CAN QUANTUM COMPUTERS CHANGE HOW WE ENCRYPT DATA?

Supervised by Roger Hinds

Word count: 3996

Subject area: Computer Science

Name: Bertram Wheen

Candidate Number: 001451-037

Centre Number: 1451

Contents

Abstract

In the digital era we live in, where so much financial activity takes place online and personal details are sent through the air, digital security is increasingly necessary and high-profile. Quantum computers – seen as the next revolution in both science and computing – could have a huge impact on encryption. I find in this paper that quantum computing can change our approach to encryption and explain the background required to understand why this is so. Quantum computing will render leading encryption methods insecure (I cover some of the effects of Shor's algorithm), however it will also provide methods of quantum encryption, one of which I discuss. To source the information I largely use secondary literature.

Acknowledgements

I'd like to thank my supervisor, Roger Hinds, for guiding me through the process and giving invaluable feedback at sometimes short notice.

Name: Bertram Wheen
Candidate Number: 001451-037

Introduction

Many encryption methods today rely on certain tasks being inherently difficult. If these tasks were found to be not-so-difficult – i.e. a faster way of doing them was found – then these encryption methods would become void; an insecure encryption method has no use. Quantum computers will let us complete certain tasks quicker and some of these tasks are the same ones that much encryption relies on. Clearly, if we can break these methods by using quantum computers, then how encryption is done – how it remains secure – will have to be rethought. The same quirks of quantum mechanics that give quantum computers all their power are able to be used in quantum encryption, which could be used to replace our existing encryption methods.

Quantum Computers

Before we attempt to understand quantum computing (which we should do before discussing their potential impact) we must first know the model of computing in use currently.

In the early 20th century, Alan Turing, a British scientist, was working on the Entscheidungsproblem[1]. In the process of resolving it, he came up with the concept of the "universal Turing machine", which is arguably the basis for the computers we use today[2]. A *fixed* Turing machine (later I'll refer to this as a deterministic Turing machine) he defined (theoretically) as a contraption which 'is supplied with a "tape" ... running through it, and divided into sections (called "squares") each capable of bearing a "symbol"'[3]. The machine has a read/write head which is over one of the squares at any one time, and has the ability to erase the symbol on the "scanned square", print a new symbol there, or move the head's position on the tape left or right. What the machine will do next is determined by its fixed set of internal rules. These machines do what they do, say, calculate $\sqrt{2}$ and print it on the tape, but that's about the extent of it. You can't ask the machine just mentioned to now calculate π.

This is where the universal Turing machine comes in. Every Turing machine has a description number, which is the machine's m-configurations encoded to one number. The universal machine is supplied with a tape bearing the description number of some Turing machine, and proceeds to act as if it were the machine on tape. For instance, given the $\sqrt{2}$ computing machine, it would compute and print $\sqrt{2}$. So, the universal machine is programmable – it reads in a number, and alters what it does accordingly. Nowadays, computers are effectively based on this model; reading in bits and bytes of data and changing functionality accordingly. Instead of thinking of current computers as Dells or Macs and all their inner workings, we can consider Turing machines. This is

[1] A simple explanation of the problem: http://mathworld.wolfram.com/DecisionProblem.html
[2] DAVIS, M. 2001. *Engines of Logic: Mathematicians and the origin of the Computer.*
[3] TURING, A. 1936. *On Computable Numbers, with an application to the Entscheidungsproblem.*

because Turing machines are the simplest universal model of computation as "every … computation can be carried out by a Turing machine"[4] (this was shown in the Church-Turing thesis). This means that if we can calculate some function on our laptop, the function is also calculable on a Turing machine.

Quantum computers – proposed by Richard Feynman in 1982 – use some of the central ideas in quantum physics and are fundamentally different from 'normal' computers. Before we consider quantum computing's impact, we shall first cover some of the ideas that they are based on. This should serve as a taste of their complexity, but nothing more than that; even Feynman admitted "I think I can safely say that nobody understands quantum mechanics"[5].

Possibly the most important idea is the concept of things being in multiple states simultaneously; in a superposition of states. Austrian physicist Erwin Schrödinger illustrated this with his famous cat thought experiment:

A cat is penned up in a steel chamber, along with the following device (which must be secured against direct interference by the cat): in a Geiger counter, there is a tiny bit of radioactive substance, so small that perhaps in the course of the hour, one of the atoms decays, but also, with equal probability, perhaps none; if it happens, the counter tube discharges, and through a relay releases a hammer that shatters a small flask of hydrocyanic acid. If one has left this entire system to itself for an hour, one would say that the cat still lives if meanwhile no atom has decayed. The psi-function of the entire system would express this by having in it the living and dead cat (pardon the expression) mixed or smeared out in equal parts.[6]

[4] COPELAND, J. 2002. 4/11/12. http://plato.stanford.edu/entries/church-turing/
[5] FEYNMAN, R. 1964. 4/11/12. http://bouman.chem.georgetown.edu/general/feynman.html
[6] SCHRÖDINGER, E. 1935. *The Present Situation in Quantum Mechanics.* 4/11/12. http://www.tu-harburg.de/rzt/rzt/it/QM/cat.html

The idea is that until you measure the cat's state it could be either alive or dead, so it is viewed as both alive and dead simultaneously.

A classical computer deals in bits of data – each one representing a 1 or a 0. Quantum computers, however, use qubits. Qubits differ in that, while they also represent 1s or 0s, they can be in a superposition of all states – which means that they are, like Schrödinger's cat, both alive and dead; both 1 and 0 simultaneously.

Because of this, relatively few particles in superpositional states can represent a huge amount of data. For example:

A mere 1,000 particles can be in a superposition that represents every number from 1 to 2^{1000} (about 10^{300})[7]

Notice the difference here. 1,000 bits can represent *any* number from 1 to 2^{1000} (inclusive), but 1,000 qubits in a superposition can represent *every* number in the range. Similarly, Schrödinger's cat is representing every possible state (i.e. dead *and* alive), whereas a normal cat is representing any state (i.e. dead *or* alive). Many things can be used as qubits; people have used photons and electrons. With photons the direction of the light's polarization can be used to represent 1/0, with electrons their spin direction can be used.

Another phenomena of quantum physics is known as quantum entanglement. Referred to as "*spukhafte Fernwirkung" ("spooky action at a distance")* by Albert Einstein in 1947[8], quantum entanglement is when two particles (though it has been demonstrated with things as large as

[7] AARONSON, S. 2008. *The Limits of Quantum Computers.* 4/11/12. http://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf

[8] In a letter from Einstein to Max Born, 3 March 1947. Find this in *The Born-Einstein Letters; Correspondence between Albert Einstein and Max and Hedwig Born from 1916 to 1955*, Walker, New York, 1971

small diamonds[9]) interact and are then separated. The interaction must be such that both particles have the same indefinite quantum mechanical description (state). This means that the pair is entangled, and like superpositions, their state is indefinite until measured. The act of measurement of one particle immediately gives information about the other one, even if it is unmeasured. Dr R. A. Bertlmann is possibly best known for his socks:

Dr. Bertlmann likes to wear two socks of different colours. Which colour he will have on a given foot on a given day is quite unpredictable. But when you see that the first sock is pink you can be already sure that the second sock will not be pink. Observation of the first, and experience of Bertlmann, gives immediate information about the second.[10]

If these were quantum socks, then we can prove through experiment that neither of them has a colour – neither of them knows if they will be pink or not pink. But if you observe one, then it will randomly assume pink or not pink and the other one will immediately turn into the opposite.[11]

In quantum systems, the act of observation of particles changes them. So, to be able to "read" a qubit, you entangle it with another and read the latter (as they are linked).

There are many problems facing quantum computing; one such problem is decoherence. Decoherence basically means that quantum wave function collapses into one state; so for instance a qubit in superposition may 'collapse' to a 1. This isn't an impossible problem to overcome, however it is taking lots of research to get around. With these problems, at the time of

[9] LEE, K.C.; SPRAGUE, M.R.; SUSSMAN, B.J.; NUNN, J.; LANGFORD, N.K.; JIN, X.M.; CHAMPION, T.; MICHELBERGER, P.; REIM, K.F.; ENGLAND, D.; JAKSCH, D.; WALMSLEY, I.A. 2011. *Entangling Macroscopic Diamonds at Room Temperature*. 4/11/12. http://www.sciencemag.org/content/334/6060/1253.abstract
[10] BELL, J.S. 1987. *Speakable and Unspeakable in Quantum Mechanics: "Bertlmann's socks and the nature of reality"*
[11] Adapted from an interview with Anton Zeilinger
http://www.metacafe.com/watch/5705317/quantum_entanglement_and_teleportation_anton_zeilinger/

writing no-one has a sufficiently large scale computer to be of any use yet (however there have been many small-scale efforts[12] [13] [14]).

[12] SAR, T.; WANG, Z.H.; BLOK, M.S.; BERNIEN, H.; TAMINIAU, T.H.; TOYLI, D.M.; LIDAR, D.A.; AWSCHALOM, D.D.; HANSON, R.; DOBROVITSKI, V.V. 2012. *Decoherence-protected quantum gates for a hybrid solid-state spin register.* 4/11/12. http://www.nature.com/nature/journal/v484/n7392/full/nature10900.html
[13] PLA, J.J.; TAN, K.Y.; DEHOLLAIN, J.P.; LIM, W.H.; MORTON, J.J.L.; JAMIESON, D.N.; DZURAK, A.S.; MORELLO, A. 2012. *A single-atom electron spin qubit in silicon*. 4/11/12. http://www.nature.com/nature/journal/vaop/ncurrent/full/nature11449.html
[14] http://www.dwavesys.com/en/products-services.html

Computational Complexity Theory

Many encryption algorithms rely on problems being fundamentally hard. But how do you define a problem as being hard? Ask a computer scientist how difficult a problem is and you might get an answer claiming the problem to be in P. What are they talking about?

Computational complexity theory attempts to place computational problems into classes. These classes, such as P and NP contain problems of varying inherent difficulty.

But how do you classify how difficult a problem is? When classifying problems, we use the fastest[15] known algorithm. Our measure of complexity is how its time taken to solve the problem's worst-case scenarios grows with the size of the problem. Say the problem is to guess a number that has been randomly chosen between 1 and 10 inclusive, and your hypothetical algorithm is to guess each of the numbers in turn. In this case, the problem size – the range of numbers – is 10, and the worst-case number of guesses is 10 (if the correct number was the last one you guessed). If the number is now between 1 and 20, then both the size of the problem and the algorithm's worst-case performance are 20. Because the worst-case performance is clearly varying in proportion to the size of the problem, the algorithm would be said to have a BigO efficiency of O(n). O(…) is BigO notation and is used to express the worst-case order of growth of algorithms. Whatever is contained within the parentheses is the number of operations required to solve the problem, generally given in terms of n (with the exception of O(1), which is an algorithm that always completes in the same time regardless of size), where n is the size of the problem. In computer science, whether something's actually O(n) or O(5n + 17) is deemed effectively irrelevant; they'd both be given as O(n), as what we're interested is the shape of the algorithm's growth curve, rather than the finer details.

[15] Fastest in the sense of having the quickest worst-case time.

P is the class of problems for which an algorithm exists which can solve them in polynomial time using a deterministic Turing machine. An algorithm is said to run in polynomial time if it runs no slower than n raised to a fixed power; if it has a worst-case performance of $O(n^k)$ where k is some constant. So for instance an $O(n^2)$ algorithm runs in polynomial time. Problems in P are considered to be efficient, however problems with large exponents are unfeasible for all except very small problem sizes. For example, an algorithm that runs no slower than n^7 milliseconds given a data set of n elements would have a worst-case scenario length of time longer than the universe has been in existence given a data set of only 1000 elements[16].

The NP class holds problems for which possible solutions can be verified in polynomial time. The distinction here is that it's not a question of whether you can solve the problem, but whether, given a solution, you can verify its correctness. P is said to be a subset of NP, as all problems in P are also in NP. This is because if you can create a correct solution for the problem efficiently (i.e. in $O(n^k)$ time), it follows that you should be able to verify a solution efficiently as well.

The problems in the class NP-complete – another subset of NP – have no known efficient solutions. As it is in the NP class, solutions are however efficiently verifiable. NP-complete problems can be transformed to any other NP problem efficiently – this is the test for NP-completeness, and will come up again later. These play a crucial part in the $P = NP?$ problem, which is as yet un-resolved and one of the biggest open questions in computer science.

The $P = NP?$ (also known as P vs NP and $P =? NP$) problem, a solution for which the Clay Mathematics Institute is offering $1,000,000[17], asks whether all problems in NP can be solved efficiently and so are in P. If this is the case, then $P = NP$, otherwise $P \neq NP$. This is effectively asking whether there exists an algorithm that can solve an NP-complete problem efficiently.

[16] Estimates such as one in http://arxiv.org/abs/1001.4744 give the age of the universe to be roughly 4.3×10^{17}s, which is less than 1000^7ms = 10^{21}ms = 10^{18}s
[17] http://www.claymath.org/millennium/

NP-complete problems are hard. Not the kind of hard that means you scratch your head for half an hour and then do it on your computer. They are the kind of hard that means your computer can take billions of years to complete them. $P = NP?$ is asking if there's a better – as in, faster – way to solve them. Currently there is no known algorithm that can solve an NP-complete problem efficiently – i.e. in polynomial time. While most computer scientists suspect that there is no algorithm that will do this (and subsequently believe that $P \neq NP$)[18], it has not (at the time of writing) been proved that this is true. If an algorithm exists that will solve *any* NP-complete problem efficiently, then *all* problems in NP can also be solved efficiently – i.e. are in P.

This is because problems can be transformed into other problems. As we said before, the test for NP-completeness is the following: A problem is NP-complete if and only if it can be transformed to any other NP-problem efficiently. What this means is, given two problems A and B where A is in NP and B is NP-complete, we can solve A using an algorithm we have for B with at most a polynomial slowdown. So if we know an efficient algorithm for B, we can solve A efficiently also; it would take the sum of B's time to solve plus the slowdown, and summing two polynomials gives a polynomial. The key point here is that A can be *any problem in* NP, meaning that if we can find a single efficient algorithm for a single NP-complete problem, then we have an efficient algorithm for *every problem in* NP. So to prove that $P = NP$, all that is needed is one algorithm. It is much harder to prove that $P \neq NP$, because you have to prove that no such algorithm can exist; pointing out that even though scores of top mathematicians/computer scientists have been trying to find one for decades and none have been found does not constitute proof.

[18] GASARCH, W.I. 2002. *The P=?NP Poll*. 4/11/12. http://www.cs.umd.edu/~gasarch/papers/poll.pdf

Encryption

$P = NP$? asks if there exists efficient algorithms for "hard" problems. Quantum computers should provide a means to solving these "hard" problems with quantum algorithms. Many quantum algorithms have already been written, providing sometimes exponential speedups. The first example of this is from 1993, with an oracle problem which is exponentially faster on a quantum computer than on a classical one. In 1994 Peter Shor created Shor's algorithm, which may have huge effects on encryption (which we shall cover later). Shor's algorithm is used in factoring large integers, and provides an exponential speed up to the fastest known classical algorithm.

Encryption is the process of converting some data – also known as plaintext – into a form which should be unreadable to everyone. Everyone, that is, except those who are able to decrypt it. Decryption takes the encrypted data and turns it back into the original plaintext. Plaintext doesn't necessarily have to be words; it can also be sound, images and so on (really any sort of data; on computers the type of data is irrelevant as everything is stored as 1s and 0s). Encryption is done through applying an encryption algorithm to the data and decryption just uses the reverse of the encryption algorithm.

As an example, say we want to encrypt "Cat" and our encryption algorithm is to increment each letter ('A'=>'B', 'B'=>'C'…), then the encrypted version would be "Dbu". To decrypt it, we just decrement each letter; "Dbu"=>"Cat". Hopefully to someone who doesn't know your encryption algorithm, "Dbu" will appear to be gibberish.

Now, if someone wished to decrypt the "Dbu", they would need to know how it was encrypted – in this case by cycling the letters – to be able to do the reverse (and so to convert it back to "Cat"). However, there is a method of encryption whereby *even if they know the encryption method*, others may find decryption incredibly difficult. This method is known as public key encryption.

Encryption & decryption algorithms can modify their behaviour based on a number. For example, we could modify our earlier algorithm to use a key, whereby the key is some number, and that is the amount to change the letters by. So, where the plaintext is still “Cat”:

Key	Encrypted
1	“Dbu”
2	“Ecv”
3	“Fdw”

As should be clear, how the data is encrypted is different for each key. Now, even if someone (who we don’t want viewing the data) knows our encryption/decryption algorithms, they cannot decipher the encrypted information easily. Given “Fdw”, you would have to try various keys to successfully decrypt it – for example trying to decrypt with a key of 2 would give “Dbu”, which is no more decipherable than it was before.

In public key encryption methods there are 2 keys: An encryption key and a decryption key. The encryption key is – or can be – publicly distributed (hence the name of ‘public key’ encryption). The decryption key, however, must be kept private. The two keys are related such that the message – encrypted using the public key – can only be decrypted by the private key. However, as people may know the public key, their relationship must be one whereby, given the public key, it is impossible – or so complex as to render it *effectively* impossible – to find the private key. This is obviously necessary as people may know the (public) encryption key. There are various algorithms which generate good public/private key pairs which satisfy these conditions; the most popular one is RSA, which I shall discuss later.

Public key encryption is incredibly secure, as someone can know a lot about some encrypted information – the encryption method, the decryption method *and* the public encryption key – yet *still* be left unable to decrypt it. For this reason it is very widely used.

Effects on classical encryption

RSA – invented in 1977[19] [20] – is an algorithm used for generating (public and private) keys, as well as encryption/decryption. We won't cover the actual algorithm (it's quite complicated), however it will suffice to say that the product of two prime numbers is used to generate the public and private keys. The security of RSA relies on the fact that factoring this product back into the two primes is computationally hard; integer factorization is NP-complete. While there is some debate over whether breaking RSA is as difficult as factoring[21], it is no *more* difficult than factoring[22] and so if an efficient integer factorization algorithm is found, then RSA can be broken efficiently.

The fastest known 'classical' (i.e. non-quantum) algorithm for large integer factorization is the general number field sieve (abbreviated to GNFS), which runs in superpolynomial time; i.e. inefficiently. There is no known efficient (non-quantum) algorithm and so integer factorization is outside P. Clearly though it is in NP, as verifying a solution is easy; if the product of the factors is the original number (the one which was factorized) and the factors are prime (there are efficient algorithms for primality testing, so this is not a problem to check), then the solution is verified. The fact that it is in NP means that if $P = NP$, then we can solve it efficiently through transformation as we discussed before. However, judging by the decades of work done and general view that $P \neq NP$, it is unlikely that $P = NP$, meaning this is unlikely to break RSA.

What is likely to break RSA, however, is quantum computing. Shor's algorithm (mentioned earlier) factorizes large integers efficiently. A quantum computer running Shor's algorithm would be able to factorize RSA numbers very quickly, meaning it is possible to break RSA encryption. It isn't just

[19] It was independently invented about 3 years before, however this was kept secret until 1997.
[20] 1999. *Basic Concepts in Data Encryption: Key-Based Encryption.* 4/11/12. http://library.thinkquest.org/27158/concept2_4.html
[21] BONEH, D.; VENKATESAN, R. 1998. *Breaking RSA may not be equivalent to factoring*. 4/11/12. http://theory.stanford.edu/~dabo/abstracts/no_rsa_red.html
[22] GREGG, J.A. 2003. *On Factoring Integers and Evaluating Discrete Logarithms*. 4/11/12. http://wstein.org/projects/john_gregg_thesis.pdf

theory: people have successfully factorized numbers using Shor's algorithm[23] [24] [25] [26] [27], however the numbers factorized have been very small due to the small number of qubits.

Of course, it is *possible* to break RSA without quantum computers – RSA numbers (the products of two large primes which can be used for public/private key creation) have been broken in the past – it just requires huge effort. For example, RSA-768 was broken in 2009 through using many hundreds of machines for two years, 80 processors for half a year and more[28]. RSA-768 is a 232-digit RSA number (the product of two 116 digit primes) which, prior to it being factorized could be used for creating a public and private key. At the time of writing, RSA-768 is largest RSA number to be successfully factorized. RSA-1024 – the most commonly used RSA number – is 309 digits long and would be about 1000 times harder to factorize than RSA-768[29]. RSA-1024 is not, however, *impossible* to factorize, it's just that – unless computers advance a lot and there is another effort involving many hundreds of computers – it is completely unfeasible to do. It is unlikely such an event will occur; it is so widely used, people would not attempt to break it except with malicious intent (even then it would be difficult to convince lots of people to lend you their computers for a few years).

---

[23] VANDERSYPEN, L.M.K.; STEFFEN, M.; BREYTA, G.; YANNONI, C.S.; SHERWOOD, M.H.; CHUANG, I.L. 2001. *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*. 4/11/12. http://cryptome.org/shor-nature.pdf

[24] LU, C.; BROWNE, D.E.; YANG, T.; PAN, J. 2007. *Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits.* 4/11/12. http://prl.aps.org/abstract/PRL/v99/i25/e250504

[25] LANYON, B.P.; WEINHOLD, T.J.; LANGFORD, N.K.; BARBIERI, M.; JAMES, D.F.V.; GILCHRIST, A.; WHITE, A.G. 2007. *Experimental Demonstration of a Compiled Version of Shor's Algorithm with Quantum Entanglement.* 4/11/12. http://prl.aps.org/abstract/PRL/v99/i25/e250504

[26] LUCERO, E.; BARENDS, R.; CHEN, Y.; KELLY, J.; MARIANTONI, M.; MEGRANT, A.; O'MALLEY, P.; SANK, D.; VAINSENCHER, A.; WENNER, J.; WHITE, T.; YIN, Y.; CLELAND, A.N.; MARTINIS, J.M. 2012. *Computing prime factors with a Josephson phase qubit quantum processor.* 4/11/12. http://arxiv.org/pdf/1202.5707v1.pdf

[27] MARTÍN-LÓPEZ, E.; LAING, A.; LAWSON, T.; ALVAREZ, R.; ZHOU, X.; O'BRIEN, J.L. 2012. *Experimental realization of Shor's quantum factoring algorithm using qubit recycling.* 4/11/12. http://arxiv.org/pdf/1202.5707v1.pdf

[28] KLEINJUNG, T.; KAZUMARO, A.; FRANKE, J.; LENSTRA, A.K.; THOMÉ, E.; BOS, J.W.; GAUDRY, P.; KRUPPA, A.; MONTGOMERY, P.L.; OSVIK, D.A.; RIELE, H.; TIMOFEEV, A.; ZIMMERMANN, P. 2010. *Factorization of a 768-bit RSA modulus.* 4/11/12. http://eprint.iacr.org/2010/006.pdf

[29] Ibid.

Quantum encryption

So what's the point of quantum computers? If they're going to cause headaches for data security, why is so much effort being expended on them? Their main use is going to be in scientific simulations, where it is expected they will excel, making scientists lives easier and hopefully causing faster development as a society. Even encryption-wise, quantum computing isn't all bad news:

Quantum computing may break RSA encryption (and other encryption methods, such as those based on elliptic curves) however, instead of rendering encryption useless, it is more likely to simply mean we have to change how we encrypt data. As well as making various methods of encryption insecure, quantum mechanics provides a solution to its own problem: quantum encryption.

Quantum encryption is just encryption using quantum properties, such as entanglement. One example, proposed by Artur Ekert in 1991, is Quantum Key Distribution (QKD), which works like so: Say there are two people – Alice and Bob[30] – and Alice wishes to securely send a message to Bob. To do this, they must keep the key they will use for encrypting/decrypting the information secure. Alice creates a key (it can be anything, however for security against brute-force attacks[31] a very large, random number is preferable) and now wants to send it to Bob so they can start secure communication. Through entangling particles (the nitty-gritty details are outside the scope of this paper), Alice can not only send the key to Bob, but also detect a third-party (say, Eve) who somehow intercepts the transfer. This is because Eve, by measuring the transfer, disturbs the system (this is unique to quantum mechanics) and so they will know that the key is now insecure (known by others) and so can simply try again using a different key.

[30] http://www.networkworld.com/news/2005/020705widernetaliceandbob.html
[31] Where hackers use try each possible key in turn until they find the correct one

Conclusion

Quantum computing can, and moreover probably *will*, have a large impact on how we do encryption. However, the actual nature of the impact is dependent on various factors.

If it is shown that $P = NP$, then encryption methods that rely on NP problems being computationally hard (such as RSA) will already be broken. Quantum computing, therefore, may not have an effect on these encryption methods. It may (depending on the speed of the efficient classical algorithm discovered) provide a means to cracking them faster, however people will most likely have ceased use of these methods anyway if they are broken.

Quantum computing may not have an impact at all, as it is so incredibly difficult to create the computers that it may never happen – and if it never happens, then clearly it will have no impact. There are many problems such as decoherence (which was discussed earlier) left to overcome.

Quantum encryption may be too complicated for use to become common. It is much easier to encrypt data in the classical way, meaning methods such as the one-time pad (OTP), which works in a more 'normal' way, may be more widespread in use.

However, whether it happens or not, quantum computing has the potential to redefine how we do encryption, and for this reason I say yes: Quantum computers *can* change the way we encrypt data.

Bibliography

Format: Author(s), Year of publication, Title, Date accessed website, URL

DAVIS, M. 2001. *Engines of Logic: Mathematicians and the origin of the Computer.*

FEYNMAN, R. 1964. 4/11/12. http://bouman.chem.georgetown.edu/general/feynman.html

SCHRÖDINGER, E. 1935. *The Present Situation in Quantum Mechanics.* 4/11/12. http://www.tu-harburg.de/rzt/rzt/it/QM/cat.html

AARONSON, S. 2008. *The Limits of Quantum Computers.* 4/11/12. http://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf

LEE, K.C.; SPRAGUE, M.R.; SUSSMAN, B.J.; NUNN, J.; LANGFORD, N.K.; JIN, X.M.; CHAMPION, T.; MICHELBERGER, P.; REIM, K.F.; ENGLAND, D.; JAKSCH, D.; WALMSLEY, I.A. 2011. *Entangling Macroscopic Diamonds at Room Temperature*. 4/11/12. http://www.sciencemag.org/content/334/6060/1253.abstract

BELL, J.S. 1987. *Speakable and Unspeakable in Quantum Mechanics: "Bertlmann's socks and the nature of reality"*

SAR, T.; WANG, Z.H.; BLOK, M.S.; BERNIEN, H.; TAMINIAU, T.H.; TOYLI, D.M.; LIDAR, D.A.; AWSCHALOM, D.D.; HANSON, R.; DOBROVITSKI, V.V. 2012. *Decoherence-protected quantum gates for a hybrid solid-state spin register.* 4/11/12. http://www.nature.com/nature/journal/v484/n7392/full/nature10900.html

PLA, J.J.; TAN, K.Y.; DEHOLLAIN, J.P.; LIM, W.H.; MORTON, J.J.L.; JAMIESON, D.N.; DZURAK, A.S.; MORELLO, A. 2012. *A single-atom electron spin qubit in silicon*. 4/11/12. http://www.nature.com/nature/journal/vaop/ncurrent/full/nature11449.html

GASARCH, W.I. 2002. *The P=?NP Poll*. 4/11/12. http://www.cs.umd.edu/~gasarch/papers/poll.pdf

BONEH, D.; VENKATESAN, R. 1998. *Breaking RSA may not be equivalent to factoring*. 4/11/12. http://theory.stanford.edu/~dabo/abstracts/no_rsa_red.html

GREGG, J.A. 2003. *On Factoring Integers and Evaluating Discrete Logarithms*. 4/11/12. http://wstein.org/projects/john_gregg_thesis.pdf

VANDERSYPEN, L.M.K.; STEFFEN, M.; BREYTA, G.; YANNONI, C.S.; SHERWOOD, M.H.; CHUANG, I.L. 2001. *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*. 4/11/12. http://cryptome.org/shor-nature.pdf

LU, C.; BROWNE, D.E.; YANG, T.; PAN, J. 2007. *Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits.* 4/11/12. http://prl.aps.org/abstract/PRL/v99/i25/e250504

LANYON, B.P.; WEINHOLD, T.J.; LANGFORD, N.K.; BARBIERI, M.; JAMES, D.F.V.; GILCHRIST, A.; WHITE, A.G. 2007. *Experimental Demonstration of a Compiled Version of Shor's Algorithm with Quantum Entanglement.* 4/11/12. http://prl.aps.org/abstract/PRL/v99/i25/e250504

LUCERO, E.; BARENDS, R.; CHEN, Y.; KELLY, J.; MARIANTONI, M.; MEGRANT, A.; O'MALLEY, P.; SANK, D.; VAINSENCHER, A.; WENNER, J.; WHITE, T.; YIN, Y.; CLELAND, A.N.; MARTINIS, J.M. 2012.

Computing prime factors with a Josephson phase qubit quantum processor. 4/11/12. http://arxiv.org/pdf/1202.5707v1.pdf

MARTÍN-LÓPEZ, E.; LAING, A.; LAWSON, T.; ALVAREZ, R.; ZHOU, X.; O'BRIEN, J.L. 2012. *Experimental realization of Shor's quantum factoring algorithm using qubit recycling.* 4/11/12. http://arxiv.org/pdf/1202.5707v1.pdf

KLEINJUNG, T.; KAZUMARO, A.; FRANKE, J.; LENSTRA, A.K.; THOMÉ, E.; BOS, J.W.; GAUDRY, P.; KRUPPA, A.; MONTGOMERY, P.L.; OSVIK, D.A.; RIELE, H.; TIMOFEEV, A.; ZIMMERMANN, P. 2010. *Factorization of a 768-bit RSA modulus.* 4/11/12. http://eprint.iacr.org/2010/006.pdf