# Trapdoor simulation of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

## Algorithms in CS courses

"WHAT is your algorithm?"

Trapdoor simulation
of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

# Trapdoor simulation of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

# Trapdoor simulation of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

Trapdoor simulation
of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

Trapdoor simulation
of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

Algorithms in CS courses

“WHAT is your algorithm?”

“Heapsort. Here’s the code.”

“WHAT does it accomplish?”

“It sorts the input array in place.
Here’s a proof.”

“WHAT is its run time?”

“$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here’s a proof.”

# Trapdoor simulation of quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:
Tung Chou
Technische Universiteit Eindhoven

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

r simulation
tum algorithms

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

rk with:
ou
che Universiteit Eindhoven

Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

Algorith

Critical

How har

on
thms

n
is at Chicago &
siteit Eindhoven

siteit Eindhoven

Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms for har

Critical question fo
How hard is ECDL

ago &
hoven

hoven

Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

Algorithms for hard problem

Critical question for ECC se

How hard is ECDLP?

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

## Algorithms in CS courses

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons; and $\Theta(n \lg n)$ comparisons for most inputs. Here's a proof."

"You may pass."

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

ms in CS courses

is your algorithm?"

rt. Here's the code."

does it accomplish?"

the input array in place.
proof."

is its run time?"

$n$) comparisons;
$n \lg n$) comparisons
inputs. Here's a proof."

ay pass."

Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Ga
inadequa
of a neg

courses

gorithm?"

the code."

ccomplish?"

array in place.

time?"

isons;

parisons

Here's a proof."

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lam

inadequately speci

of a negating rho

”

?”

lace.

oof.”

## Algorithms for hard problems

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for “strong” ECC groups of prime order $\ell$:
Latest “negating” variants of “distinguished point” rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it’s true?

2000 Gallant–Lambert–Vans
inadequately specified stater
of a negating rho algorithm.

## Algorithms for hard problems

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for “strong” ECC groups of prime order $\ell$: Latest “negating” variants of “distinguished point” rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it’s true?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

## Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $\ell$: Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

ms for hard problems

question for ECC security:
rd is ECDLP?

d estimate for “strong”
oups of prime order $\ell$:
negating” variants of
uished point” rho methods
n average ECDLP instance
$0.886\sqrt{\ell}$ additions.

roven? No!

rovable? Maybe not!

do we think it’s true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
**Experiments!**

Similar s
we don’t
best fact

d problems

or ECC security:
LP?

for "strong"
me order $\ell$:
variants of
nt" rho methods
ECDLP instance
dditions.

!

Maybe not!

nk it's true?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds?
**Experiments!**

Similar story for R
we don't have pro
best factoring alg

s

curity:

ng"
$\ell$:
of
ethods
stance

!

e?

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

Similar story for RSA securit
we don't have proofs for the
best factoring algorithms.

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds?
**Experiments!**

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

MQ-based cryptography: we don't have proofs for the best system-solving algorithms.

2000 Gallant–Lambert–Vanstone: inadequately specified statement of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is *non-functional*.

See 2011 Bernstein–Lange–Schwabe for more history and better algorithms.

Why do we believe that the latest algorithms work at the claimed speeds? **Experiments!**

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

MQ-based cryptography: we don't have proofs for the best system-solving algorithms.

Confidence relies on experiments.

llant–Lambert–Vanstone:
ately specified statement
ating rho algorithm.

s–Kleinjung–Lenstra:
le interpretation of
orithm is *non-functional*.

1 Bernstein–Lange–
e for more history
ter algorithms.

we believe that
st algorithms work
laimed speeds?
**ents!**

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

Where's

Quantur
is movin
into algo

Example
exponen
Bernstei

Don't ex
for the b
to attacl

How do
in analys
Quantur

bert–Vanstone:
fied statement
algorithm.

ng–Lenstra:
etation of
*non-functional*.

n–Lange–
history
hms.

e that
ms work
eeds?

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

MQ-based cryptography: we don't have proofs for the best system-solving algorithms.

Confidence relies on experiments.

Where's my quant

Quantum-algorithr
is moving beyond
into algorithms wi

Example: subset-s
exponent ≈0.241 f
Bernstein–Jeffery–

Don't expect proo
for the best quant
to attack post-qua

How do we obtain
in analysis of these
Quantum experime

stone:
ment

n:

onal.

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

MQ-based cryptography: we don't have proofs for the best system-solving algorithms.

Confidence relies on experiments.

Where's my quantum compu

Quantum-algorithm design
is moving beyond textbook s
into algorithms without proo

Example: subset-sum
exponent ≈0.241 from 2013
Bernstein–Jeffery–Lange–Me

Don't expect proofs or prova
for the best quantum algorit
to attack post-quantum cryp

How do we obtain confidenc
in analysis of these algorithm
Quantum experiments are ha

Similar story for RSA security: we don't have proofs for the best factoring algorithms.

Code-based cryptography: we don't have proofs for the best decoding algorithms.

Lattice-based cryptography: we don't have proofs for the best lattice algorithms.

MQ-based cryptography: we don't have proofs for the best system-solving algorithms.

Confidence relies on experiments.

## Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent $\approx$0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

story for RSA security:
t have proofs for the
toring algorithms.

sed cryptography:
t have proofs for the
oding algorithms.

based cryptography:
t have proofs for the
tice algorithms.

ed cryptography:
t have proofs for the
tem-solving algorithms.

nce relies on experiments.

Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent ≈0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

Where's

Analogy
a $2^{80}$ NF

SA security:
ofs for the
orithms.

ography:
ofs for the
orithms.

tography:
ofs for the
hms.

graphy:
ofs for the
g algorithms.

on experiments.

Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent ≈0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

Where's my big co

Analogy: Public h
a $2^{80}$ NFS RSA-10

ty:
e
e
e
e
ms.
nents.

## Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent ≈0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

## Where's my big computer?

Analogy: Public hasn't carri
a $2^{80}$ NFS RSA-1024 experi

## Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent $\approx$0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

## Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent $\approx 0.241$ from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

## Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs.

Example: subset-sum exponent ≈0.241 from 2013 Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability for the best quantum algorithms to attack post-quantum crypto.

How do we obtain confidence in analysis of these algorithms? Quantum experiments are hard.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## my quantum computer?

m-algorithm design
g beyond textbook stage
orithms without proofs.

e: subset-sum
t ≈0.241 from 2013
n–Jeffery–Lange–Meurer.

xpect proofs or provability
best quantum algorithms
k post-quantum crypto.

we obtain confidence
sis of these algorithms?
m experiments are hard.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## Simulati

An algor
is a com
of the al
*for a par*

tum computer?

m design
textbook stage
thout proofs.

sum
from 2013
-Lange–Meurer.

fs or provability
um algorithms
antum crypto.

confidence
e algorithms?
ents are hard.

Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

Simulation

An algorithm simu
is a computer-assis
of the algorithm's
*for a particular inp*

uter?

stage
ofs.

.
eurer.

ability
thms
oto.

ce
ns?
ard.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## Simulation

An algorithm simulation is a computer-assisted proof of the algorithm's performar *for a particular input*.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## Simulation

An algorithm simulation is a computer-assisted proof of the algorithm's performance *for a particular input*.

## Where's my big computer?

Analogy: Public hasn't carried out a $2^{80}$ NFS RSA-1024 experiment.

But public has carried out $2^{50}$, $2^{60}$, $2^{70}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing $2^{80}$ operations on $2^{40}$ qubits; compare to today's challenges of $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## Simulation

An algorithm simulation is a computer-assisted proof of the algorithm's performance *for a particular input*.

Compared to traditional proofs:

Theorem statement is easier. Steps in proof are easier. Don't need to generalize beyond a single input.

Provability is guaranteed.

Proof has computer assistance, so less chance of error.

my big computer?

: Public hasn't carried out
FS RSA-1024 experiment.

lic has carried out
, $2^{70}$ NFS experiments.
y not too much
ation error for $2^{80}$.

arger extrapolation
quantum situation.
attacker performing
rations on $2^{40}$ qubits;
to today's challenges
, $2^3$, $2^4$, $2^5$, $2^6$ qubits.

## Simulation

An algorithm simulation
is a computer-assisted proof
of the algorithm's performance
*for a particular input*.

Compared to traditional proofs:

Theorem statement is easier.
Steps in proof are easier.
Don't need to generalize
beyond a single input.

Provability is guaranteed.

Proof has computer assistance,
so less chance of error.

The star
of an alg

Compute
and $t_0$, $t$
such tha
algorithn

Prove th
matches

Special
The com
the origi
plus prin
Particula

omputer?

asn't carried out
024 experiment.

rried out
experiments.
much
for $2^{80}$.

polation
ituation.
performing
$2^{40}$ qubits;
s challenges
$2^5$, $2^6$ qubits.

Simulation

An algorithm simulation
is a computer-assisted proof
of the algorithm's performance
*for a particular input*.

Compared to traditional proofs:

Theorem statement is easier.
Steps in proof are easier.
Don't need to generalize
beyond a single input.

Provability is guaranteed.

Proof has computer assistance,
so less chance of error.

The standard struc
of an algorithm sin

Compute $s_0, s_1, s_2$
and $t_0, t_1, t_2, \ldots$
such that $s_i$ repres
algorithm state at

Prove that the co
matches the origin

Special case: expe
The computation
the original algorit
plus printouts of s
Particularly easy p

ed out
ment.

nts.

;
s;
es
its.

Simulation

An algorithm simulation
is a computer-assisted proof
of the algorithm's performance
*for a particular input*.

Compared to traditional proofs:

Theorem statement is easier.
Steps in proof are easier.
Don't need to generalize
beyond a single input.

Provability is guaranteed.

Proof has computer assistance,
so less chance of error.

The standard structure
of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$
and $t_0, t_1, t_2, \ldots$
such that $s_i$ represents
algorithm state at time $t_i$.

Prove that the computation
matches the original algorith

Special case: experiment.
The computation *is*
the original algorithm
plus printouts of state.
Particularly easy proof.

## Simulation

An algorithm simulation is a computer-assisted proof of the algorithm's performance *for a particular input*.

Compared to traditional proofs:

Theorem statement is easier.
Steps in proof are easier.
Don't need to generalize beyond a single input.

Provability is guaranteed.

Proof has computer assistance, so less chance of error.

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment.
The computation *is* the original algorithm plus printouts of state.
Particularly easy proof.

on

rithm simulation
puter-assisted proof
lgorithm's performance
*rticular input*.

ed to traditional proofs:

n statement is easier.
proof are easier.
eed to generalize
a single input.

ity is guaranteed.

as computer assistance,
hance of error.

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

Simulati

"If you c
a quantu
pre-quar
have an
algorithr

ulation
sted proof
performance
*out*.

itional proofs:

nt is easier.
easier.
eralize
put.

anteed.

er assistance,
error.

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

Simulation of quan

"If you can efficien
a quantum algorit
pre-quantum comp
have an efficient p
algorithm for the s

f

nce

ofs:

r.

nce,

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

Simulation of quantum algo

"If you can efficiently simula a quantum algorithm using a pre-quantum computer then have an efficient pre-quantu algorithm for the same prob

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

The standard structure
of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$
and $t_0, t_1, t_2, \ldots$
such that $s_i$ represents
algorithm state at time $t_i$.

Prove that the computation
matches the original algorithm.

Special case: experiment.
The computation *is*
the original algorithm
plus printouts of state.
Particularly easy proof.

## Simulation of quantum algorithms

"If you can efficiently simulate
a quantum algorithm using a
pre-quantum computer then you
have an efficient pre-quantum
algorithm for the same problem."

No, not necessarily!

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$ such that $s_i$ represents algorithm state at time $t_i$.

Prove that the computation matches the original algorithm.

Special case: experiment. The computation *is* the original algorithm plus printouts of state. Particularly easy proof.

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

Ah, but did I say that the simulation takes only this input?

ndard structure
gorithm simulation:

e $s_0, s_1, s_2, \ldots$
$t_1, t_2, \ldots$
nt $s_i$ represents
m state at time $t_i$.

hat the computation
the original algorithm.

case: experiment.
nputation *is*
nal algorithm
ntouts of state.
arly easy proof.

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

Ah, but did I say that the simulation takes only this input?

## Trapdoo

Input to
to be inp

Simulati
that mal
faster th

Typical
- Algorit
- Algorit
- Simula

This is s
can try
understa

cture

mulation:

, . . .

sents

time $t_i$.

mputation

nal algorithm.

eriment.

*is*

thm

tate.

roof.

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

Ah, but did I say that the simulation takes only this input?

## Trapdoor simulatio

Input to simulatior

to be input to orig

Simulation can use

that makes simula

faster than origina

Typical example:

- Algorithm input:
- Algorithm outpu
- Simulation input

This is still useful:

can try many choi

understand algorit

nm.

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

Ah, but did I say that the simulation takes only this input?

## Trapdoor simulation

Input to simulation doesn't to be input to original algori

Simulation can use extra inp that makes simulation much faster than original algorithn

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful:
can try many choices of $x$,
understand algorithm for $f($

## Simulation of quantum algorithms

"If you can efficiently simulate a quantum algorithm using a pre-quantum computer then you have an efficient pre-quantum algorithm for the same problem."

No, not necessarily!

"Yes, you do! Simply run the simulation on the same input and extract the original algorithm's output from the final state."

Ah, but did I say that the simulation takes only this input?

## Trapdoor simulation

Input to simulation doesn't have to be input to original algorithm.

Simulation can use extra input that makes simulation much faster than original algorithm.

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful:
can try many choices of $x$,
understand algorithm for $f(x)$.

on of quantum algorithms

can efficiently simulate
um algorithm using a
ntum computer then you
efficient pre-quantum
m for the same problem."

necessarily!

u do! Simply run the
on on the same input and
the original algorithm's
rom the final state."

did I say that the
on takes only this input?

## Trapdoor simulation

Input to simulation doesn't have to be input to original algorithm.

Simulation can use extra input that makes simulation much faster than original algorithm.

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful:
can try many choices of $x$,
understand algorithm for $f(x)$.

For com

Often se
in tradit

Typical
$(x, i) \mapsto$
Formula

Simulati
Given $x$,
for each
simulatio
Doesn't
that wor
Proof ca

ntum algorithms

ntly simulate
hm using a
puter then you
pre-quantum
same problem."

y!

nply run the
same input and
l algorithm's
nal state."

that the
nly this input?

Trapdoor simulation

Input to simulation doesn't have to be input to original algorithm.

Simulation can use extra input that makes simulation much faster than original algorithm.

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful:
can try many choices of $x$,
understand algorithm for $f(x)$.

For comparison:

Often see $x$ inside
in traditional algor

Typical proof has
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven

Simulation is more
Given $x$,
for each $i$,
simulation comput
Doesn't need unifi
that works for all
Proof can work "l

rithms

ate
a
you
m
lem.”

ne
t and
m’s
’

put?

## Trapdoor simulation

Input to simulation doesn’t have to be input to original algorithm.

Simulation can use extra input that makes simulation much faster than original algorithm.

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful:
can try many choices of $x$,
understand algorithm for $f(x)$.

For comparison:

Often see $x$ inside proofs
in traditional algorithm anal

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductivel

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn’t need unified formul
that works for all $x, i$.
Proof can work “locally”.

## Trapdoor simulation

Input to simulation doesn't have to be input to original algorithm.

Simulation can use extra input that makes simulation much faster than original algorithm.

Typical example:

- Algorithm input: $f(x)$.
- Algorithm output: $x$.
- Simulation input: $x$.

This is still useful: can try many choices of $x$, understand algorithm for $f(x)$.

For comparison:

Often see $x$ inside proofs in traditional algorithm analyses.

Typical proof has formula $(x, i) \mapsto (s_i, t_i)$. Formula is proven inductively.

Simulation is more flexible. Given $x$, for each $i$, simulation computes $(s_i, t_i)$. Doesn't need unified formula that works for all $x, i$. Proof can work “locally”.

r simulation

simulation doesn't have
put to original algorithm.

on can use extra input
kes simulation much
an original algorithm.

example:
thm input: $f(x)$.
thm output: $x$.
ation input: $x$.

till useful:
many choices of $x$,
nd algorithm for $f(x)$.

For comparison:

Often see $x$ inside proofs
in traditional algorithm analyses.

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductively.

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn't need unified formula
that works for all $x, i$.
Proof can work "locally".

Proof of

2014.04
Simulati
proof of
distinctn

on

n doesn't have
ginal algorithm.

e extra input
tion much
l algorithm.

: $f(x)$.
ut: $x$.
t: $x$.

ces of $x$,
hm for $f(x)$.

For comparison:

Often see $x$ inside proofs
in traditional algorithm analyses.

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductively.

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn't need unified formula
that works for all $x, i$.
Proof can work "locally".

Proof of concept

2014.04 Chou $\rightarrow$
Simulation shows
proof of 2003 Aml
distinctness algorit

have
ithm.

out

n.

x).

For comparison:

Often see $x$ inside proofs
in traditional algorithm analyses.

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductively.

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn't need unified formula
that works for all $x, i$.
Proof can work "locally".

Proof of concept

2014.04 Chou $\rightarrow$ Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

For comparison:

Often see $x$ inside proofs
in traditional algorithm analyses.

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductively.

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn't need unified formula
that works for all $x, i$.
Proof can work "locally".

## Proof of concept

2014.04 Chou $\rightarrow$ Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

For comparison:

Often see $x$ inside proofs
in traditional algorithm analyses.

Typical proof has formula
$(x, i) \mapsto (s_i, t_i)$.
Formula is proven inductively.

Simulation is more flexible.
Given $x$,
for each $i$,
simulation computes $(s_i, t_i)$.
Doesn't need unified formula
that works for all $x, i$.
Proof can work "locally".

## Proof of concept

2014.04 Chou $\to$ Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

Ambainis: Yes, thanks, will fix.

For comparison:

Often see $x$ inside proofs in traditional algorithm analyses.

Typical proof has formula $(x, i) \mapsto (s_i, t_i)$. Formula is proven inductively.

Simulation is more flexible. Given $x$, for each $i$, simulation computes $(s_i, t_i)$. Doesn't need unified formula that works for all $x, i$. Proof can work "locally".

Proof of concept

2014.04 Chou $\to$ Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou $\to$ Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

For comparison:

Often see $x$ inside proofs in traditional algorithm analyses.

Typical proof has formula $(x, i) \mapsto (s_i, t_i)$. Formula is proven inductively.

Simulation is more flexible. Given $x$, for each $i$, simulation computes $(s_i, t_i)$. Doesn't need unified formula that works for all $x, i$. Proof can work "locally".

## Proof of concept

2014.04 Chou $\rightarrow$ Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou $\rightarrow$ Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.