Are Quantum Computers Really A Threat To Cryptography?

A Practical Overview Of Current State-Of-The-Art Techniques With Some Interesting Surprises

www.quintessencelabs.com

About me

- Cryptography & Malware Researcher at TrustDefender
- CTO @ ThreatMetrix
- Quantum Technologies @ Qlabs
  - http://www.quintessencelabs.com



©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Cryptography

- Symmetric cryptosystem (shared secred key)
  - The same key (the secret key) is used to encrypt and decrypt the message
  - Examples: AES

- Asymmetric cryptosystem (public & private key)
  - Use a public key to encrypt a message and a private key to decrypt it
  - Examples: RSA, ECC

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

- Virtually all current cryptosystems are “computationally secure”
  - Not decodable with available computing power, but no proof that you can’t break them
  - To factor a 2048-bit RSA key, the best classical algorithm needs ~ 10^{34} steps and ~317 trillion years on a classical ThZ Computer (with a trillion operations per second):

- There are information-theoretic cryptosystems (e.g. One-Time-Pad)
  - However to enjoy the benefits of the proof, many assumptions must be met
    - E.g. secret key is truly random. Secret key has the same length as the message, …

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Attacks on Cryptosystems

- Symmetric encryption (e.g. AES, $N = 256$)
  - Key can be recovered classically with a computational complexity of $O(2^N)$
  - Best quantum algorithm (Grover) provides "only" a squared speedup of $O\left(\sqrt{2^N}\right)$
  - While this is still a massive speedup, doubling the keylength will compensate for this
- Asymmetric encryption (e.g. RSA, ECC)
  - Used virtually everywhere to negotiate a symmetric key (e.g. VPN's, TLS, Diffie-Hellman, Digital Signatures, …)
  - Multiple quantum algorithms available
  - Focus for this talk

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Attacks on Cryptosystems

- Symmetric encryption (e.g. AES, N=256)
  - Key can be recove[illegible]ically with a [illegible]tational complexity of $O(2^N)$
  - Best quantum algorithm (G[illegible]) p[illegible]es "only" a squared speedup of $O\left(\sqrt{2^N}\right)$
  - While this is still [illegible]ssive speedup, doubling [illegible]length will compensate for this
- Asymmetric encryption (e.g. RSA, ECC)
  - Used virtually everywhere to negotiate a symmetric key (e.g. VPN's, TLS, Diffie-Hellman, Digital Signatures, …)
  - Multiple quantum algorithms available
  - Focus for this talk

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

RSA encryption – How it works

- Alice chooses two distinct prime numbers p, q which are kept secret
- Key generation
  - Compute $n = pq$
  - Compute $\lambda(n) = \mathrm{lcm}(\lambda(p), \lambda(q))$
  - Choose e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$, meaning e, $\lambda(n)$ are co-prime
  - n, e is released as the public key
  - Calculate $d = e^{-1} \pmod{\lambda(n)}$
  - d is the private key

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

RSA encryption – How it works

- Bob encrypts a message M with Alice's public key (n,e)
  - Turn M into integer m (the padded plaintext), $0 \leq m < n$ (padding scheme)
  - Ciphertext $c = m^{\mathrm{e}}$ (mod n)

- Alice can now decrypt this ciphertext c by using private key d
  - $c^d = (m^{\mathrm{e}})^d = m$ (mod n)
  - Given m, Alice can recover M by reversing the padding scheme

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

How to retrieve the private key from a public key?

- Given a public key (n,e), how can one "crack" the private key d?
  - From n, find prime numbers p,q such that $n = pq$
  - Calculate $\lambda(n)$ = lcm($\lambda(p)$, $\lambda(q)$)
  - Private key $d = \mathrm{e}^{-1} \pmod{\lambda(n)}$

- So all I have to do is to find p,q such that $n = pq$, right?

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

How to factor an integer?

- No algorithm has been published that can factor all integers in polynomial time (e.g. $O(n^k)$ for some constant k)
- Most algorithms are of exponential complexity
  - Best algorithm for large n is GNFS (General Number Field Sieve) which is sub-exponential, but still massively bigger than polynomial

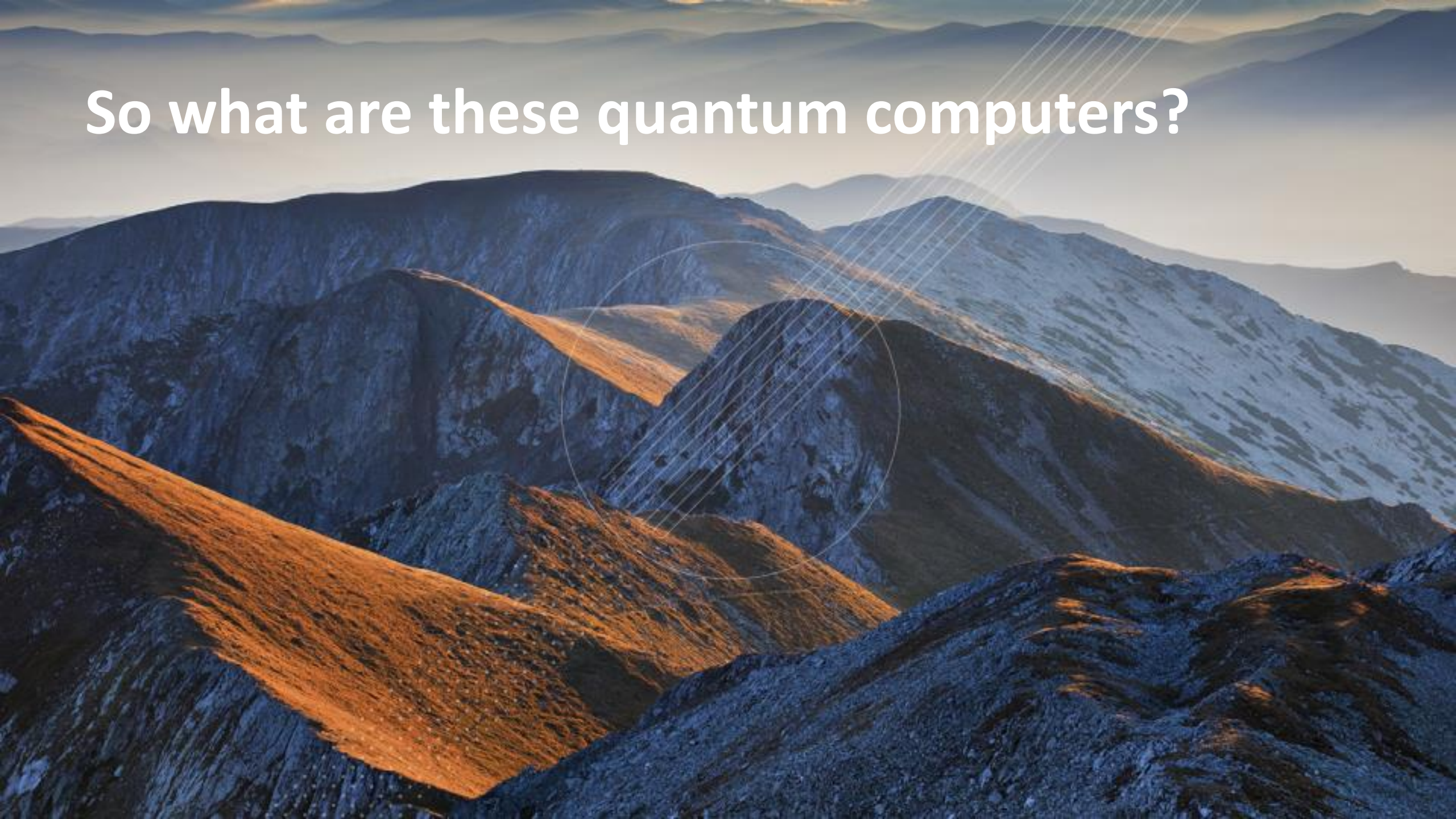
- Shor's algorithm can solve this with only polynomial complexity
  - The good news is that Shor's algorithm can't be implemented on a classical computer
- That difference is incomprehensible.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Exponential vs polynomial complexity

- Task: factor 2048-bit number
- **Best classical Algorithm (GNFS Algorithm)**
  - ~ 10^{34} steps
  - On classical ThZ Computer (with a trillion operations per second): ~317 trillion years
- **Best quantum algorithm (Shor's Algorithm)**
  - ~ 10^{7} steps
  - On a quantum MhZ computer (with a million operations per second): ~10 seconds
  - Needs 4099 logical qubits

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.


So what are these quantum computers?

Two main types

- Gate Based Quantum Computing (Universal Quantum Computing)
  - IBM, Intel, Microsoft, Alibaba, …
- Adiabatic Quantum Computing
  - E.g. D-Wave

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Two main types

- Gate Based Quantum Computing (Universal Quantum Computing)
  - Start with a known quantum state (input)
  - Apply a sequence of quantum gates (1 or 2 qubit logic gates)
  - Close to classical computing (Input → Compute → Output)

- Adiabatic Quantum Computing (e.g. D-Wave)
  - Encode solutions to physical systems
  - Physical systems tend to be in the lowest energy state (called ground state)
  - Define a Hamiltonian H_f with a ground state that is the solution to a computational problem
  - Evolve system slowly and measure to obtain answer

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum approaches to solve factorization problem

- **Both approaches** can be used to solve the factorization problem

- Universal Gate Quantum Computer
  - Shor's algorithm (1984)

- Quantum Annealing (since 2002)
  - Need to articulate factorization problem as an optimization problem (Hamiltonian H_{f})
  - The Adiabatic theorem guarantees that the ground state at the end is the optimal solution if the transition from H_0 to H_{f} is performed slowly enough

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Computing Introduction

- QC uses quantum-mechanical phenomena such as superposition and entanglement to perform computation

- Basic building block is a qubit – the quantum version of a bit
  - A classical bit is either 0 or 1
  - A qubit is a two-state quantum-mechanical system with two possible outcomes for a measurement (0 or 1) based on probabilities

- Quantum Computers can only run probabilistic algorithms

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Computing Introduction - Superposition

- A qubit is represented by two complex numbers $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle,\ \alpha, \beta \in \mathbb{C}$
  - $|0\rangle, |1\rangle$ represent the orthogonal qubits with 0,1 as measurement-outcome
  - α, β are probability amplitudes.
  - Measurement in the standard basis , the probability of outcome $|0\rangle$ with value 0 is $|\alpha|^2$, the probability of outcome $|1\rangle$ with value 1 is $|\beta|^2$

- Each measurement is a probability, typically resulting in the need to execute the same program multiple times

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Computing Introduction - Entanglement

- Quantum Entanglement is a property between two or more qubits that allows these qubits to express higher correlation than it is possible in classical systems
- Simple Example: Bell State of two qubits
  - $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$
  - Equal probabilities of measuring outcome of $|00\rangle \; or \; |11\rangle$ as $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$
  - Imagine now you take these two qubits and give one to Alice and one to Bob
    - If Alice measures her qubit to be $|0\rangle$, Bob must now get exactly the same outcome with perfect correlation

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Computing Introduction - Exponential large size

- Two classical bits can be in four possible states (00, 01, 10, or 11), but only one of them at any time.
  - This limits the computer to processing one input at a time.
- In the quantum case, two qubits can also represent the exact same four states (00, 01, 10, or 11).
  - The difference is, because of superposition, the qubits can represent all four at the same time.

- If you have n qubits, you can simultaneously represent 2^n states

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

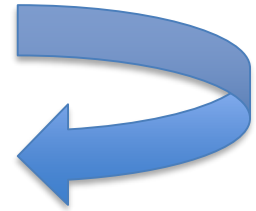
Quantum Computer Introduction 101

- We now have everything we need to have to look at Shor's algorithm in more detail
  - Qubit
  - Superposition
  - Entanglement
  - Exponential large size of the state space of a quantum mechanical system

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor algorithm – main idea

- It is possible to factor $N = pq$,
  if you can find the period with respect to r of the sequence x^r (mod N)



  - This isn't useful for classical computers because if N is large, the period is exponentially long
  - However a quantum computer can process an exponential amount of data that is in superposition
  - So they can put the entire sequence into their memory in superposition
  - Quantum computers can now do a quantum fourier transform, which lets them find the period of the sequence

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm: turn factoring problem into period finding

- The number theory that underlines Shor's algorithm relates to periodic modulo sequences
  - Let's look at a number sequence 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, …
  - Now let's look at the same sequence 'modulo 15', that is, the remainder after fifteen divides each of these powers of two: 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, …

- Factorization of N can be reduced to the problem of finding the period of an integer $0 < x < N$ depends on the following result from number theory
  - The function $F(a) = x^a$ mod N is a periodic function where x is an integer coprime to N and a >= 0

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm

- Three phases
  - Turn factoring problem into period finding
    - computed on classical computer
  - Find the period using Quantum Fourier Transform
    - This is the part responsible for the quantum speedup
  - Use the period to find the factors

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm: turn factoring problem into period finding

- Since F(a) is periodic (and x^0 mod N = 1) with period r, that means x^r mod N = 1 and thus r is just the first non-zero power where $x^r = 1 \pmod N$
- This means
  - x^r = 1 mod N
  - $x^r = \left(x^{\frac{r}{2}}\right)^2 = 1 \bmod N$
  - $\left(x^{\frac{r}{2}}\right)^2 - 1 = 0 \bmod N$
  - If r is an even number: $\left(x^{\frac{r}{2}} + 1\right)\left(x^{\frac{r}{2}} - 1\right) = 0 \bmod N$

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm: turn factoring problem into period finding

- So $\left(x^{\frac{r}{2}}+1\right)\left(x^{\frac{r}{2}}-1\right)$ is an integer multiple of N, the number to be factored
- So for as long as $\left(x^{\frac{r}{2}}+1\right)$ or $\left(x^{\frac{r}{2}}-1\right)$ is not a multiple of N, then at least one of them must have a nontrivial factor in common with N
- So computing $\gcd(\left(x^{\frac{r}{2}}-1\right), N)$ and $\gcd(\left(x^{\frac{r}{2}}+1\right), N)$ will obtain a factor for N

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm: turn factoring problem into period finding

As an example, consider $N = 15$. Let's look at all values of $1<x<15$ where x is coprime with 15:

x	$x^a \bmod 15$	**Period** r	$\gcd(x^{r/2}-1, 15)$	$\gcd(x^{r/2}+1, 15)$
2	1,2,4,8,1,2,4,8,1,2,4...	4	3	5
4	1,4,1,4,1,4,1,4,1,4,1...	2	3	5
7	1,7,4,13,1,7,4,13,1,7,4...	4	3	5
8	1,8,4,2,1,8,4,2,1,8,4...	4	3	5
11	1,11,1,11,1,11,1,11,1,11,1...	2	5	3
13	1,13,4,7,1,13,4,7,1,13,4,...	4	3	5
14	1,14,1,14,1,14,1,14,1,14,1,,,	2	1	15

As can be seen, any value of x except 14 will return the factors of 15, that is, 3 and 5. 14 is an example of the special case where $(x^{r/2}+1)$ or $(x^{r/2}-1)$ is a multiple of N and thus another x needs to be tried.

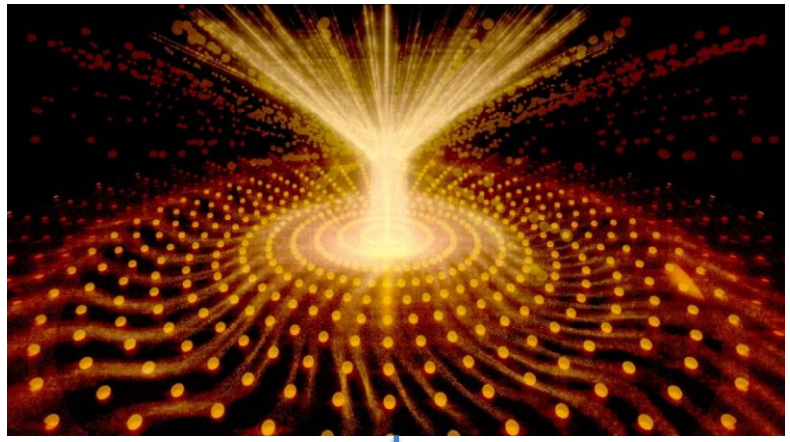
In general, it can be shown that this special case occurs infrequently, so on average only two calls to the period finding machine are sufficient to factor N.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Period Finding (highly simplified)

- Initialize Qubits into an equal superposition
- Compute modular exponentiation
- Perform Quantum Fourier Transform
  - Amplitude of the correct result will be amplified
- Measure the system to obtain the result r

$x^a \bmod N$



r

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure

- Pick a random number $a < N$
- Compute $\gcd(a, N)$
  - If $\gcd(a, N) \neq 1$, this number is a non-trivial factor and we are done
- Use **quantum-period-finding routine** to find r, which denotes the period for $f(x) = a^x \bmod N$
  - If r is odd, go back to step 1
  - If $a^{\frac{r}{2}} = -1 \pmod N$, go back to step 1
- At least one factor of $\gcd\left(\left(a^{\frac{r}{2}} + 1\right), N\right)$ and $\gcd\left(\left(a^{\frac{r}{2}} - 1\right), N\right)$ is a non-trivial factor for N and we are done ☺

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Example (N=15)

- We "randomly" choose: a=7
- We calculate period: r=4
- We have $\gcd\left(\left(7^2 \pm 1\right), 15\right) = \gcd\left((49 \pm 1), 15\right)$
  - $\gcd(48, 15) = 3$
  - $\gcd(50, 15) = 5$

- $15 = 3 \times 5$

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Example (N=15)

- To calculate r=4 with the Quantum Fourier Transform, we use Qiskit (www.qiskit.org) an open-source quantum computing framework
  - That allows us to use a Quantum Simulator or a real Quantum Computing Hardware (e.g. IBM's Q-Experience)

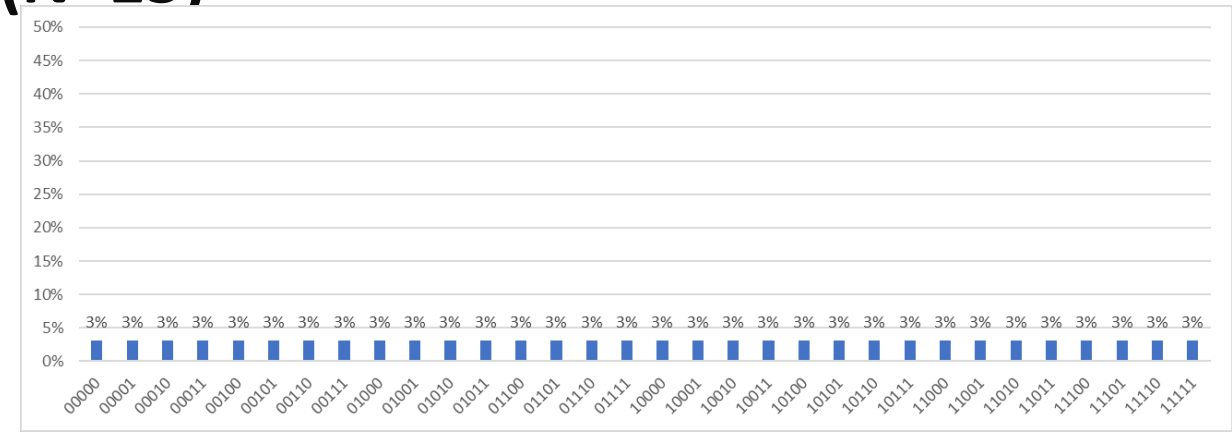
- Good example of the QFT for Shor is here: https://github.com/Qiskit/qiskit-tutorials/blob/ec7c630a15d81583876205a9bee67858fc504911/community/algorithms/shor_algorithm.ipynb

- Basic approach to many Quantum algorithms is Amplitude Amplification

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Example (N=15)

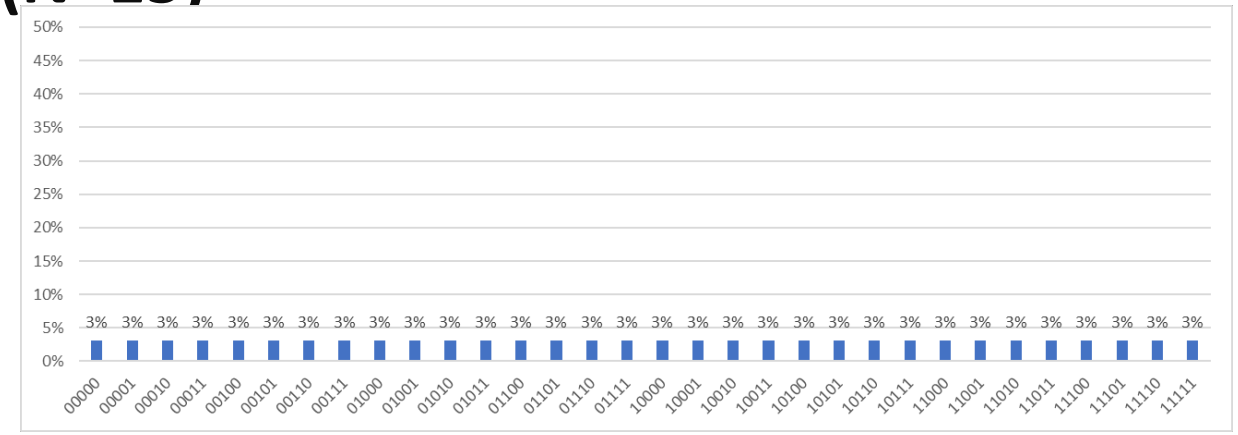
- At the start, system will be in a superposition where all results have an equal probability



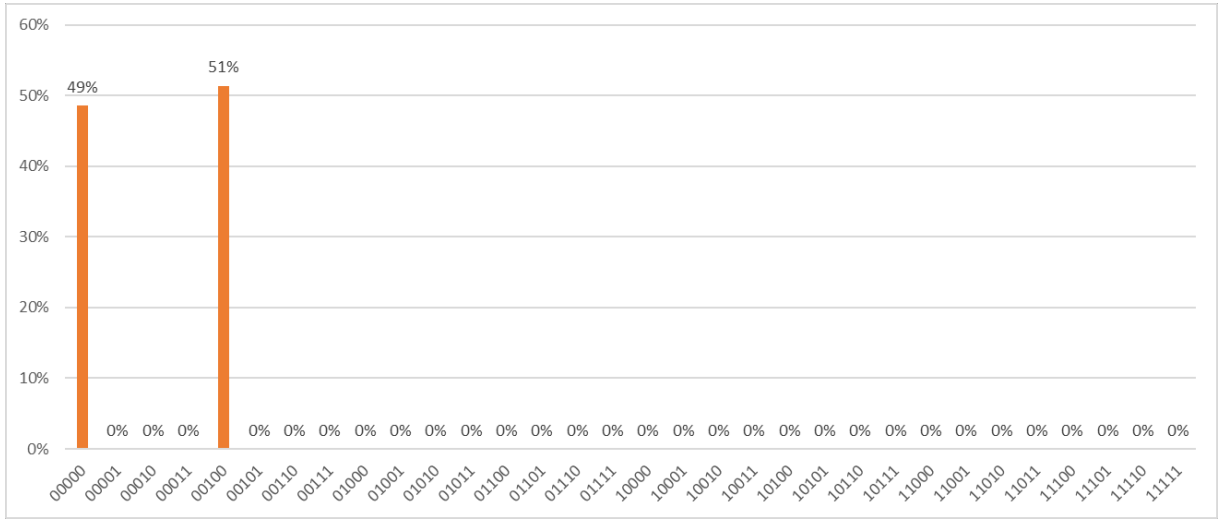
©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Example (N=15)

- At the start, system will be in a superposition where all results have an equal probability



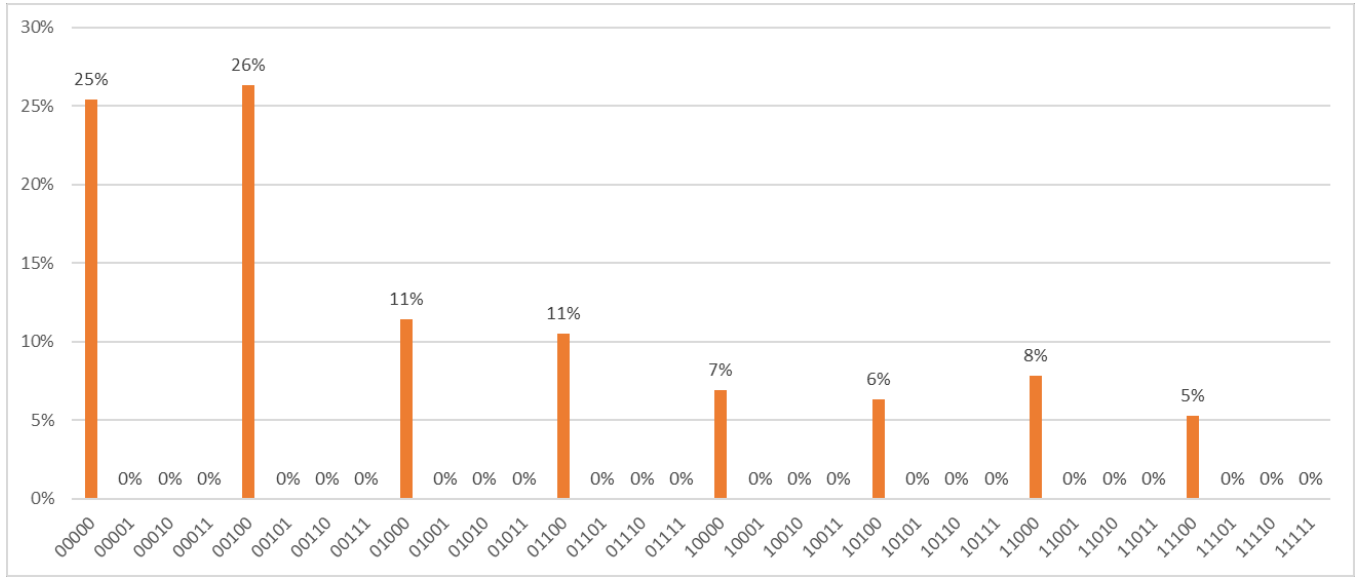
- After execution, the results have elevated probability
  - r = 0 is ignored as a trivial probability, so the result is r = 4
  - Executed on the simulator



©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Example (N=15)

- If we execute this on ibmqx4 (a 5-qubit Quantum Processor from IBM), the results are



- r=4 still has the highest probability, but the result contains much more noise

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Aqua

- Qiskit Aqua (http://www.qiskit.org/aqua) contains libraries for quantum algorithms and makes running Shor (and other algorithms) dead easy

In [1]:
```
from qiskit import BasicAer
from qiskit.aqua import QuantumInstance
from qiskit.aqua import run_algorithm
from qiskit.aqua.algorithms import Shor
```

In [2]:
```
N = 15
shor = Shor(N)
backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
ret = shor.run(quantum_instance)
print("The list of factors of {} as computed by the Shor's algorithm is {}.".format(N, ret['factors'][0]))
```

```
The list of factors of 15 as computed by the Shor's algorithm is [3, 5].
```

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm procedure: Aqua

- Qiskit Aqua (http://www.qiskit.org/aqua) contains libraries for quantum algorithms and makes running Shor (and other algorithms) dead easy

In [1]:
```
from qiskit import BasicAer
from qiskit.aqua import QuantumInstance
from qiskit.aqua import run_algorithm
from qiskit.aqua.algorithms import Shor
```

In [2]:
```
N = 15
shor = Shor(N)
backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024)
ret = shor.run(quantum_instance)
print("The list of factors of {} as computed by the Shor's algorithm is {}.".format(N, ret['factors'][0]))
```

```
backend = BasicAer.get_backend('ibmqx4')
```

```
The list of factors of 15 as computed by the Shor's algorithm is [3, 5].
```

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Shor's algorithm in practice

- So what's the problem?
  - To factor RSA-2048, Shor's algorithm needs 4099 qubits and 100 million gates
    - Qubits and gates need to be fully error-free for a long time
    - Runs in polynomial time!
- Shor's algorithm was never meant to be run on a Quantum Computer.
  - In 1984 when Peter Shor came up with it, Quantum Computers were a fantasy
  - Even today, there are no perfect (logical) qubits
- The quantity of qubits and the noise level are way too high to run Shor's algorithm directly

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.


So let’s look at some of the research how
Shor’s algorithm could be run realistically

Fowler et al, 2012

- In 2012, Fowler et al presented a way to implement Shor using ‘surface codes’ approach, which are basically a two-dimensional array of physical qubits.
- Surface codes allow quantum computers to operate successfully under local errors
- However higher tolerance to errors involve large numbers of qubits
- To factor a 2048-bit RSA integer with a gate error-rate of 0.1%, Fowler et al need around 1,000 million qubits

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

O'Gorman et al 2017

- "Only" 230m qubits are needed to factor a 2048-bit RSA integer
  - Estimate based on various optimizations in the physical connectivity of the qubits and the distillation strategy

- Gheorghiu can reduce this to 170m qubits in 2019

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Gidney, Ekera, 2019

- The biggest contribution came from Gidney, Ekera in 2019 where they were able to estimate the qubits needed to factor a 2048-bit RSA integer to “just” 20m
  - They combined techniques from Griths-Niu 1996, Zalka 2006, Fowler 2012, Ekera-Hastad 2017, Ekera 2017, Ekera 2018, Gidney-Fowler 2019, Gidney 2019

- Let’s look at this research in a bit more detail…

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Gidney, Ekera, 2019

- They were able to do this by
  - Transforming the original factoring problem $N = pq$ into a short discreet logarithm problem
    - Both the classical and the quantum part is similar to Shor, however the period finding has a reduced exponent length
      - translates into an overall reduction in the number of multiplications needed to perform on the quantum computer.
  - Heavy optimizations on various fronts
    - Reduction of the number of multiplications, reduction of the cost of the multiplication
    - Clever post processing which recovers d (= p+q mod r) in 99% of the cases, which means the algorithm mostly only need to be run once on a Quantum Computer!!!

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Gidney, Ekera, 2019

- Short Discreet Algorithm to factor $N = pq$
  - First $y = g^{(N+1)}$ is computed classically, where g is randomly selected from $\mathbb{Z}_N^*$ and of unknown order r
  - Then $d = \log_g y = p + q$ (mod r) is computed quantumly

- For large RSA integers, the order $r > p + q$ with overwhelming probability
  - Hence $d = p + q$ is true.
  - With $N = pq$ & $d = p + q$ (where N and d are both known), it is trivial to recover p and q as the roots of the quadratic equation $p^2 - \mathrm{d}p + N = 0$

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

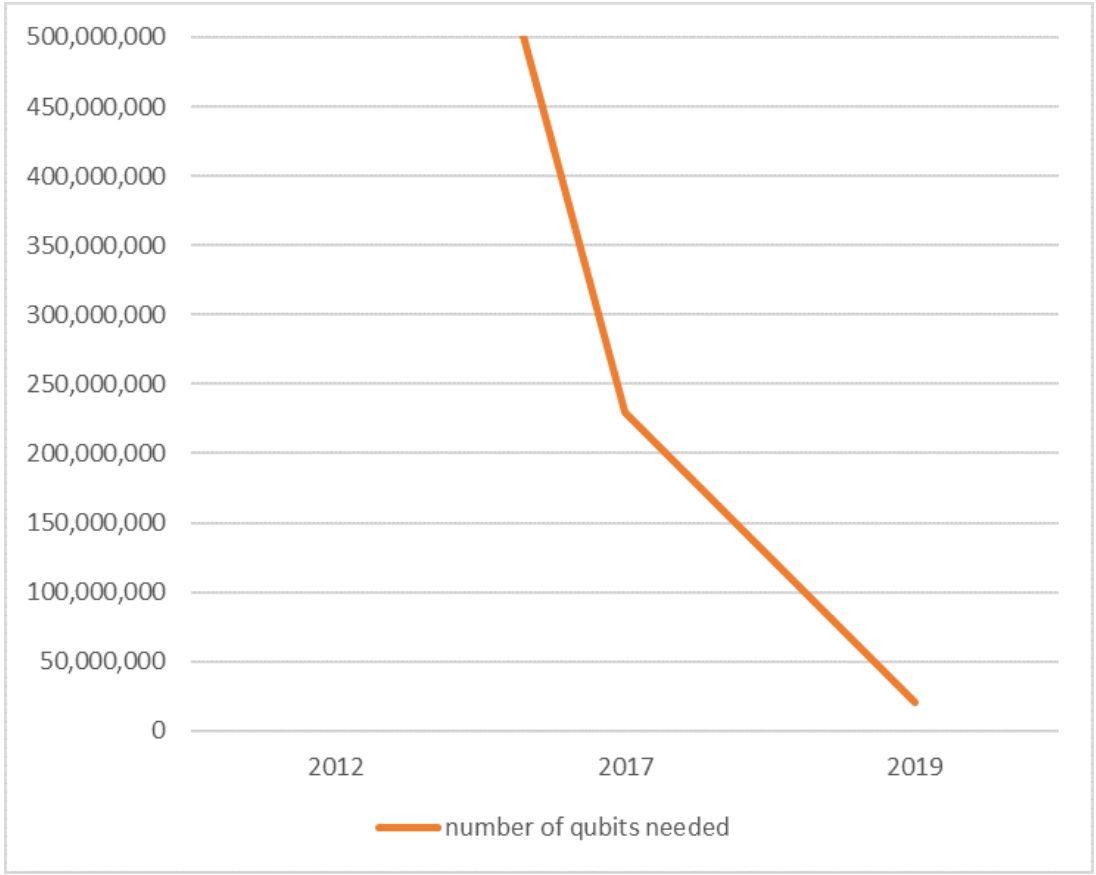
Gidney, Ekera, 2019

- Quantum Part is similar to Shor, except
  - There are two exponents e_1, e_2 of lengths $2m$ and m qubits respectively, for m a positive integer such that $p + q < 2^m$
  - Period finding is performed against the function $f(e_1, e_2) = g^{e_1} y^{e_2}$ rather than $f(\mathrm{e}) = g^{\mathrm{e}}$
  - The total exponent length is $n_e = 3m = 1.5n + O(1)$ compared to $2n$ qubits for Shor
  - This reduction in exponent length will result in the reduction in overall multiplications needed

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Gidney, Ekera, 2019 cont'd

- Using these optimizations, they've been able to improve on Fowler and Gheorgiu by over 100x
- We went from 1bn qubits to 20m in the space of 7 years
- The next set of optimization will be incredibly exciting



©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Gidney, Ekera, 2019: Factoring n-bit RSA integer overview

n	n_e	Parameters d_1	d_2	δ_{off}	c_{mul}	c_{exp}	c_{sep}	Retry Risk	Volume (megaqubitdays) per run	expected	Qubits (megaqubits) per run	Runtime (hours) per run
1024	$3(n/2-1)-40$	15	27	5	5	5	1024	6%	0.5	0.5	9.7	1.3
2048		15	27	4	5	5	1024	31%	4.1	5.9	20	5.1
3072		17	29	6	4	5	1024	9%	19	21	38	12
4096		17	31	9	4	5	1024	5%	48	51	55	22
8192		19	33	4	4	5	1024	5%	480	510	140	86
12288		19	33	3	4	5	1024	12%	1700	1900	200	200
16384		19	33	4	4	5	1024	24%	3900	5100	270	350

TABLE III. Factoring an n bit RSA integer by computing a short discrete logarithm. This table was produced by the script in the ancillary file "estimate_costs.py".

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Annealing



Quantum Annealing

- Quantum Annealing is the process of finding a global minimum of a given objective function.

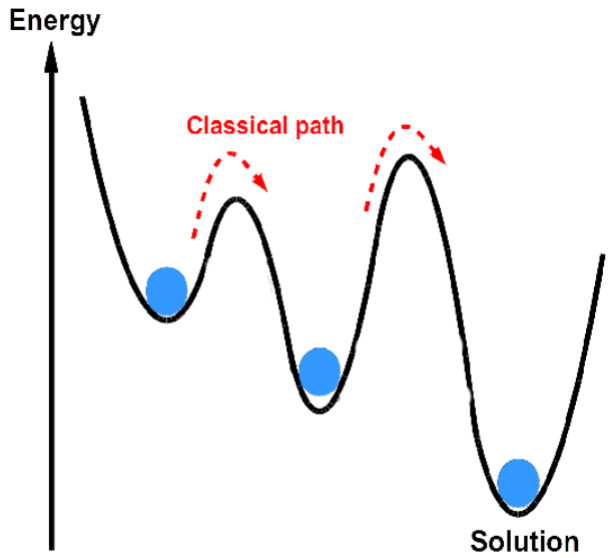
- A quantum computer codifies the optimization problem into a physical system by constructing a Hamiltonian

- The optimal solution to the optimization problem corresponds with the minimum energy state of the system.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Annealing

- The space of energy states resembles a landscape of formed by mountains and valleys
- The solution corresponds to the lowest valley, but how do we find the lowest one?
- Classical Solution
  - Tries to solve this problem by “climbing” the higher energy solutions by increasing the energy (temperature) and letting the system cool down gradually to find the path to the minimum
  - This solution can easily get stuck in a local minima.

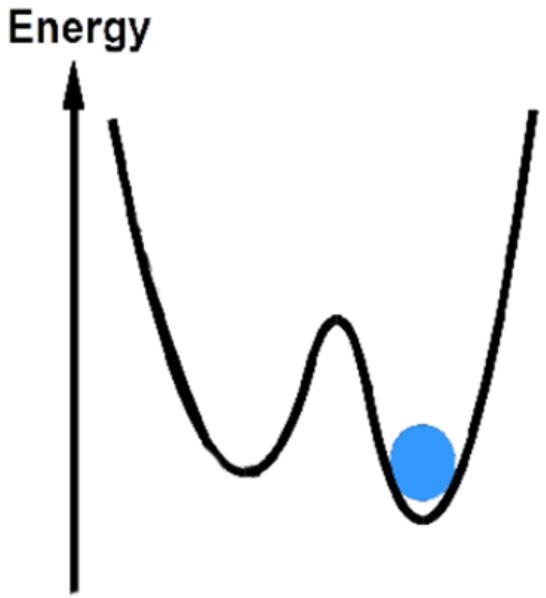


©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Annealing

- In QA, we start in a ground state of some well-known physical system which is easy to prepare (H_0).

- Then we evolve adiabatically (very slowly) the Hamiltonian of this system until it transforms into the problem Hamiltonian H_1

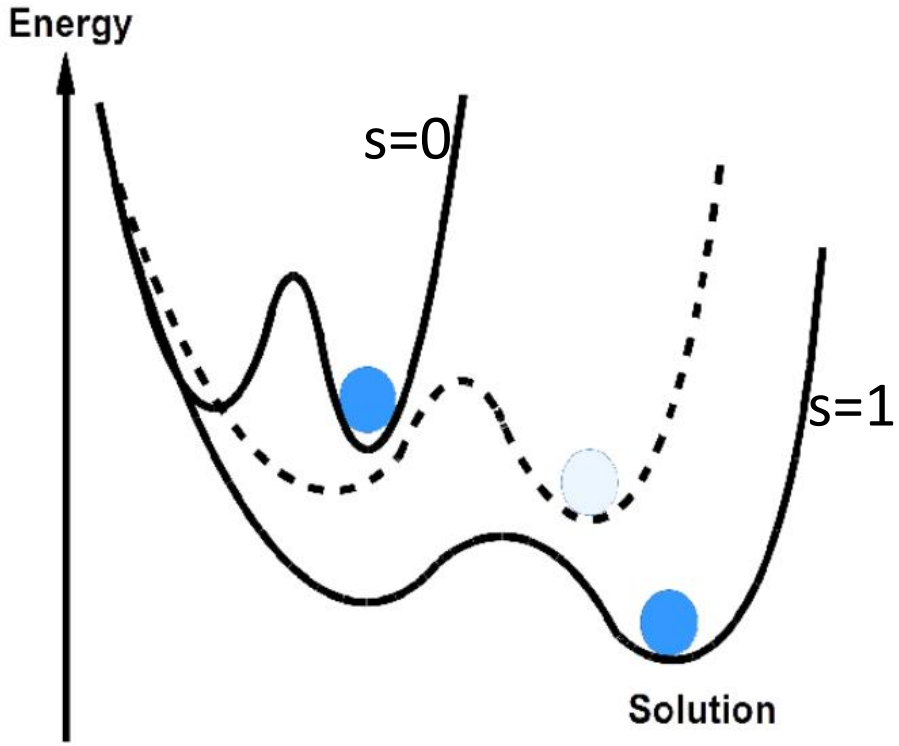


©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Quantum Annealing

- $H = (1 - s)H_0 + sH_1$
  - Initially we compute $s = 0$, so $H = H_0$
- Then we increase s and compute again the ground state of H
- We repeat this process until s=1 and therefore $H = H_1$

- The adiabatic theorem guarantees that the ground state at the end of the computation is the optimal solution.



Adiabatic evolution

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

We need to formulate an optimization problem

- First fundamental research was from Burges, 2002, “Factoring as Optimization”, Microsoft Research, https://www.microsoft.com/en-us/research/publication/factoring-as-optimization/

- The idea is simple: We are looking for p, q so that $N = pq$

- We “just” need to write this as an optimization problem

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

We need to formulate an optimization problem

- $N = pq$
- Binary representation $p = 1 + \sum_{i=1..s_p} 2^i P_i$, $q = 1 + \sum_{i=1..s_q} 2^i Q_i$
  - P_i, Q_i is the i-th bit for p,q,
  - remember that in binary all prime numbers begin and end with a 1

- We can define a cost function (to be minimized)
  - $f\left(P_1, P_2, \dots, P_{s_p}, Q_1, Q_2, \dots, Q_{s_q}\right) = (N - pq)^2$
  - If I find P_i, Q_i so that $f(\dots) = 0$, then N $= pq$ and we are done ☺
  - This is a QUBO, which we can run on a Quantum Annealer (D-Wave)

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

We need to formulate an optimization problem

- Example: $N = 15 = 5 \times 3$
  - $p = (x_1 1) = 2x_1 + 1$
  - $\mathrm{q} = (x_2 x_3 1) = 2^2 x_2 + 2x_3 + 1$

  - $f(x_1, x_2, x_3) = (N - pq)^2 = (15 - (2x_1 + 1)(2^2 x_2 + 2x_3 + 1))^2$
  - $f(x_1, x_2, x_3) = 128\, x_1\, x_2\, x_3 - 56x_1\, x_2 - 48x_1\, x_3 + 16x_2\, x_3 - 52x_1 - 52x_2 - 96x_3 + 196$

- Task: find $x_1, x_{21} x_3$ so that the positive $f(x_1, x_2, x_3)$ is minimal (equal to 0)

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Example N=15 (= 5×3)

- This can be run on D-Wave's Quantum Computer (https://github.com/dwavesystems/demos/tree/master/factoring)
  - Free open-source SDK (dwave-ovean-sdk)

```
# get input from user
print("Enter a number to be factored:")
P = sanitised_input("product", "P", range(2 ** 6))

# send problem to QPU
print("Running on QPU")
output = factor(P)

# output results
pprint(output)
```

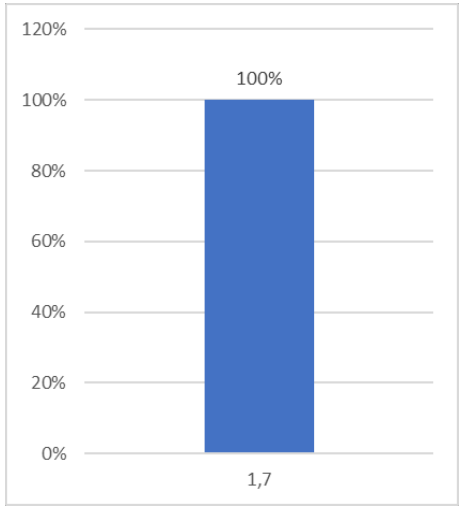
- Not realistic as factoring a $2n$ bit integer requires O(n^2) qubits

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

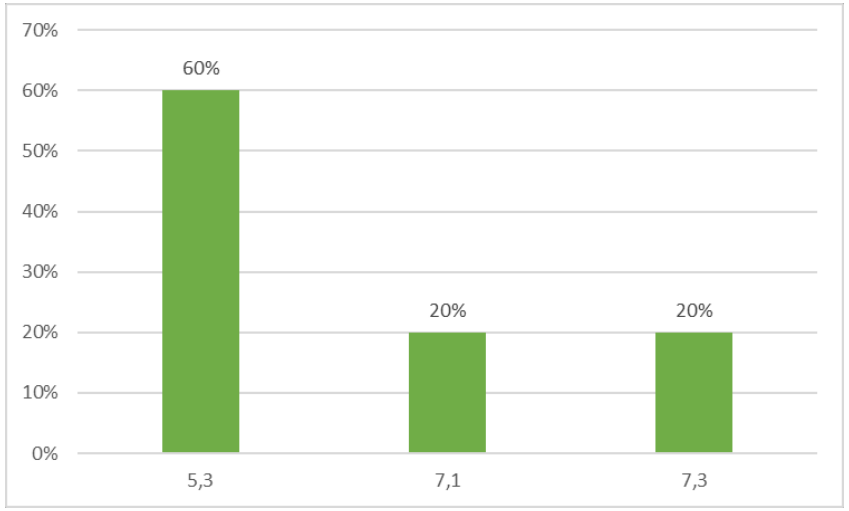
Example: Factoring N=15 (= 5×3) on DWave's QA

- Remember all Quantum Algorithms are probabilistic

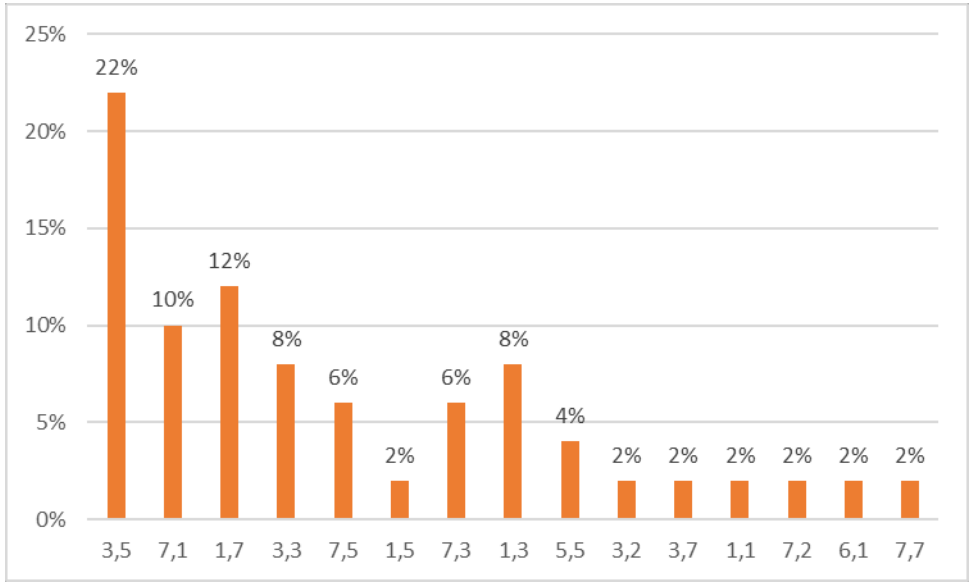
1 run



5 runs



50 runs



©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Multiplication Matrix for N=91 (= 13×7)

- $N = pq$, using binary representation with bits z, x, y respectively
- Binary multiplication shows (91 = 1011011)

1	0	1	1	0	1	1	pq=91
			x_1	x_2	x_3	1	p
			y_1	y_2	y_3	1	q
			x_1	x_2	x_3	1	
		y_3x_1	y_3x_2	y_3x_3	y_3	0	
	y_2x_1	y_2x_2	y_2x_3	y_2	0	0	
y_1x_1	y_1x_2	y_1x_3	y_1	0	0	0	

Virtually all optimizations improve the multiplication table somehow

e.g. rightmost bit means x_3 not y_3, so we can reduce this with $x_3 = 0$ and $y_3 = 1$

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Dridi, Alghassi refined this approach in 2016

- So-called “Gröbner” bases are used to reduce the degree of the Hamiltonian
- This pre-processing significantly reduces the size of the problem
- Their algorithm can factor all bi-primes up to 2×10^5 using a D-Wave 2X Processor
  - Main limitation is the number of qubits available
    - Dwave 2X has 1,100 qubits, however 5,600 qubit system will be available in 2020

- They were able to factor 200,099 with 897 qubits

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Jiang et al provided a massive breakthrough in 2018

Quantum Annealing for Prime Factorization

Shuxian Jiang[1], Keith A. Britt[2], Alexander J. McCaskey[2], Travis S. Humble[*2], and Sabre Kais[†1,3]

[1]Department of Computer Science, Purdue University, West Lafayette, IN 47906
[2]Quantum Computing Institute, Oak Ridge National Laboratory, Oak Ridge, TN 37831
[3]Department of Chemistry, Physics and Birck Nanotechnology Center, Purdue University, West Lafayette, IN 47906

Submitted April 2018

https://arxiv.org/pdf/1804.02733.pdf

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Jiang et al 2018

- General idea is the same
  - Formulate the factorization problem to a QUBO problem that runs on an adiabatic Quantum Annealer (D-Wave)

- Jiang et al proposed a new map which raised the record for a quantum factorized integer to 376,289 with just 94 qubits

- They successfully ran their algorithm on D-Wave's 2000Q Quantum Annealer

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Peng et al further optimized this in January 2019

Factoring larger integers with fewer qubits via quantum annealing with optimized parameters

WangChun Peng[1], BaoNan Wang[1], Feng Hu[1], YunJiang Wang[2], XianJin Fang[3], XingYuan Chen[4], and Chao Wang[1*]

[1] *Key Laboratory of Specialty Fiber Optics and Optical Access Networks, Joint International Research Laboratory of Specialty Fiber Optics and Advanced Communication, Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai 200444, China;*

[2] *State Key Laboratory of Integrated Services Networks (ISN), Xidian University, Xi'an 710071, China;*

[3] *School of Computer Science and Technology Engineering, Anhui University of Science and Technology, Huainan 232001, China;*

[4] *State Key Laboratory of Cryptology, Beijing 100094, China*

Received July 13, 2018; accepted September 27, 2018; published online January 21, 2019

http://engine.scichina.com/publisher/scp/journal/SCPMA/62/6/10.1007/s11433-018-9307-1?slug=fulltext

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Peng et al 2019

- General idea is the same
  - Formulate the factorization problem to a QUBO problem that runs on an adiabatic Quantum Annealer (D-Wave)
- Jiang et al could run a quantum integer factorization of 376,289 with just 94 qubits
- Peng et al optimize the problem Hamiltonian of Jiang's algorithm by reducing the number of qubits involved
  - They were able to factor 1,005,973 with just 89 qubits with an increased error tolerance as an added benefit.
  - This is now already a 20-bit number

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Peng et al 2019

- Approach is similar to Burgess's multiplication table.

Table 1 Multiplication table for $n=p\times q$

	2^{nLen}				…					2^2	2^1	2^0
p						1	p_{pLen-1}	…	p_2	p_1	1
q						1	q_{pLen-1}	…	q_2	q_1	1
						1	p_{pLen-1}	…	p_2	p_1	1
					q_1	$p_{pLen-1}q_1$	…	p_2q_1	p_1q_1	q_1	
Product terms				q_2	$p_{pLen-1}q_2$	…	p_2q_2	p_1q_2	q_2		
			⋰	⋰	…	⋰	⋰	⋰			
		q_{pLen-1}	$p_{pLen-1}q_{pLen-1}$	…	p_2q_{pLen-1}	p_1q_{pLen-1}	q_{pLen-1}				
	1	p_{pLen-1}	…	p_2	p_1	1					
T[a)]	n_{nLen}				…				n_2	n_1	1

a) T represents the target value.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Peng et al 2019

- Jiang optimized this by creating a modified multiplication table

Table 2 Modified multiplication table for 143=11×13, from ref. [33]

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
p					1	p_2	p_1	1
q					1	q_2	q_1	1
					1	p_2	p_1	1
Product terms				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
Carries		$c[2][2]$	$c[2][1]$	$c'[1][2]$	$c'[1][1]$[a)]			
Target values	1	0	0	0	1	1	1	1

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Peng et al 2019

- Jiang optimized this by creating a modified multiplication table

Table 2 Modified multiplication table for 143=11×13, from ref. [33]

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
p					1	p_2	p_1	1
q					1	q_2	q_1	1
					1	p_2	p_1	1
Product terms				q_1	p_2q_1	p_1q_1	q_1	
			q_2	p_2q_2	p_1q_2	q_2		
		1	p_2	p_1	1			
Carries		$c_{[2][2]}$	$c_{[2][1]}$	$c_{[1][2]}$	$c_{[1][1]}$[a)]			
Target values	1	0	0	0	1	1	1	1

- Peng et al removes the carry variables, thus achieving the reduction in complexity

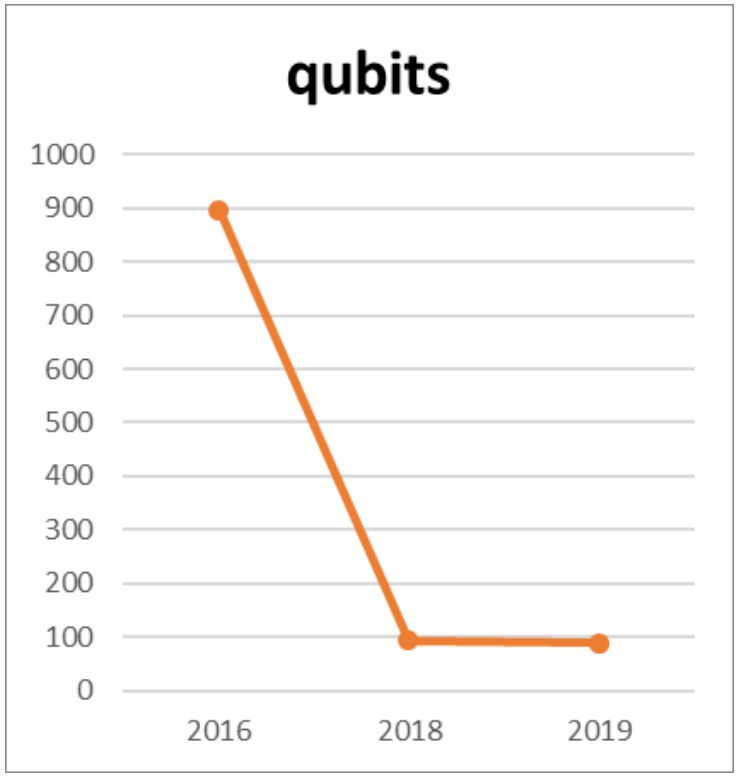
©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

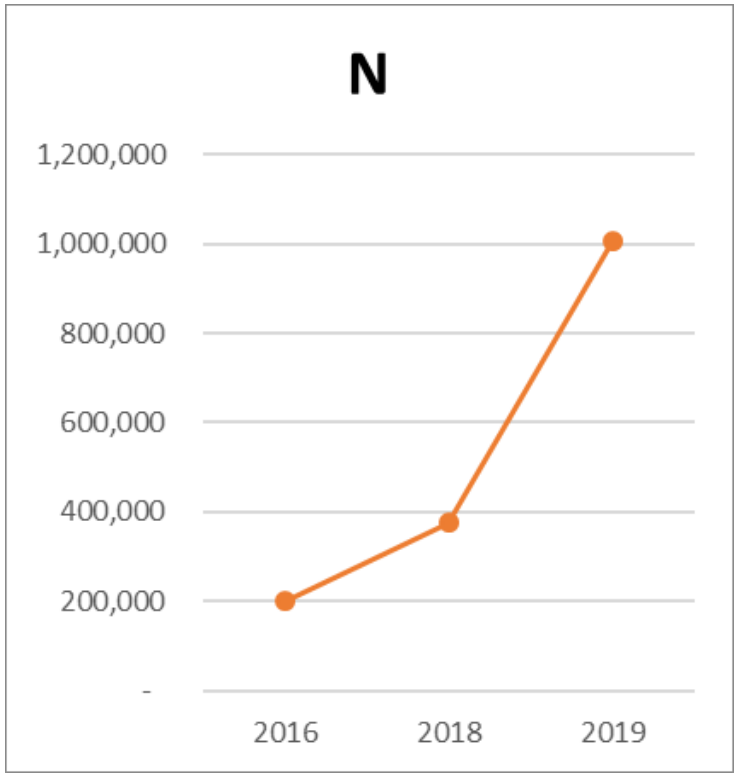
Peng et al 2019

- Three things were very interesting about their approach.
  - They were able to run this on currently available hardware
    - current quality of qubits is good enough to run this algorithm (unlike e.g. Shor's algorithm).
  - To factor an RSA-768 number (current factorization record on classical computers), their algorithm would "only" need 147,454 qubits.
    - D-Wave have announced a quantum computer with 5,640 qubits already, so the more qubits there are, the more vulnerable RSA will become.
  - Their algorithm uses a combination of quantum and classical computation to maximise the results.
    - interestingly that's the same for Shor's algorithm and a common approach. Use classical computers for what they are good at and quantum computers for what they are good at

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

year	qubits	number
2016	897	200,099
2018	94	376,289
2019	89	1,005,973





©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Conclusion



Conclusion

- While Shor's algorithm gets most of the PR attention, QAC is currently a thousand-fold better than UQC approaches
  - Both from the hardware (D-Wave systems have much more qubits)
  - As well from the research (massive optimizations in the last 3 years alone)

- QC's are way too noisy to be a threat anytime soon, but
  - QC's are getting better and better
  - Algorithms are being optimized heavily

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Conclusion

- Myth: Shor is currently the best-known algorithm to factor integers
- Reality: QA based algorithms are outperforming Shor by a factor of a thousand

- Myth: Shor's algorithm will eventually break cryptography
- Reality: Shor's algorithm was never meant to be implemented. Derivations of it will be used to break cryptography

- Myth: Today we have X qubits, Shor's algorithm needs Y qubits. Based on the last few years of qubit growth, it'll take Z years to break cryptography
- Reality: It'll be much quicker as you need to take the optimizations in the algorithms into account (e.g. from 1bn to 200m in just 7 years)

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Conclusion

- Myth: No need to worry as QC-based approaches are at least 10 years away
- Reality: That may or may not help you
  - Example: Satoshi's BTC coins have well-known public key. If I have a QC in 10 years time, these coins are mine and there is nothing anyone can do about it
  - We talk about over 1.1m BTC, which is currently around 12bn USD

- Myth: QC may well be 20 years away and not 10 years
- Reality: It all depends on breakthroughs in a) number of qubits, b) quality of qubits, c) quality of gate operation, d) optimizations in algorithms.
  - We've seen massive breakthroughs in all 4 areas over the last 6 years. It may be possible that we see none over the next 6 years, although I don't think so.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Conclusion

- Myth: We are safe because we use symmetric ciphers
- Reality: computationally secure ciphers are only as good as the currently known algorithms

Our work has focused on the practical and promising application of eventually breaking the RSA cryptosystem. In addition, quantum annealing-based algorithms are expected make increasing progress towards practical implementations of Secure Hash Algorithm (SHA) decryption and post-quantum cryptography by reducing the time required to break cryptosystems by an order of magnitude.

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61332019, 61572304, 61572034, and 61272096), and the Grant of the Special Zone Project of National Defense Innovation.

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.

Call to action

- Go out and play around with the available resources
- The feeling when you write your first quantum computer program and run it against a real QC hardware is just awesome :)

- Lots of resources to get you started

- Any questions: ab@quintessencelabs.com
- P.S. we are hiring :)

©2018 QuintessenceLabs. All rights reserved. Commercial in Confidence.



Thank you!

www.quintessencelabs.com