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 Θ(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.”
for simulation
quantum algorithms

D. 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.”

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?
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?
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?
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 Θ(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 ℓ:

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?

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?”

$O(n \log n)$ comparisons; and $\Theta(n \log n)$ 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?

**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?”

$O(n \log n)$ comparisons; and $\Theta(n \log n)$ 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?

---

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?

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?


2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is \textit{non-functional}. 
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?


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.


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!
Algorithms for hard problems

Critical question for ECC security: How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order \( \ell \):

- "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?


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.
Algorithms for hard problems

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong" ECC groups of prime order $l$:

```
Latest "negating" variants of "distinguished point" rho methods break an average ECDLP instance using $\approx 0.886 \sqrt{l}$ 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!**

Similar story for RSA security:
we don’t have proofs for the best factoring algorithms.
Critical question for ECC security: How hard is ECDLP?

Standard estimate for “strong” ECC groups of prime order:

\[ \approx 0.886 \sqrt{\text{additions.}} \]

Is this proven? No!
Is this provable? Maybe not!
So why do we think it’s true?


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.


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.

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.

2010 Bos–Kleinjung–Lenstra: a plausible interpretation of that algorithm is \textit{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? \textbf{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.

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.

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.


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.

Where’s my quantum computer?
Quantum-algorithm design is moving beyond the textbook stage into algorithms without proofs.

Example: subset-sum exponent \( \approx 0 \): \( 2^{41} \) 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.
Inadequately specified statement of a negating rho algorithm.

A plausible interpretation of that algorithm is non-functional.

See 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.

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 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.
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.
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.
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.
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.

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.
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 $\approx 0.241$ 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 $2^{80}$.
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.
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.

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.

Simulation

An algorithm simulation is a computer-assisted proof of the algorithm’s performance for a particular input.
Where's my quantum computer?

Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs. Example: subset-sum exponent $\approx 0$ 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.

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.
Quantum-algorithm design is moving beyond textbook stage into algorithms without proofs. Example: subset-sum exponent $\approx 0$ 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.

**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.
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.
Where's my big computer?

Analogy: Public hasn’t carried out a $2^{80}$ NFS RSA-1024 experiment. Public has carried out $2^{70}$, $2^{60}$, $2^{50}$ NFS experiments. Hopefully not too much extrapolation error for $2^{80}$.

Larger extrapolation for the quantum situation. Imagine attacker performing 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.

The standard structure of an algorithm simulation:

Computations are $s_0$, $s_1$, $s_2$, $t_0$, $t_1$, $t_2$, $s_0$ represents algorithm state at time $t_0$, $t_1$, $t_2$, $t_3$, $t_4$, $t_5$.

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

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.
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.
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.
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.

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.”
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.

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.”
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.

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.”
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?
The standard structure of an algorithm simulation:

Compute $s_0, s_1, s_2, \ldots$ and $t_0, t_1, t_2, \ldots$

where $s_i$ represents the 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?

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 $f(x)$.

Trapdoor simulation
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 $f(x)$.
### 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 $f(x)$. 

<table>
<thead>
<tr>
<th>Simulation of quantum algorithms</th>
<th>Trapdoor simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>“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.”</td>
<td>Input to simulation doesn’t have to be input to original algorithm.</td>
</tr>
<tr>
<td>No, not necessarily!</td>
<td>Simulation can use extra input that makes simulation much faster than original algorithm.</td>
</tr>
<tr>
<td>“Yes, you do! Simply run the simulation on the same input and extract the original algorithm’s output from the final state.”</td>
<td>Typical example:</td>
</tr>
<tr>
<td>Ah, but did I say that the simulation takes only this input?</td>
<td>- Algorithm input: $f(x)$.</td>
</tr>
<tr>
<td></td>
<td>- Algorithm output: $x$.</td>
</tr>
</tbody>
</table>
| | - Simulation input: $x$.
| | This is still useful: can try many choices of $x$, understand algorithm for $f(x)$.
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 $f(x)$.
Simulation of quantum algorithms

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!

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 $f(x)$.

For comparison:

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

Typical proof has formula $\langle x, i \rangle \mapsto \langle s_i, t_i \rangle$.

Formula is proven inductively.

Simulation is more flexible.

Given $x$, for each $i$, simulation computes $\langle s_i, t_i \rangle$.

Doesn’t need unified formula that works for all $x, i$.

Proof can work “locally.”
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 $f(x)$.

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”.

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 $f(x)$.
**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 $f(x)$.

---

**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”.

---

**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 $f(x)$.
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”.
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”.

Proof of concept
2014.04 Chou → Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.
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 \( (s_i, t_i) \).

Proof of concept
2014.04 Chou \( \rightarrow \) Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

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 $\rightarrow$ 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 → 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 $\rightarrow$ 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.
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.