Public-key criptography and the issues it faces with the emergence of quantum computers

Marko Vejnović

March 30, 2018

1 Rationale

As someone who is interested in computers, criptography has always piqued my interests. Although we use some sort of encryption almost every time we access the internet, my first interaction with cryptographic algorithms was when using the *SSH* protocol.[1] The *OpenSSH* implementation can be configured to use different encryption algorithms, out which I found *RSA* [2] to be the most interesting one, as it is by far the most commonly used one. Exploring this algorithm sparked interest in me to further explore the field of cryptography, especially modern cryptography.

2 Introduction

Cryptography is the study of confidential communications in the presence of adversaries (Leeuwen, J. van.). The process of converting a readable message[3] into an undiscernable message[4] is called *enciphering*, whereas the process of converting an enciphered message into readable form is refered to as *deciphering* (ISO 7498-2:1989). The mathematical function which is used for enciphering and deciphering is called a *cipher* (Schneier, Bruce). Usually these operations require different functions. A message M is enciphered using the function E to a ciphertext C.

$$E(M) = C$$

This ciphertext is then deciphered using D back into M again.

$$D(C) = M$$

Cryptography requires that the following must be true, as the essence of ciphers is being able to recover the original plaintext (Schneier, Bruce):

$$D(E(M)) = M$$

It is worth noting that characters, symbols or other similar notions are represented with numbers in the majority of cryptogrpahic systems.

[1] The Secure Shell (SSH) Protocol is a protocol for secure remote login and other secure network services over an insecure network. (Internet Requests for Comments, 2006)

[2] Rivest-Shamir-Adleman

[3] Refered to as plaintext or cleartext.

[4] Refered to as ciphertext.

Historically, two types of ciphers existed - restricted ciphers and key ciphers. The former relied on adversaries not knowing the cipher itself. Restricted ciphers have almost completely vanished from use, as they present many issues - no quality control, no standardization, modern computers being able to crack such types of ciphers only being a few of them (Schneier, Bruce). The earliest of ciphers, such as the Atbaš cipher were, indeed, restricted ciphers. Key ciphers present themselves as a solution to these issues. Both enciphering and deciphering is done using a key:

$$E_k(M) = E(M, k) = C$$

$$D_k(C) = D(C, k) = M$$

For the majority of the history of cryptography, *symmetric ciphers* were employed. Symmetric ciphers are ciphers where the same key for enciphering and deciphering must be used.

$$E_k(M) = C$$

$$a \neq k \implies D_k(C) \neq M$$

One of the first noted uses of a symmetric key cipher was the *Caesar's cipher*. It, among other substitution ciphers, shifted a character in the alphabet to another by the key:

$$C = M + k$$

$$M = C - k$$

Other similar, more complex, systems have emerged throughout history, however, they all shared a common issue - they key was required to be transmitted in secrecy.

Until 1976, with the publication of *New Directions in Cryptography* by Whitfield Diffie and Martin E. Hellman, the key was necessary to be transmitted in secrecy[5]. The work of this paper resulted in an algorithm which allowed for transmitting a key in public, called the *Diffie-Hellman key exchange*. *New Directions in Cryptography* is the paper with which modern cryptography starts.

The Diffie-Hellman key exchange does however, have two issues. The first is that there is a need for a courier to actually transmit the key (although the courier does not know what the key is). The second issue is that the listening party has no way of knowing if a message sent was sent by the expected party.

Public-key ciphers present themselves as a solution to both of these issues. Public-key ciphers are ciphers which operate by using different keys for enciphering and deciphering.

$$C = E_{k1}(M) \tag{1}$$

$$M = D_{k2}(C) \tag{2}$$

These ciphers are called "public-key" because the enciphering can be done in public. The enciphering key is called the *public key*, whereas, the deciphering key is denoted to as the *private key*. Anyone is able to enipher a message using the public key, but only specific

[5]Actually, this paper was preceeded by James Ellis's work (1975), however, Ellis's work remained top secret in the *Goverment Communications Headquarters* until 1997, when it was fully released to the public (Sawer, Patrick).

people are able to decipher it with their private key. The most widely known, and most widely used public-key cipher is the *RSA*[6] cipher[7].

Both of these ciphers are based on modular arithmetic. For analyzing how well different algorithms perform, the notion of time complexity is used[8].

3 Modular arithmetic

3.1 Introduction

It was Gauss who gave the modern definition of modular arithmetic: If a number m divides the difference of the integers b and c[9], b and c are *congruent relative* to m, otherwise, they are *noncongruent*. The number m is defined as the *modulus* (Gauss, Carl Friedrich). If the numbers b and c are, indeed, congruent, they are called a *residue* of the other.

For a number a, all of its residues modulo m are contained in the formula

$$a + km$$

where k is any integer (Gauss, Carl Friedrich).

Gauss also introduced the modern congruence notation[10]:

$$a \equiv b \mod m$$

To exemplify the previous statements:

$$5 \equiv 28 \mod 23$$

5 is congruent to 28 modulo 23 means the following:

$$|28 - 5| = 23 \cdot k$$
$$\text{where } k \in \mathbb{Z}$$

[6]Rivest-Shamir-Adleman

[7]The paper *A Method For Obtaining Digital Signatures And Public-Key Cryptosystems* was published by Rivest, R. L. et al. in 1977.

[8]Time complexity is a measure of the time taken to perform a certain algorithm in terms of individual instructions. It is the number of instructions required $O(n)$ to perform on input whose size is n. To exemplify, a function which searches for an element in an array of length n takes n instructions to perform - for every element of the array, a check is done to see whether the element is the one that is searched for.

[9]These integers can be positive or negative, they are taken absolutely, unsigned.

[10]It is worth noting that Gauss in reality wrote the modulus in parenthesees

$$a \equiv b \pmod m$$

but for the purposes of this paper, I will use the more modern, also widely accepted, parenthesees-less notation.

This also means that the remainder of integer division between 28 and 23 is 5. Some textbooks explain modular arithmetic as an arithmetical system where values "wrap around" upon reaching the modulus value. This is often exemplified with the idea of a clock: the hour hand is congruent relative to modulo 12, the minute and second hands are congruent relative to 60.

3.2 Properties

4 The *Diffie-Hellman key exchange*

The *Diffie-Hellman key exchange*, as applied as it is today, is mathematically not complex. The name "key exchange" is a misnomer, however, as this algorithm does not actually exchange keys between communicators, rather, the communicators generate the same key.

Suppose that you have two people, a and b. They are in the presence of an adversary e. a and b have their private spaces in which they are able to store information safely, however, in order for them to communicate any piece of data, it must go through a public space which e has access to. The *Diffie-Hellman key exchange* works as follows. First, a and b agree on two shared constants, q and n. q is a relatively small prime integer, whereas n is a big integer[11]. These are communicated via the public space and e also knows these values. a and b then randomly choose a value x_a and x_b such that

$$0 < x < n$$

, and these values are kept private and not shared. Both of these people now calculate the value

$$Y = q^x \mod n$$

These result in Y_a and Y_b. Y_a and Y_b are then shared to the public space. a and b now calculate K_a and K_b as follows:

$$K_a \equiv Y_b^{x_a} \mod n \iff (q^{x_b} \mod n)^{x_a} \mod n \iff q^{x_b x_a} \mod n$$

$$K_b \equiv Y_a^{x_b} \mod n \iff (q^{x_a} \mod n)^{x_B} \mod n \iff q^{x_a x_b} \mod n$$

These values of K_a and K_b are equal and used as the common key for further symmetric enciphering. e is unable to calculate K without knowing either x_a or x_b.

In order for e to calculate an x he must perform the following calculation:

$$X = log_q Y \mod n$$

This issue is named the *discrete logarithm.*

It is precisely because of this why the *Diffie-Hellman key exchange* is cryptographically secure. The computation time required for calculating the discrete logarithm, with today's algorithms, at best is $\sqrt{n}$. The computational time required for calculating Y from x is at worst $2log_2 q$ (Diffie, W., and M. Hellman).

[11] Today, the value of n most commonly used is around 10^{1200}.

In figure 1, the relationship between the difference of time required to generate Y and the time to perform a discrete logarithm in order to calculate x, as performed by a simulation[12][13].

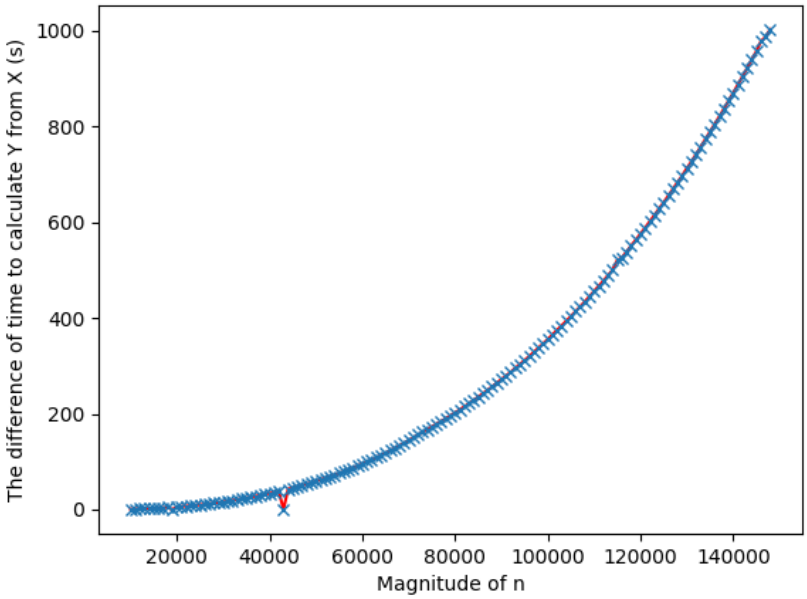

Figure 1: The relationship between the magnitude of n and the difference of time required to calculate X from Y and X using a *brute-force* X calculation algorithm.

5 The *RSA* cipher

5.1 Method

The *RSA cipher* operates as follows. The enciphering key is (e, n), where e and n are positive integers. The deciphering key is a pair of positive integers (d, n). The enciphering key is made public by all users, while the deciphering key is kept secret. Enciphering is done by raising the message to the power of e, modulus n. Deciphering is done by raising the ciphertext to the power d, modulus n.

$$E(M) \equiv M^e \mod n$$

$$D(C) \equiv C^d \mod n$$

[12]The simulation was written in *python* and the data was plotted using *matplotlib*. The exact code is published online at the following link: `https://github.com/markovejnovic/Cripto-IA`

[13]The outlier at is most likely due to incorrect time measurements by the system. Reference the original code for more information as to why this might've ocurred.

The enciphering keys are chosen as follows.

$$n = p \cdot q$$

where p and q are extremely large, random primes. n must be greater than the numerical representation of M. When n is made public, it is computationally difficult to discover p and q, as factorization of n is enormously difficult. The integer d is picked to be a large, random integer which satisfies the following:

$$gcd(d, (p-1) \cdot (q-1)) = 1$$

ie. it is relatively prime to $(p-1) \cdot (q-1)$. e is computed from p, q and d, as such:

$$e \cdot d \equiv 1 \mod (p-1) \cdot (q-1)$$

The proof why this method satisfies (1) and (2) is given in the following section.

5.2 Proof

The proof for why this cipher satisfies both (1) and (2) relies on the *Euler-Fermat totient theorem*:

$$M^{\phi(n)} \equiv 1 \mod n \tag{3}$$

where $\phi(n)$ is the *Euler totient function*. The *Euler totient function* gives the number of positive integers less than n which are relatively prime to it. For prime numbers this is always $n-1$ (as they are only divisible by 1 and themselves). Due to the multiplicative property of the *Euler totient function* [14], if $n = p \cdot q$ and p and q are prime:

$$\begin{aligned} \phi(n) &= \phi(p) \cdot \phi(q) \\ &= (p-1) \cdot (q-1) \\ &= pq - (p+q) + 1 \\ &= n - (p+q) + 1 \end{aligned} \tag{4}$$

Since d was chosen to be relatively prime to $(p-1) \cdot (q-1)$, ie. to $\phi(n)$, it does, indeed, have a multiplicative inverse in the finite field of modulo $\phi(n)$:

$$e \cdot d \equiv 1 \mod \phi(n) \tag{5}$$

Based on the previous deductions, the following can be asserted for the enciphering and deciphering functions E and D.

$$E(D(M)) \equiv D(M)^e \mod n \equiv M^{d^e} \mod n \iff M^{e \cdot d} \mod n$$

and

$$D(E(M)) \equiv E(M)^d \mod n \equiv M^{e^d} \mod n \iff M^{e \cdot d} \mod n$$

[14] The multiplicative property of the *Euler totient function* $\phi(ab) = \phi(a)\phi(b)$ is proved by the following: Let A, B and C be sets of nonnegative integers, which are relatively prime to a, b and ab, respectivelly, then there is a bijection mapping between $A \times B$ and C, trivially provable by the *Chinese remainder theorem*.

From equation 5, the following holds:

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n)+1} \mod n, \text{ for a positive integer } k$$

According to equation 3:

$$M^{\phi(p)} \equiv 1 \mod p \iff M^{p-1} \equiv 1 \mod p$$

Because $p - 1$ divides $\phi(n)$:

$$M^{k \cdot \phi(n)+1} \equiv M \mod p$$

Similary, for q:

$$M^{k \cdot \phi(n)+1} \equiv M \mod q$$

Finally, due the multiplicative property of the modulus:

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n)+1} \equiv M \mod n$$

For any positive integer M up to n, the previous statement, therefore, implies that E and D are inverse functions, therefore achieving the goal of being ciphers.

5.3 Cryptographic security

The cryptographic security of this cipher lies in the fact that up to date, no algorithm exists for factoring n with the same efficency as enciphering the message.

References

[1] Ylonen, T. *The Secure Shell (SSH) Protocol Architecture*. Internet Requests for Comments. Edited by Lonvick, C. The Internet Society, 2006.

[2] *sshd_config(5) Linux User's Manual*. 2006.

[3] Leeuwen, J. van. *Handbook Of Theoretical Computer Science*. Elsevier Science Publishers, 1990, p. 719.

[4] Schneier, Bruce. *Applied Cryptography, Second Edition: Protocols, Algorthms, And Source Code In C*. 2nd ed., John Wiley & Sons, Inc., 1996.

[5] *A Brief History of Cryptography*. http://cryptozine.blogspot.com/2008/05/brief-history-of-cryptography.html

[6] Sawer, Patrick. "*The Unsung Genius Who Secured Britain's Computer Defences and Paved the Way for Safe Online Shopping.*" The Telegraph, 11 Mar. 2016.

[7] Rivest, R. L. et al. "*A Method For Obtaining Digital Signatures And Public-Key Cryptosystems*". 1977, Accessed 27 Mar 2018.

[8] Gauss, Carl Friedrich. *Disquisitiones Arithmeticae*. Yale University, 1966.

[9] Diffie, W., and M. Hellman. "*New Directions in Cryptography.*" IEEE Transactions on Information Theory, vol. 22, no. 6, 1976, pp. 644654., doi:10.1109/tit.1976.1055638.

[10] Python Software Foundation. *Python Language Reference*, version 2.7. Available at `http://www.python.org`

[11] Hunter, J. D. *Matplotlib: A 2D graphics environment*. Computing In Science & Engineering, vol. 9, no. 3, 2007, pp. 90-95., doi:10.1109/MCSE.2007.55