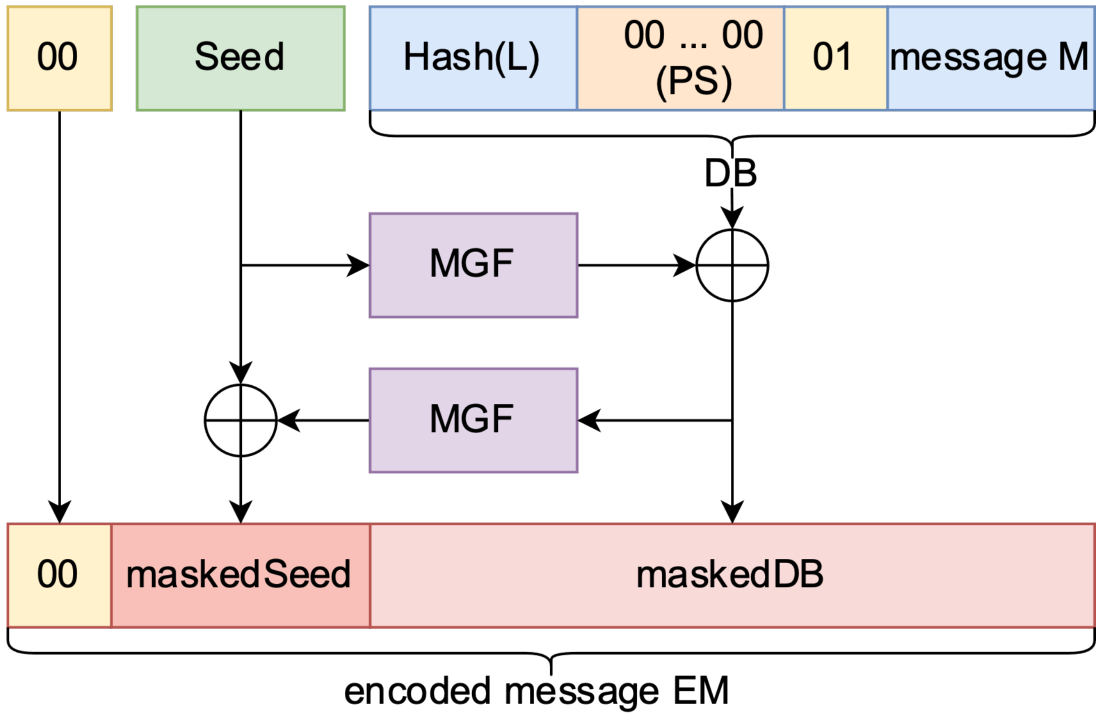
- Zero-knowledge proofs
  - Definition (high level)
  - Construction for an NP-complete language
  - Another flavor: non-interactive zero-knowledge proof (NIZK)
- Public-key encryption: definition
- Trapdoor permutation (aka OWP with a trapdoor)
  - Definition
  - Examples
- Construct public-key encryption from NIZK and TDPs
  - Very theoretical construction, don’t use it in practice!
- Look at practical constructions and try to make sense of them using our theoretical tools

Why did the two TDPs and NIZK help?

- Intuition: that way, in order to form a ciphertext, you “had to know” the message.

This helps us make sense of public-key encryption that is used in practice, RSA-OAEP

- (picture from Wikipedia)
- In RSA-OAEP:
  the public key is the RSA TDP f
  to encrypt message M, you encode it as shown in the picture, then output c = f(EM)



Today: Cryptomania

- Zero-knowledge proofs
  - Definition (high level)
  - Construction for an NP-complete language
  - Another flavor: non-interactive zero-knowledge proof (NIZK)
- Public-key encryption: definition
- Trapdoor permutation (aka OWP with a trapdoor)
  - Definition
  - Examples
- Construct public-key encryption from NIZK and TDPs
  - Very theoretical construction, don’t use it in practice!
- Look at practical constructions and try to make sense of them using our theoretical tools

Problems for Thursday’s problem-solving session with Illia

The definition of security for public-key encryption that we saw in today’s lecture is called “semantic security against adaptive chosen-ciphertext attack (CCA).” Sometimes it’s called CCA2-security, because there are two decryption query phases. If we change the security game so that the adversary is not able to issue decryption queries, then we get a weaker notion of security, called “semantic security.”

(1) Prove that our Try1 cryptosystem (slide 57) is secure if f is a trapdoor permutation with hardcore bit B.

(2) Give a semantically secure cryptosystem that allows one to encrypt messages that are longer than one bit. Prove security of your construction.