Cryptography

Ron Reaser Chris Wilson

Lane Department of Computer Science and Electrical Engineering
West Virginia University

April 28, 2009

1. Theory
   - Purpose and Examples
   - Public-Key Cryptosystems
   - One-Way Functions
   - Randomization

2. Applications
   - Protocols
   - Signed Messages
   - Mental Poker
   - Interactive Proofs
   - Zero-Knowledge Proofs

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

Definition

Encryption is a method used by two parties to ensure secrecy if their communications are intercepted by a malevolent third party.

The parties agree on two polynomial time algorithms E and D assumed to be known to the general public. Some values e and d are chosen. These are called keys. The sender computes the encrypted message y, called ciphertext, as $E(e, x) = y$ for some original message x. The receiver then computes the decrypted message, called plaintext, as $D(d, y) = x$. The keys are selected such that D is the inverse of E. There should be no way of computing x from y easily without knowing d.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

One-Time Pad

One can choose both d and e to be the same arbitrary string e of length x.

Let both E and D merely be the *exclusive-or* operator. That is, $E(e, x) = e \oplus x$ and $D(e, y) = e \oplus y$. No eavesdropper can deduce x from y without first knowing e.

Observation

The key and message length must be exactly the same for the *exclusive-or* to work. Further, the key must be agreed upon and transmitted securely. Both of these problems make the use of a one-time pad undesirable, especially for frequent communication.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Modern cryptography relies on the separation of encode and decode keys. The encoding and decoding functions are still E and D respectively, but the decoding key is now d and is distinct from the encoding key e, so that $D(d, E(e, x)) = x$. An (e, d) pair is generated by an individual who makes e known publicly but keeps d private. The point is that it is intentionally infeasible to deduce d even knowing e, making it computationally hard to recover x from y without knowing d.

This is called a public-key cryptosystem.

Observation

The safety of a public-key cryptosystem lies in the difficulty of guessing x given y. So there can be no impossibility proof as there was for the one-time pad. Checking all x until $E(e, x) = y$ can find the original message. Since x cannot be more than polynomially longer than y, this problem is in **FNP**.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Definition

Public-key cryptosystems are secure only if **P** $\neq$ **NP**. Even so, they are not guaranteed to be secure without utilizing a special function called a one-way function and described as follows.

(i) The function f is one-to-one, so $x = f^{-1}(f(x))$ for any input x.

(ii) The value $f(x)$ is at most polynomially longer or shorter than x.

(iii) The function f is in **FP**, so it can be computed in polynomial time. This constitutes encoding, which is necessarily easy.

(iv) The inverse function f^{-1} is in **FNP**, so it cannot be computed in polynomial time. This constitutes decoding, which is intentionally difficult.

There is currently no guarantee that such functions exist even if **P** $\neq$ **NP**, but they are believed to exist as will be shown in the following examples.

Integer Multiplication of Primes

One function that many suspect is indeed a one-way function is the integer multiplication of primes.

Supposed that $p < q$ are prime numbers and $C(p)$ and $C(q)$ are certificates of their primality. The function $f_{MULT}(p, C(p), q, C(q)) = p \cdot q$ is indeed one-to-one and computable in polynomial time. Nevertheless there is no known polynomial time algorithm to compute ${f_{MULT}}^{-1}$ and thereby factor the product of two large primes.

Integer Multiplication of Primes

One function that many suspect is indeed a one-way function is the integer multiplication of primes.

Supposed that $p < q$ are prime numbers and $C(p)$ and $C(q)$ are certificates of their primality. The function $f_{MULT}(p, C(p), q, C(q)) = p \cdot q$ is indeed one-to-one and computable in polynomial time. Nevertheless there is no known polynomial time algorithm to compute ${f_{MULT}}^{-1}$ and thereby factor the product of two large primes.

Integer Multiplication of Primes

One function that many suspect is indeed a one-way function is the integer multiplication of primes.

Supposed that $p < q$ are prime numbers and $C(p)$ and $C(q)$ are certificates of their primality. The function $f_{MULT}(p, C(p), q, C(q)) = p \cdot q$ is indeed one-to-one and computable in polynomial time. Nevertheless there is no known polynomial time algorithm to compute ${f_{MULT}}^{-1}$ and thereby factor the product of two large primes.

Integer Multiplication of Primes

One function that many suspect is indeed a one-way function is the integer multiplication of primes.

Supposed that $p < q$ are prime numbers and $C(p)$ and $C(q)$ are certificates of their primality. The function $f_{MULT}(p, C(p), q, C(q)) = p \cdot q$ is indeed one-to-one and computable in polynomial time. Nevertheless there is no known polynomial time algorithm to compute ${f_{MULT}}^{-1}$ and thereby factor the product of two large primes.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in **P** and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in **P** and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in $\mathbf{P}$ and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in **P** and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in **P** and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Exponentiation Modulo a Prime

The function f_{EXP}, exponentiation modulo a prime, takes as parameters a prime p with a certificate $C(p)$ guaranteeing primality, a primitive root r modulo p that is part of $C(p)$, and an arbitrary integer $x < p$.

By definition, $f_{EXP}(p, C(p), r, x) = (p, C(p), r^x \bmod p)$, which can be evaluated in polynomial time. Determining x given $(r, p, r^x \bmod p)$ is known as the discrete logarithm problem, and there is no known polynomial time algorithm to solve it.

Since f_{EXP} is in **P** and ${f_{EXP}}^{-1}$ is not known to be, f_{EXP} is believed to be one-way.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

Assertion

The following function, RSA, is one-way. It can be computed in polynomial time, yet no polynomial algorithm for inverting it has ever been announced.

RSA

Combining f_{MULT} and f_{EXP} as a public-key cryptosystem is called RSA. This stands for Rivest, Shamir, and Adleman, the scientists who devised the function at MIT in 1977.

Let p and q be two prime numbers. The number of bits in their product pq is $n = \lceil \ln pq \rceil$. All numbers modulo pq will be considered as n-bit strings over $\{0, 1\}$.

Let $\phi(pq)$ be $pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq - p - q + 1$. Assume that the key d is just a number that is relatively prime to $\phi(pq)$. The RSA function is defined to be $f_{RSA}(x, e, p, C(p), q, C(q)) = (x^e \mod pq, pq, e)$.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

RSA (Continued)

Bob knows p and q, and he shares with Alice their product pq. He also shares a publicly-known integer e that is relatively prime to $\phi(pq)$.

Alice uses the public key to encrypt the message x, an n-bit integer, as follows: $y = x^e \mod pq$. Bob alone knows an integer d such that $ed = 1 + k\phi(pq)$ for some integer k.

Bob can decrypt y as follows: $y^d = x^{ed} = x^{1+k\phi(pq)} = x \mod pq$.

So the encryption key is actually (pq, e) and the decryption key (pq, d).

Observation

All attempts to "break" RSA will be at least as hard as inverting f_{MULT}.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Observation

Sufficiently short messages may be easy to decode with a regular public-key cryptosystem. Specifically, a message composed of one bit, either 0 or 1, would be very easy to decode because the always one-to-one mapping can be exploited.

Solution

This weakness can be patched for single-bit or other short messages by padding the unencoded message with randomized junk bits. For example, instead of encoding a bit x, encode a message $2k + x$ where k is a random integer. Because $2k$ is even, the last bit in the sum will be the x bit. However, because there are as many possible one-to-one mappings for this k-padded x bit as there are choices for k, the encoded message is much harder to decrypt without knowing the private key.

Application

Rather than using a plain public-key cryptosystem, one can break a message into many parts, padding each part with a random bit string, and send each encoded alone. Although this slows down both the encoding and decoding, the decoding is still fast enough for the intended recipient while even harder for an assailant. This is called a randomized public-key cryptosystem.

Concept

Compared to previous studies in computational complexity, encryption is a significant departure. In most of the problems studied so far, there has been one agent (the algorithm) and one problem and a goal of being as efficient as possible. With cryptosystems, there are often multiple agents in the system, and it is the goal to make certain tasks easy for some agents and difficult for others. Steps to achieve this constitute a protocol.

Concept

Compared to previous studies in computational complexity, encryption is a significant departure. In most of the problems studied so far, there has been one agent (the algorithm) and one problem and a goal of being as efficient as possible. With cryptosystems, there are often multiple agents in the system, and it is the goal to make certain tasks easy for some agents and difficult for others. Steps to achieve this constitute a protocol.

Concept

Compared to previous studies in computational complexity, encryption is a significant departure. In most of the problems studied so far, there has been one agent (the algorithm) and one problem and a goal of being as efficient as possible. With cryptosystems, there are often multiple agents in the system, and it is the goal to make certain tasks easy for some agents and difficult for others. Steps to achieve this constitute a protocol.

Concept

Compared to previous studies in computational complexity, encryption is a significant departure. In most of the problems studied so far, there has been one agent (the algorithm) and one problem and a goal of being as efficient as possible. With cryptosystems, there are often multiple agents in the system, and it is the goal to make certain tasks easy for some agents and difficult for others. Steps to achieve this constitute a protocol.

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Definition

Suppose Alice wants to send Bob a message and ensure that he will know it was truly sent by her. Such a message $S_{Alice}(x)$ should contain the original x plus something which unmistakably identifies the sender. Such a process is called signing a message and can be achieved using a public-key cryptosystem.

Algorithm

Suppose Alice has the key pair (e_{Alice}, d_{Alice}) and Bob (e_{Bob}, d_{Bob}) and that the encoding and decoding functions E and D are the same for both Alice and Bob.

Alice starts by creating a signed message $S_{Alice}(x) = (x, D(d_{Alice}, x))$. That is, she sends Bob an original message x as well as its decryption (rather than encryption) using her private key. (The entire signed message can then be encrypted using Bob's public key to keep it secret, but this step is outside the scope of message signing.)

Algorithm (Continued)

When Bob receives the message, he encodes (rather than decodes) the decrypted part using Alice's public key. That is, he finds $E(e_{Alice}, D(d_{Alice}, x))$. This is the same thing as $D(d_{Alice}, E(e_{Alice}, x))$, because encoding and decoding are inverse functions. This inversion yields x. So given the message $(x, D(d_{Alice}, x))$ Bob is able to find (x, x).

This confirms that the sender had Alice's private key. Bob can plausibly believe she sent the message. This signing approach is only secure if the private keys are.

Algorithm (Continued)

When Bob receives the message, he encodes (rather than decodes) the decrypted part using Alice's public key. That is, he finds $E(e_{Alice}, D(d_{Alice}, x))$. This is the same thing as $D(d_{Alice}, E(e_{Alice}, x))$, because encoding and decoding are inverse functions. This inversion yields x. So given the message $(x, D(d_{Alice}, x))$ Bob is able to find (x, x).

This confirms that the sender had Alice's private key. Bob can plausibly believe she sent the message. This signing approach is only secure if the private keys are.

Algorithm (Continued)

When Bob receives the message, he encodes (rather than decodes) the decrypted part using Alice's public key. That is, he finds $E(e_{Alice}, D(d_{Alice}, x))$. This is the same thing as $D(d_{Alice}, E(e_{Alice}, x))$, because encoding and decoding are inverse functions. This inversion yields x. So given the message $(x, D(d_{Alice}, x))$ Bob is able to find (x, x).

This confirms that the sender had Alice's private key. Bob can plausibly believe she sent the message. This signing approach is only secure if the private keys are.

Algorithm (Continued)

When Bob receives the message, he encodes (rather than decodes) the decrypted part using Alice's public key. That is, he finds $E(e_{Alice}, D(d_{Alice}, x))$. This is the same thing as $D(d_{Alice}, E(e_{Alice}, x))$, because encoding and decoding are inverse functions. This inversion yields x. So given the message $(x, D(d_{Alice}, x))$ Bob is able to find (x, x).

This confirms that the sender had Alice's private key. Bob can plausibly believe she sent the message. This signing approach is only secure if the private keys are.

Algorithm (Continued)

When Bob receives the message, he encodes (rather than decodes) the decrypted part using Alice's public key. That is, he finds $E(e_{Alice}, D(d_{Alice}, x))$. This is the same thing as $D(d_{Alice}, E(e_{Alice}, x))$, because encoding and decoding are inverse functions. This inversion yields x. So given the message $(x, D(d_{Alice}, x))$ Bob is able to find (x, x).

This confirms that the sender had Alice's private key. Bob can plausibly believe she sent the message. This signing approach is only secure if the private keys are.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Mental Poker

Mental poker refers to a set of cryptographic problems that concern playing a fair game over distance without the need for a trusted third party. The term is also applied to the theories surrounding these problems and their possible solutions. The name stems from the card game poker, one of the games to which this kind of problem applies. A similar problem is flipping a coin over a distance.

The problem can be described as follows: "How can one allow only authorized agents to have access to certain information while not using a trusted arbiter?" Note that eliminating the trusted third-party avoids the problem of trying to determine whether the third party can be trusted or not. This may also reduce the resources required.

Mental poker requires a commutative algorithm, an algorithm where the order of encryption does not matter.

Example

An algorithm for shuffling cards using commutative encryption would be as follows.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.
(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.
(3) Alice shuffles the cards.
(4) Alice passes the deck to Bob.
(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.
(6) Bob shuffles the deck.
(7) Bob passes the deck back to Alice.
(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.
(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.
(10) Alice passes the deck to Bob.
(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.
(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.
(13) Bob passes the deck back to Alice.
(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.
(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.
(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.
(3) Alice shuffles the cards.
(4) Alice passes the deck to Bob.
(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.
(6) Bob shuffles the deck.
(7) Bob passes the deck back to Alice.
(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.
(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.
(10) Alice passes the deck to Bob.
(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.
(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.
(13) Bob passes the deck back to Alice.
(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.
(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Shuffling Cards

(1) Alice and Bob agree on a "deck" of cards. This abstracts an arbitrary data set.

(2) Alice picks an encryption key a and uses this to encrypt each card of the deck.

(3) Alice shuffles the cards.

(4) Alice passes the deck to Bob.

(5) Bob picks an encryption key b and uses this to again encrypt each card of the deck.

(6) Bob shuffles the deck.

(7) Bob passes the deck back to Alice.

(8) Alice decrypts each card using her key a, leaving only Bob's encryption in place.

(9) Alice picks an encryption key a_i for each i-numbered card and encrypts them individually.

(10) Alice passes the deck to Bob.

(11) Bob decrypts each card using his key b, leaving only Alice's encryption in place.

(12) Bob picks an encryption key e_j for each j-numbered card and encrypts them individually.

(13) Bob passes the deck back to Alice.

(14) Alice repeats steps 2 to 4 and 8 to 10 with each additional player besides Bob.

(15) Alice publishes the deck for everyone playing.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Concept

Suppose Eric has exponential computing powers while Paula has strictly polynomial computing powers. Eric can decide some $\textsc{Problem}$ in **NP** on any input. He wants to prove $\textsc{Problem}(x)$ to Paula given that they both know some input x.

If $\textsc{Problem}(x) =$ "yes," Eric can find in exponential time a "yes" certificate to show Paula such that she can verify it in polynomial time. If however $\textsc{Problem}(x) =$ "no," Eric cannot prove this to Paula because he cannot provide a "yes" certificate. She would have to reject the input herself, and that would require exponential time.

However, utilizing randomization and a series of alternating messages, Eric can make Paula probabilistically sure that $\textsc{Problem}(x) =$ "no" if it does in fact reject without violating her time constraints. This is called an interactive proof.

Example

This is most intuitively explained through an example, although the protocol can be generalized quite clearly for other problems in NP.

Graph Isomorphism

The problem of Graph Isomorphism is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide Graph Isomorphism. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of Graph Isomorphism is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide Graph Isomorphism. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of Graph Isomorphism is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide Graph Isomorphism. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of Graph Isomorphism is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide Graph Isomorphism. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of GRAPH ISOMORPHISM is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide GRAPH ISOMORPHISM. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Graph Isomorphism

The problem of Graph Isomorphism is one of the few **NP** problems that is not specifically known to be in **P** or **NP**-complete. Eric wants to prove to Paula the he can decide this problem. The following experiment is conducted several times honestly and accurately.

(i) Eric sends Paula a non-isomorphic graph pair (G_1, G_2).

(ii) Paula chooses either G_1 or G_2 randomly and permutes it into a new graph H. She then sends Eric the graph H as a challenge.

(iii) Eric determines which of G_1 and G_2 is isomorphic to H and sends back a response of 1 or 2 accordingly.

(iv) Paula verifies whether the response corresponds to the graph she originally chose to permute. Eric passes the challenge if and only if he was correct.

At the end, Eric will have always managed to respond correctly to Paula's challenge. After so many passed challenges, it becomes less and less likely that Eric does not actually have the ability to decide Graph Isomorphism. Paula can probabilistically trust Eric if he declares two graphs either isomorphic or non-isomorphic. Further, Paula never used more than polynomial time computations.

Definition

The class of all problems solvable with an interactive proof is called **IP**. So GRAPH ISOMORPHISM is in **IP**.

Observation

The class **NP** is just a subclass of **IP** where Paula uses no randomization. Further, the class **BPP** is a subset of **IP** where Paula ignores all of Eric's messages. Finally, it so happens that **IP** has been proven identical to **PSPACE**.

Definition

The class of all problems solvable with an interactive proof is called **IP**. So GRAPH ISOMORPHISM is in **IP**.

Observation

The class **NP** is just a subclass of **IP** where Paula uses no randomization. Further, the class **BPP** is a subset of **IP** where Paula ignores all of Eric's messages. Finally, it so happens that **IP** has been proven identical to **PSPACE**.

Definition

The class of all problems solvable with an interactive proof is called **IP**. So GRAPH ISOMORPHISM is in **IP**.

Observation

The class **NP** is just a subclass of **IP** where Paula uses no randomization. Further, the class **BPP** is a subset of **IP** where Paula ignores all of Eric's messages. Finally, it so happens that **IP** has been proven identical to **PSPACE**.

Definition

The class of all problems solvable with an interactive proof is called **IP**. So GRAPH ISOMORPHISM is in **IP**.

Observation

The class **NP** is just a subclass of **IP** where Paula uses no randomization. Further, the class **BPP** is a subset of **IP** where Paula ignores all of Eric's messages. Finally, it so happens that **IP** has been proven identical to **PSPACE**.

Definition

There is an interesting subset of interactive proofs in which the participants have the same computational power but there is an added secrecy requirement.

Suppose that Peggy wants to prove something in such a way that Victor can verify it without actually knowing how to prove it himself. Before it was merely the case that Paula was computationally incapable of using Eric's method. In this case, Peggy simply does not want Victor to know the method. A protocol to achieve this is called a zero-knowledge proof.

Examples

As before, this will be explained through an example that is easily generalized.

Definition

There is an interesting subset of interactive proofs in which the participants have the same computational power but there is an added secrecy requirement.

Suppose that Peggy wants to prove something in such a way that Victor can verify it without actually knowing how to prove it himself. Before it was merely the case that Paula was computationally incapable of using Eric's method. In this case, Peggy simply does not want Victor to know the method. A protocol to achieve this is called a zero-knowledge proof.

Examples

As before, this will be explained through an example that is easily generalized.

Definition

There is an interesting subset of interactive proofs in which the participants have the same computational power but there is an added secrecy requirement.

Suppose that Peggy wants to prove something in such a way that Victor can verify it without actually knowing how to prove it himself. Before it was merely the case that Paula was computationally incapable of using Eric's method. In this case, Peggy simply does not want Victor to know the method. A protocol to achieve this is called a zero-knowledge proof.

Examples

As before, this will be explained through an example that is easily generalized.

Definition

There is an interesting subset of interactive proofs in which the participants have the same computational power but there is an added secrecy requirement.

Suppose that Peggy wants to prove something in such a way that Victor can verify it without actually knowing how to prove it himself. Before it was merely the case that Paula was computationally incapable of using Eric's method. In this case, Peggy simply does not want Victor to know the method. A protocol to achieve this is called a zero-knowledge proof.

Examples

As before, this will be explained through an example that is easily generalized.

Definition

There is an interesting subset of interactive proofs in which the participants have the same computational power but there is an added secrecy requirement.

Suppose that Peggy wants to prove something in such a way that Victor can verify it without actually knowing how to prove it himself. Before it was merely the case that Paula was computationally incapable of using Eric's method. In this case, Peggy simply does not want Victor to know the method. A protocol to achieve this is called a zero-knowledge proof.

Examples

As before, this will be explained through an example that is easily generalized.

Hamilton Cycle

Suppose that both Peggy and Victor are familiar with a graph G and that Peggy alone knows a tour in it. For some reason, she wants to convince Victor that she knows a tour without giving it to him. Perhaps she wants to sell him the knowledge, or perhaps she is using it as her signature. Regardless, Victor is willing to entertain her for a coveted solution to an instance of this **NP**-complete problem.

Hamilton Cycle

Suppose that both Peggy and Victor are familiar with a graph G and that Peggy alone knows a tour in it. For some reason, she wants to convince Victor that she knows a tour without giving it to him. Perhaps she wants to sell him the knowledge, or perhaps she is using it as her signature. Regardless, Victor is willing to entertain her for a coveted solution to an instance of this **NP**-complete problem.

Hamilton Cycle

Suppose that both Peggy and Victor are familiar with a graph G and that Peggy alone knows a tour in it. For some reason, she wants to convince Victor that she knows a tour without giving it to him. Perhaps she wants to sell him the knowledge, or perhaps she is using it as her signature. Regardless, Victor is willing to entertain her for a coveted solution to an instance of this **NP**-complete problem.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

(a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

(b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

(a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

(b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

(a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

(b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

(a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

(b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

(a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

(b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

   (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

   (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

  (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.

  (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

- (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.
- (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

- (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.
- (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

- (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.
- (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

- (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.
- (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

As before, they undertake the following experiment honestly in alternation.

(i) Peggy privately creates a graph H isomorphic to G. For each edge in H, Peggy creates an index card describing its mapping as $(i', j') \in H \mapsto (i, j) \in G$. She puts all of these cards face-down on a table. From now on she is unable to change the H she has committed to, but Victor does not yet know its structure.

(ii) Victor flips a coin to choose one of two questions with which to challenge Peggy.

- (a) If he asks to be shown that H and G are indeed isomorphic, Peggy turns over all of the cards. Victor can then verify the isomorphism based on the node mappings, but he will not find out anything about a possible tour.
- (b) If instead Victor asks to be shown a tour in H, she turns over just the cards that are in the tour. Victor can verify that there is a tour in H based on the edges shown, but he will not know whether H and G are truly isomorphic because he is missing part of H.

(iii) Peggy burns all of the cards, or maybe if she is thoughtful she recycles them.

(iv) Victor verifies that Peggy was able to pass his challenge.

Note that the cards are an abstraction for encryption and decryption and that the coin flip is an abstraction for randomization.

Hamilton Cycle (Continued)

With each successive experiment it becomes less and less likely that Peggy always guessed which challenge Victor would choose to be able to pass it rather than being prepared for both by actually knowing the tour. After sufficiently many tests, Victor will be probabilistically certain that Peggy does indeed know a tour in G, yet he will not himself know the tour.

With k tests, his doubt $\frac{1}{2^k}$ approaches 0. With just 7 tests this is less than 1%.

If Peggy were dishonest, she could rig Victor's coin so that she would not actually have to know a tour in G to pass all of the challenges: she could just be prepared to pass whichever was chosen. It is assumed, however, that Peggy's behavior is honest.

Hamilton Cycle (Continued)

With each successive experiment it becomes less and less likely that Peggy always guessed which challenge Victor would choose to be able to pass it rather than being prepared for both by actually knowing the tour. After sufficiently many tests, Victor will be probabilistically certain that Peggy does indeed know a tour in G, yet he will not himself know the tour.

With k tests, his doubt $\frac{1}{2^k}$ approaches 0. With just 7 tests this is less than 1%.

If Peggy were dishonest, she could rig Victor's coin so that she would not actually have to know a tour in G to pass all of the challenges: she could just be prepared to pass whichever was chosen. It is assumed, however, that Peggy's behavior is honest.

Hamilton Cycle (Continued)

With each successive experiment it becomes less and less likely that Peggy always guessed which challenge Victor would choose to be able to pass it rather than being prepared for both by actually knowing the tour. After sufficiently many tests, Victor will be probabilistically certain that Peggy does indeed know a tour in G, yet he will not himself know the tour.

With k tests, his doubt $\frac{1}{2^k}$ approaches 0. With just 7 tests this is less than 1%.

If Peggy were dishonest, she could rig Victor's coin so that she would not actually have to know a tour in G to pass all of the challenges: she could just be prepared to pass whichever was chosen. It is assumed, however, that Peggy's behavior is honest.

Hamilton Cycle (Continued)

With each successive experiment it becomes less and less likely that Peggy always guessed which challenge Victor would choose to be able to pass it rather than being prepared for both by actually knowing the tour. After sufficiently many tests, Victor will be probabilistically certain that Peggy does indeed know a tour in G, yet he will not himself know the tour.

With k tests, his doubt $\frac{1}{2^k}$ approaches 0. With just 7 tests this is less than 1%.

If Peggy were dishonest, she could rig Victor's coin so that she would not actually have to know a tour in G to pass all of the challenges: she could just be prepared to pass whichever was chosen. It is assumed, however, that Peggy's behavior is honest.

Hamilton Cycle (Continued)

With each successive experiment it becomes less and less likely that Peggy always guessed which challenge Victor would choose to be able to pass it rather than being prepared for both by actually knowing the tour. After sufficiently many tests, Victor will be probabilistically certain that Peggy does indeed know a tour in G, yet he will not himself know the tour.

With k tests, his doubt $\frac{1}{2^k}$ approaches 0. With just 7 tests this is less than 1%.

If Peggy were dishonest, she could rig Victor's coin so that she would not actually have to know a tour in G to pass all of the challenges: she could just be prepared to pass whichever was chosen. It is assumed, however, that Peggy's behavior is honest.