Quantum Computer Threats against PKI Data Security and a Digital-ID Based Self-Protecting-Data Solution

(A white Paper)

Version 2.3

By: Brad Nadji, Ph.D.
Dec 2023

Contents

Introduction

PKI infrastructure, also known as Public/Private Key Infrastructure, is the most commonly used technology for secure communication on the internet. Public key encryption is also known as asymmetric encryption. It is widely used, especially for the common protocols such as TLS/SSL, which make the secure internet protocol, HTTPS, possible.

> The history of the PKI invention goes back nearly half a century. Invented in the early 1970s at the British intelligence agency GCHQ, PKI has played a big role in the growth of security software and platforms we currently use. In 1982, **R**ivest, **S**hamir, and **A**dleman founded **RSA** Data Security Company to develop and market the PKI software. In 1990 RSA PKI software was adopted by the US Defense Department -- a move that National Security Agency opposed on the grounds that it "impacted national security." In 1993 the US government introduced Clipper Chip technology which allowed access to all coded voice and data transmissions. However, the leading electronic companies rejected its use in favor of RSA's PKI software. In 1994 RSA source code was published on the internet.

The PKI software and a variety of its implementations have withstood the test of time for several decades, and PKI has proven to be unbreakable using classical computers. However, recent advancements in the development of quantum computers and their threat to break PKI security have caused concerns about PKI's usability in the future.

In order to understand the reason that PKI is vulnerable to quantum computer attacks, it is best to go over a simple explanation of why and how PKI is used. Readers already familiar with the PKI exchange protocols can skip the coming section and move to the next section.

Basic PKI Usage Example

Imagine two entities on the internet who want to have a private and secure exchange. Let's call the two entities Alice and Bob, although for the most part Alice and Bob are really computers and not necessarily people.

Using Symmetric Key for Secure Communication

The most common way to communicate is using an *agreed-upon key* that Alice and Bob use to encrypt their messages on both ends. The same key is used for encrypting and decrypting; therefore, it is called a *symmetric* key:

- *Alice encrypts her message using the agreed-upon symmetric key and sends the encrypted message to Bob.*
- *Bob decrypts Alice's message using the same agreed-upon symmetric key that he holds. Bob can also send a message to Alice the same way, by encrypting it with the agreed-upon symmetric key.*

This looks like a simple, practical, and secure exchange. Except that it is not clear how Alice and Bob agree upon the same symmetric key and how do they exchange that key safely so they can communicate? In addition, what if Bob has thousands of people that need to communicate with him? How does he manage to have a secure symmetric key for each person that wants to communicate with him? He cannot use the same symmetric key to communicate with all others because all those people will have the same symmetric key and can read each other's messages.

The PKI key exchange protocol can help in this case and establish a secure symmetric key exchange.

Using PKI to Exchange a Symmetric Key

The PKI exchange is built upon the use of a Public key and its corresponding Private key. Each Public/Private key pair is generated at the same time, and the keys have a very important property:

> *Any message encrypted with the Public key can only (and only) be decrypted with its corresponding Private key.*

The public and Private keys are called asymmetric because they are different, and the same (public) key cannot be used for both encryption and decryption. Let's see how the Public/Private key property can be used by Alice and Bob, in a simple example, to exchange a symmetric key so they can communicate, as explained before.

- *Bob creates a Public/Private key pair. He keeps his Private key very secure and never shares it with anyone. He distributes the public key to everyone freely. The public key need not be protected and can be freely distributed; thus, the name public key.*
- *Alice has Bob's public key now. She can send encrypted messages using Bob's public key to encrypt the message. Only Bob can decrypt and read those messages because he is the only one holding the corresponding Private key.*
- *Alice creates a random key and calls it the symmetric key. She encrypts the symmetric key using Bob's public key and sends the encrypted package to Bob.*
- *Bob receives Alice's package. Only he can decrypt the package and if others intercept the message, they cannot decrypt the package and use it because they do not have Bob's Private key. Within the decrypted package, Bob finds the symmetric key that Alice generated.*

At this point, both Alice and Bob (and only them) have the same symmetric key. They can use the symmetric key to encrypt and decrypt messages they send to each other, as outlined before. PKI protocol has done its function and need not be used again, until the next session.

This is an oversimplified explanation of PKI and outlines one of the many possible ways it can be used. In addition, the role of Certificate Authority (CA) was not discussed here. CA is the authority that certifies Bob as the owner of the public key that Alice holds. This eliminates the possibility of the man-in-the-middle attacks by hijackers pretending to be Bob and providing their own public key to Alice.

Even in this simple exchange example, one can clearly see the value and elegance of PKI and the role it plays in internet communication. Every time you connect with a bank, email provider, internet commerce, or any other site that requires secure communication, you are most probably using PKI. PKI is built into every browser, and it is used by billions of users every day.

In order to see the quantum computer vulnerability of PKI, it is essential that we explain how the Public and Private keys are generated.

Generating Private and Public Keys in PKI

As seen in previous sections, the central principle in creating a PKI exchange is to generate a pair of interrelated Private and Public keys. That is done using the simple algorithm below. There are freely available software programs[1] on the web that have implemented this simple key generating algorithm. It is important to go through the mathematics of this algorithm so one can appreciate where its potential vulnerability lies:

1. Generate two large random primes, p and q of approximately equal size such that their product $n=p\times q$ is of the required bit length, e.g., 2048 bits (which is a number with about 617 digits). The number n is called a semi-prime because it has exactly two prime factors.

2. Calculate Euler's Totient Function, $\phi=(p-1)\times(q-1)$.

[1] See for example: https://cryptotools.net/rsagen

3. Choose an integer e between 1 and ϕ, $1<e<\phi$ such that it is relatively prime with respect to ϕ, that is the greatest common divisor of e and ϕ is 1, i.e., $gcd(e,\phi)=1$.

4. Compute the secret exponent d between 1 and ϕ, $1<d<\phi$, such that $e \times d \equiv 1 \mod \phi$. That means the remainder of the product $e \times d$ when divided by ϕ is 1.

5. The Public key is (n,e) and the Private key is (n,d) or (p,q,d). Keep all the values d, p, q and ϕ secret. Values n and e can be shared.

Now one can use the Public key (n,e) to encrypt messages that can *only* (and only) be read by someone who has the Private key (n,d).

Looking at the algorithm above, it appears that, anyone who has the Public key **(n,e)**, should be able to decompose the integer number **n** into its prime factors **p** and **q** (since **n=p*q**.) Therefore, one could eventually get all the components that are needed to create the corresponding Private key **(n,d)**. Having obtained the Private key in this way means that all communication that is using the corresponding Public key can be decrypted and exposed using the obtained Private key, resulting in the PKI breakdown.

So why is that not happening today?

Breaking the Keys

The fact is that the Public/Private key infrastructure of PKI can always be broken, but not in a reasonable amount of time, even by the fastest classical computers in the world. For example, let us analyze the following few attack vectors: the brute force attack method, and the prime factorization attack, which is of more interest to the KPI vulnerability discussion outlined above.

Using Brute Force

It is technically possible to use brute force to find the key needed to decrypt any generic encrypted message by trying all possible key combinations. But is it feasible?

Let us answer that using an example:

> Consider an example for brute forcing an encryption with a 256-bit key. Imagine that we have access to one of the fastest super computers in the world[2] that can perform at the speed of 1 ExaFLOPs. A 1-ExaFLOP computer can perform 10^{18} (1 quintillion) floating point operations per second. Let us also suppose that during a single FLOP, we can check if a given key is the secret encryption key for a given encrypted payload. In a 256-bit key encryption system, the number of possible key variations is 2^{256} or about 10^{77}. So, the number of operations that it takes to check every key (at one key check per FLOP) is 10^{77}. Our computer can perform 10^{18} operations every second. So, it takes our computer $10^{77} \div 10^{18} = 10^{59}$ seconds to go through all combinations. The average time is half the total time or about 0.5×10^{59} seconds which is approximately 1.8×10^{51} years! This is an enormous number with 52 digits. Given that the age of the universe is about 1.37

[2] At the time of this writing, the Frontier supercomputer at the Department of Energy's Oak Ridge National Laboratory is rated at 1.1 ExaFLOPs and is considered the fastest (known) computer in the world.

x 10^{10} years, the amount of time it takes to brute force a 256-bit encryption using a super computer is more than 10^{41} times the age of the universe!

In general, one can see the effect of key length on the approximate time it takes to brute force an encryption using a supercomputer[3] or a high-end home computer[4] in the following table. It is clear that encryptions with key lengths smaller than 128 bit are quite vulnerable to super computer attacks.

Key Length (bits)	Key Combinations	Approximate Time to Brute force	
		Super Computers	Home Computers
56	$7.2x10^{16}$	36 milliseconds	4 Days
64	$1.8x10^{19}$	9 seconds	2.9 years
128	$3.4x10^{38}$	10^{12} years	10^{20} years
192	$6.2x10^{57}$	10^{32} years	10^{39} years
256	$1.1x10^{77}$	10^{51} years	10^{58} years

Table 1, Brute-Forcing Encryption with Home Computers and Super Computers

So, it is safe to say that, even though technically possible, it is impractical to break a 256-bit key encryption using brute force.

Using Factorization

We saw in the previous sections that it is possible to break a PKI Private/Public key infrastructure by being able to factor the large integer number (**n**) that was used to create the Public and Private keys into its two large prime factors, **p** and **q** (**n=p*q**). The prime factorization problem is in the class of NP (nondeterministic polynomial time) problems and is known to run at an exponentially long time scale. That is, the solution-time exponentially increases as **n** is increased. This means that for a large **n** (with sufficiently large prime factors **p** and **q**) the prime factorization solutions can take an enormously long time using classical computers, not unlike the brute force approach.

However, the prime factorization problem can be solved relatively efficiently using quantum computers.

Quantum Computers and Prime Factorization Problem

Quantum computers are *not* faster versions of classical computers. They are specialized hardware devices that are also called computers unfortunately, causing a lot of confusion. They use quantum physical properties of sub-atomic particles to efficiently solve very specific problems that conform to quantum-algorithmic solutions.

Quantum algorithms are interesting as they might be able to solve certain classes of problems faster than classical algorithms because they can exploit *quantum superposition* and *quantum entanglement* of subatomic particles that are available on quantum computers and cannot be efficiently simulated on classical computers.

[3] At the time of this writing, the Frontier supercomputer at the Department of Energy's Oak Ridge National Laboratory is rated at 1.1 ExaFLOPs and is considered one of the fastest (known) computer in the world.
[4] As of this writing, a high-end home computer speed is generally less than 100 GFLOPs or 10^{11} FLOPs.

According to the US National Institute for Standards and Technology (NIST), while in the past it was less clear that large quantum computers are a physical possibility, many scientists now believe it to be merely a significant engineering challenge[5]. Some engineers even predict that within the next twenty or so years sufficiently large quantum computers will be built to break essentially all Public key schemes currently in use.

The best-known quantum algorithms designed to run on quantum computers are *Grover's algorithm* for searching and *Shor's algorithm* for factoring large numbers (which impacts PKI and is of interest to us).

Shor's Quantum Algorithm

Shor's algorithm is specifically interesting in cryptography, since it runs much faster (almost exponentially faster) than the best-known classical algorithm for prime factorization of large numbers on a quantum computer[6] and can solve the factorization problem in a much more manageable polynomial time.[7]

As discussed before, the solution to breaking a large number (n) into its large prime factors p and q can be used to break PKI. That means that if (or when) sufficiently large and error-free quantum computers become available, the security of PKI infrastructure is in jeopardy. That is the general fear that currently surrounds the internet commerce world as the majority of the crypto security of the internet is based on the PKI infrastructure.

Blockchains and Cryptocurrency Vulnerability to PKI

There is a misconception about blockchains that we first need to clarify:

> Blockchain is *not* a data security solution. It is *not* used to hide and obscure the data. Blockchain is in the class of data integrity solutions. The data within block chain is visible to everyone by design, and in fact, blockchain depends on data visibility to implement its decentralized publically traceable protocol.

Blockchain is used to immutably store a sequence (or chain) of transaction blocks and ensure the integrity of those transaction blocks. So, if a transaction block in the middle of the chain is altered, the altered transaction block and all others following it are invalidated. Eventually, through blockchain protocols, this contaminated chain is *voted out* in favor of other copies of the chain that are deemed *clean*.

> A blockchain ensures the *integrity* of transactions blocks within the chain; *it does not try to hide or obscure* the data within the chain.

---

[5] https://csrc.nist.gov/projects/post-quantum-cryptography

[6] Assuming existence of quantum computers with sufficient number of Qubits and without succumbing to quantum noise and other multi-qubit quantum-decoherence phenomena that adversely affect quantum computers' performance.

[7] The Shor's algorithm runs in polynomial time scale, with complexity estimated to be of the order of $O(\log^3(n))$.

Role of PKI in Blockchain Cryptocurrency Transactions

Blockchains generally use the SHA-256 hashing algorithm as the primary hash function to ensure the integrity of the transaction blocks as outlined above. However, certain application of blockchain, such as the ones used to implement Bitcoin or other cryptocurrencies also use PKI as part of their protocol to handle Bitcoin transactions.

In order to see the role of PKI in cryptocurrency transactions, we need to first look at the concept of a cryptocurrency *wallet*. Anyone who wishes to simply send and receive cryptocurrency needs a wallet. In our example, if Alice is sending cryptocurrency (e.g., a Bitcoin) to Bob, she is sending it to Bob's wallet. The protocol used to secure this transaction is generally the Public/Private key protocol. Bob has given his wallet's Public key to everyone including Alice, who wants to send cryptocurrency to Bob. Bob has the corresponding Private key securely in his possession. Alice encrypts the cryptocurrency transaction with Bob's Public key and sends that to Bob's wallet. Bob is the only one who can decrypt that transaction and receive the cryptocurrency because he has the corresponding Private key.

It is essential that Bob keeps his wallet's Private key secure. Anyone with access to Bob's Private key can open Bob's wallet and send its content to his own wallet -basically performing a digital version of pick-pocketing.

Moreover, if Bob loses his wallet's Private key, he can never access the content of his wallet and his wallet is basically lost forever. An estimated 20% of all Bitcoins currently in circulation, worth billions of dollars, are lost in digital wallets that owners can't access.

> To use a classical banking analogy to the cryptocurrency wallet transaction above, think about the following: Bob may give his bank account number and the bank routing number to anyone, including Alice, who wants to transfer money to Bob's account. That is the equivalent of Bob giving the Public key to his crypto wallet to anyone who wants to send cryptocurrency to Bob's wallet. On the other hand, Bob is the only person that can log into his bank account (equivalent of having the Private key) and check the received money and perform online banking.
>
> In previous sections, we saw that the vulnerability of the Public key factorization by quantum computers can provide the means to access the corresponding Private key. That is equivalent of someone who knows your bank account and bank routing number (Public key) finding a way to extract your bank account login and password from it (Private key)!

This is a major security concern for cryptocurrencies, like blockchain-based Bitcoin, that use PKI's Public/Private key concept to manage wallet transactions.

Possible Solutions to the Public Key Vulnerability problem

Motivated by the NIST's Federal Registry Notice[8], many scientific and research institutions mobilized to find a solution to the aforementioned Public key vulnerability. Technologies such as *Grid-based*

[8] "Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms" issued on December 2016. See: https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms

Cryptography, and *Multivariate Polynomial Equations*[9] solutions have been under study and development for the past few years.

In July 2022, NIST announced its selection of the CRYSTALS-Kyber algorithms for encryption. Kyber is a key encapsulation mechanism (KEM), whose security is based on the hardness of solving the learning-with-errors (LWE) problem over module lattices. While the standard is still in development, NIST encourages security experts to explore the new algorithms and consider how their applications will use them, but not to bake them into their systems yet, as the algorithms could change slightly before the standard is finalized. The process of Standardization and "baking" often takes many years. Even after standardizations, new products need to be developed based on the new standards and go through the certification process to ensure that they are reliable replacements for PKI which has been standardized, and certified for decades.

So how does one avoid the potential pitfall of Public keys in the future? The answer was given by a wise person who once said,

> "*The best way to solve a problem is to avoid the problem in the first place.*"

In other words, if Public keys are the source of the vulnerability, why not avoid using them altogether?

A Digital Identity Based Self-Protecting-Data Solution

Using digital IDs to create self-protecting data does not have the inherent Public key problems that were outlined in previous sections.

Generally speaking, the identity of an individual who is trying to get authorization to access a secure information system can usually be ascertained by one or more of the following general identifying attributes:

- What the individual knows (e.g., passwords, personal questions and challenges)
- What the individual has (e.g., access cards, access tokens)
- Who the individual is (e.g., fingerprint, facial recognition, retina scans)

A digital ID, can be defined as a secure bundle that contains digital equivalent of one or more of the above identifying attributes.

Any individual can have a (secure and self-protecting) digital ID. Alice and Bob in our example above can have their own secure digital ID which incorporates one or more of the above attributes. These IDs can be created using a local app, or by accessing a central digital ID ecosystem that allows creation and sharing of such secure digital IDs. The content of these IDs is highly secured; therefore, the individual digital IDs can be dispatched to potentially insecure environments over potentially insecure networks.

In order to utilize the digital IDs, let us imagine a Builder app and a Reader app.

- The **Builder app** is capable of taking insecure data, securing it through encryption using randomly generated encryption keys, and then incorporating the encrypted data, decryption

[9] https://eprint.iacr.org/2021/1476.pdf

keys, access policies, mitigation schemes and the digital IDs of the people who are allowed to access the data all within the same self-protecting data bundle. That data bundle is now fully secured and can be shared with anyone over potentially insecure networks via any transmission protocol.

- The **Reader app** has the knowledge needed to open the self-protecting data bundle and verify that the user who is trying to access the bundle satisfies all the appropriate identity challenges that are incorporated in their digital ID within the data bundle (as outlined above). When all identity challenges that were embedded within the secure data bundle are satisfied, access to the content (that is within the same self-protecting data bundle) is granted.

Therefore, Alice can incorporate Bob's digital ID in a self-protecting data bundle that she is building using the Builder and send it to Bob, knowing only Bob can read the content. The same is true for Bob sending information to Alice, where he would incorporate Alice's digital ID in a self-protecting data bundle that he builds using the Builder.

Note that Alice and Bob do not have to be actual human beings and could be computer systems. This means one can send information from computer system A to computer system B by incorporating computer system B's digital ID in the data bundle as the authorized receiver, ensuring that only computer system B can access the data within the self-protecting data bundle, even if the network security is compromised. Clearly, the digital ID of a computer system may be different than a human being's digital ID, but the concept is the same.

Advantages of Using Embedded Digital IDs for Access Management

Let us look at the advantages of using the digital ID self-protecting-data solution as outlined above:

- **Self-Protecting-Data**: The data bundle produced by the Builder app is autonomous, meaning that it is a self-managed and self-controlled intelligent data bundle. It can manage access controls, implement policies and mitigating actions when accessed by the Reader app. It does not rely on any third party or over-the-network central authority for access controls.
- **Safe to Share**: The data bundle produced by the Builder can be sent anywhere and shared freely with anyone knowing that no one can access the secure data inside, except those who have their digital IDs incorporated within the bundle.[10] That removes the fear of data "walking out" of a secure enterprise environment, by being "emailed out" or being "carried out" on a memory device. The data bundle secures itself and does not need protection from the environment.
- **Self-Tracking Data:** In a pure encryption scheme, anyone with the encryption key can access the data. There is no way of knowing who has accessed the data or if they have been successful. In the digital ID scenario, the identity of the person accessing the data is well known by their digital Identity. Also, the information on who, where and when the information within the secured data bundle was access is known by the Reader and can be easily reported back by the Reader to track the data bundle. This is critical for the systems that need to have irrefutable evidence and immutable proof of data access by others.

[10] Digital IDs can also be delegate stubs that will bind to a real digital ID at run time. That way, the digital identity of the recipients need not be known a priori. The details of this scheme is out of scope of this paper.

- **No key sharing:** There are no encryption keys (or *any* keys for that matter) exposed or shared. The keys protecting the data bundle are randomly generated at build time by the Builder and incorporated within the data bundle. The access is managed through digital IDs and not through encryption keys.
- **Quantum-attack safe:** In digital ID based approach, there is no need for Public/Private keys and key sharing. There are no keys to be shared; therefore, all the threats to KPI security caused by use of Public/Private keys and key sharing is circumvented.

Implementation and Products

One of the pioneering technology companies that has successfully implemented the concept of digital-ID based Self-Protecting-Data is **Sertainty Corporation**[11]. Their solution has been utilized in the market for over 5 years. Sertainty's patented self-protecting-data technology utilizes the digital ID and Self-Protecting Data concepts that we outline above; and therefore, is not suffering from the PKI vulnerability and quantum-computer threats. Their core solution is in the form of a Software Development Kit (SDK) that can be incorporated in any existing product, or used to build new products. Their solution does not enforce any business logic and therefore can be readily applied to any market segment such as, Government, Financial Services, Enterprise, IoT, Healthcare, Manufacturing, Transportation, Legal, Entertainment, Critical Infrastructures and others.

[11] https://www.sertainty.com