A Comparison Between the Security of RSA and Elliptic Curve Cryptography

Tony Wu

Mathematics

Word count: 3780

Contents

1 Introduction

Cryptography is the practice of transmitting data privately, through encrypting and decrypting data. In essence, encryption involves hiding information in plain sight, so that only the recipient can read the contents. Decryption is retrieving the data from the encrypted message, called the ciphertext. In the modern world, encryption is crucial for privacy. Cryptography is used everywhere, from signing into accounts to sending emails, so that only the intended recipients can access the data. Without cryptography, there would be no privacy on the Internet.

Perhaps one of the oldest and best known cryptographic systems is the Caesar cipher, in which each letter is replaced with a letter that is shifted k places down the alphabet (Pass and Shelat 4). However, as there are only 26 letters in the alphabet, a person that does not know k, the key, could simply test all 26 possible shifts to obtain the original message. Thus, this is obviously not very secure in the modern world.

A modern development in cryptography is public key cryptography. In these systems, everyone has a public key and a private key, where anybody can see the public key and the private key is kept hidden. The basic principle in these systems is that the public key can be used to encrypt a message and only the corresponding private key can decrypt the ciphertext (Pass and Shelat 101). The specific algorithms that make this work is different for each cryptographic system, but in general, these systems have a central mathematical problem that makes them work.

RSA is one of the first public-key cryptographic systems that is widely used to encrypt on the Internet. Another cryptographic system, Elliptic curve cryptography (ECC) is more modern and is becoming more widely used. The research question is ***"To what extent does mathematics play in the construction of public-key cryptographic systems and how can we evaluate their security and efficiency?"*** In order to answer this question, we will compare RSA and Elliptic Curve Cryptography. In this paper, the mathematics behind why these cryptographic systems work, their requirements, and their security will be evaluated. Specifically, this paper will compare the mathematical problems that make these systems secure to figure out how the security of these cryptography systems can be generalized.

2 Public-Key Cryptography

All public key cryptography is based on “hard” problems, problems that cannot be solved efficiently with any known method. Cryptography uses these hard problems to create algorithms that are easy to compute in one direction, but hard in the other. We can call these problems trapdoor functions (Pass and Shelat 56). These ”hard” computations can be solved in theory with infinite computational resources, but in practice, in the real world, it is technologically infeasible. Thus, the security and privacy of people in the modern world is always a contest between increasing computational power and the strength of cryptographic systems.

In public key cryptography, if one wants to send a message to a person, they would use that person’s public key to encrypt the message. As this is used every time a message is sent, it is better if it is computationally easy to generate a ciphertext from a message. However, as the public key is available to everyone, the reverse direction, calculating the message from a ciphertext, necessarily needs to be difficult. This is the trapdoor function and the basis of most modern day cryptography. In addition, if a person has the private key, it should be computationally easy to obtain the message from the ciphertext, since they are the intended receipt of the message (Pass and Shelat 56).

We can see from this that as we make technological advances in computing, keys need to be larger and larger in order to have the same amount of security. This has downfalls as it takes longer to encrypt and decrypt with larger keys, but it is also more secure. We will now compare how this works in the context of different cryptographic systems.

3 RSA

RSA (Rivest-Shamir-Adleman) is an example of public-key cryptography and the first system we will examine. The central trapdoor problem that RSA cryptography is based on is the difficulty in factoring large numbers, specifically the product of 2 large primes. It is easy to multiply two numbers to get another number, but much harder to factor that large number back into its factors. RSA has the following algorithm (Rivest et al. 120):

- Choose 2 different prime numbers p and q and let $n = pq$
- Compute $\phi(n)$, the Euler totient function, which is equal to $(p-1)(q-1)$
- Choose an integer e with the following requirements
  - $1 < e < \phi(n)$
  - e and $\phi(n)$ are coprime, meaning that they do not share a common factor other than 1
  - e is often chosen to be $2^{16} + 1 = 65537$, which is a prime
- Compute $d \equiv e^{-1} \pmod{\phi(n)}$, the modular multiplicative inverse of e modulo $\phi(n)$
  - Modular multiplicative inverse means that $ed \equiv 1 \pmod{\phi(n)}$
  - This can be calculated efficiently with the Extended Euclidean Algorithm

The public key then becomes the pair (n, e) and the private key is the pair (n, d). We will let m be the message and c be the ciphertext (text can be converted to a number m using a variety of methods such as ASCII encoding).

3.1 Encryption

Encryption can be done by computing

$$m^e \equiv c \pmod{n}.$$

We are given m from the message and n and e from the public key, so we can calculate c easily using modular exponentiation. We can use the identity $ab \pmod{m} = (a \pmod{m})(b \pmod{m})$ and the fact that $m^e = \underbrace{m \cdot m \cdots m}_{e\ m\text{'s}}$. For example, to calculate $13^{16} \pmod{55}$, we could perform

the following steps

$$13^2 \equiv 169 \equiv 4 \pmod{55}$$

$$13^4 \equiv 4^2 \equiv 16 \pmod{55}$$

$$13^8 \equiv 16^2 \equiv 36 \pmod{55}$$

$$13^{16} \equiv 36^2 \equiv 31 \pmod{55}$$

where the previous steps are squared to get the next step. From this example, we notice that finding $m^{2^k} \pmod{n}$ requires k steps to calculate. We can prove this using induction.

Base case: $m^{2^1} \pmod{n}$ requires 1 step to calculate, since we can calculate $m^2 = m \cdot m$.

Inductive step: We assume $m^{2^h} \pmod{n}$ takes h steps. Then, $m^{2^{h+1}} \equiv (m^{2^h})^2 \pmod{n}$ takes $h + 1$ steps to calculate, since we just square the previous number.

Thus, by mathematical induction, $m^{2^k} \pmod{n}$ requires k steps to calculate. Using this, we see that $m^{2^{16}}$ takes 16 steps to calculate. Multiplying by m again to get $m^e = m^{2^{16}+1}$ takes one more step. This means that encryption only takes 17 steps if $e = 65537$, which can be done extremely quickly for a computer. In addition, since $e = 2^{16} + 1$ is a prime number, the condition that e and $\phi(n)$ are coprime is true for all n because primes cannot share factors other than 1 with other numbers.

3.2 Decryption

Once we get c, we must also have a way to decrypt it with the private key to get m again. We have $ed \equiv 1 \pmod{\phi(n)}$ from before. Rearranging yields

$$ed - 1 \equiv 0 \pmod{\phi(n)},$$

which means that $ed - 1$ is a multiple of $\phi(n)$, since the remainder is 0. By the definition from before, $\phi(n) = (p-1)(q-1)$. Since $ed - 1$ is a multiple of $(p-1)(q-1)$, we have

$$ed - 1 = r(p-1)(q-1),$$

for some integer r. If we let $a = r(q-1)$ and $b = r(p-1)$, we have

$$ed - 1 = a(p-1) = b(q-1),$$

for some integers a and b. We know that $m^e \equiv c \pmod{n}$ from the encryption algorithm. If we raise both sides to the power of d, we get

$$m^{ed} \equiv c^d \pmod{n}.$$

However, we see that we can rearrange the left hand side using the equation we found above to get

$$\begin{aligned} m^{ed} &= m \cdot m^{ed-1} \\ &= m \cdot m^{a(p-1)} = m \cdot (m^{p-1})^a \\ &= m \cdot m^{b(q-1)} = m \cdot (m^{q-1})^b. \end{aligned}$$

We can simplify this with Fermat's Little Theorem, which states that $a^{p-1} \equiv 1 \pmod{p}$ for integer a and prime p. Applying this gives

$$\begin{aligned} m^{ed} &\equiv m \cdot (m^{p-1})^a \equiv m \cdot 1^a \equiv m \pmod{p} \\ m^{ed} &\equiv m \cdot (m^{q-1})^b \equiv m \cdot 1^b \equiv m \pmod{q}. \end{aligned}$$

By the Chinese Remainder Theorem, since $pq = n$, we have $m^{ed} \equiv m \pmod{n}$. Since we have $m^{ed} \equiv c^d \pmod{n}$ from before, we can see that

$$c^d \equiv m \pmod{n}$$

which is how we can get m from c if we have the private key d.

3.3 Key Size

In RSA, the prime numbers p and q are private, but their product n is released to the public. From before, we see that the private key d, which is used to decrypt keys, can be efficiently calculated from $\phi(n) = (p-1)(q-1)$, the totient function. This means that the entire security of RSA

depends on how hard it is to factor $n = pq$. This is an open problem in mathematics and as time passes, faster factorization algorithms may be found. Currently, the fastest factorization algorithm is the General Number Field Sieve (Weisstein). The time complexity for this algorithm is

$$O\left(\exp\left(\left(\frac{64}{9}\right)^{1/3}(\log n)^{1/3}(\log\log n)^{2/3}\right)\right),$$

where $\log$ is the natural logarithm, $\exp(x)$ is the real exponential function e^x, and n is the number to be factored.

In layman terms, the expression inside $O()$ describes the average number of steps it will take to complete the factorization for the number n. For example, an algorithm with time complexity $O(n^2)$ and $n = 10$ would take $10^2 = 100$ steps to complete on average.

The National Institute of Standards and Technology recommends that RSA keys be at least 2048 bits long, meaning that $2^{2047} < n < 2^{2048}$ (Barker and Quynh 12). For reference, this means that n is 617 digits long. If we plug $n = 2^{2048}$ into the time complexity expression, we get $1.53 \cdot 10^{35}$ steps on average to break a 2048-bit RSA key. This is not practically feasible at all. We will consider a hypothetical supercomputer that can perform 1 quadrillion (10^{15}) steps per second. If we run this factoring algorithm, it would take an average of

$$\frac{1.53 \cdot 10^{35}}{10^{15} \cdot 60 \cdot 60 \cdot 24 \cdot 365.25} = 4.85 \cdot 10^{12}$$

years to complete or 4.85 trillion years. We can clearly see that this is not feasible whatsoever, so a brute-force factoring attack on a 2048-bit key is not possible. However, it is important to consider the size of this key. Larger keys make RSA more secure, but also less efficient at the same time, so it is important to have a balance between security and efficiency.

In order to compare security across different cryptographic systems, we can compare their bits of security, which determines the number of computations necessary to break the encryption. For example, a key that requires 2^{16} steps to break has 16-bit security. Taking the base 2 logarithm of the result from the time complexity expression yields the security strength in bits. Plugging in the time complexity equation into a calculator for several key sizes yields the following table.

RSA Security	
RSA Key Size	Strength
1024	87
2048	117
2538	128
4096	156
13547	256

The security strength can compare RSA to other systems, since it compares computational steps.

3.4 Decryption Efficiency

The decryption algorithm we found earlier is similar to the encryption algorithm, since they both require modular exponentiation. However, unlike the encryption algorithm, computing c^d modulo n is not as computationally efficient. We calculated earlier that the encryption algorithm always takes 17 iterations to compute. However, in the decryption algorithm, d is not always the same number and is significantly larger than e.

We can see that d has to be larger than e. If d were a small number, an attacker could keep multiplying c by itself modulo n in a brute force attack. If an attacker computes every c^k modulo n, eventually, $k = d$ and the resulting value would decode to an actual message instead of random values. Theoretically, there is no way to prevent this from happening. However, if d is a large value, it would be computational infeasible.

In the RSA algorithm, d is the modular multiplicative inverse of e modulo $\phi(n)$. From before, we know that $ed - 1$ is a multiple of $\phi(n)$. So, we have

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

Since p and q are both much smaller than n, $\phi(n) = n - p - q + 1$ should be about the same size as n. Since $ed - 1$ is a multiple of $\phi(n)$ and e is typically 65537, d is also about the same size as n. This makes going through all possible exponents in c^k not feasible. However, this also means that it is harder to calculate c^d modulo n.

We can use a similar algorithm as encryption to do modular exponentiation, where we repeatedly squared. However, we cannot always square since d is not in the form 2^{2^x}. Essentially, when we square a number, the exponent is multiplied by 2 ($x^k \to x^{2k}$) and when we multiply by the number, the exponent increments by 1 ($x^k \to x^{k+1}$). This uses the observation that

$$x^k = \begin{cases} (x^{k/2})^2 & \text{if } x \text{ is even} \\ x \cdot (x^{(k-1)/2})^2 & \text{if } x \text{ is odd} \end{cases}$$

If we repeatedly apply this formula, we can compute x^k modulo n easily. We consider the binary form of numbers to see an easy way to represent this. One way to convert from binary to decimal is to read the binary representation from left to right and follow this algorithm: if there is a 1, add 1 and multiply by 2, and if there is a 0, multiply by 2. We can do the same thing to calculate an exponent faster. For example, to compute x^{22}, we look at the binary form $22_{10} = 10110_2$.

$$\begin{aligned} 1: &\quad x^0 \to x \cdot x^0 = x^1 \\ 0: &\quad x^1 \to x^2 \\ 1: &\quad x^2 \to x \cdot x^4 = x^5 \\ 1: &\quad x^5 \to x \cdot x^{10} = x^{11} \\ 0: &\quad x^{11} \to x^{22} \end{aligned}$$

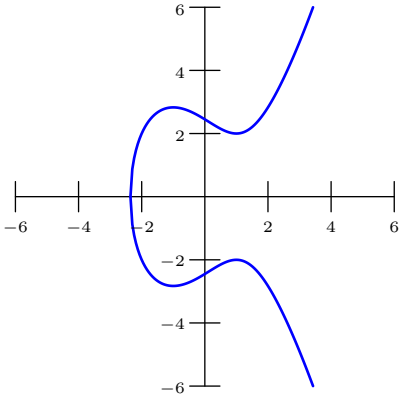
The number of steps to complete this is just the number of bits in the exponent minus 1, since the first step is always x. Since the decryption algorithm is $c^d \equiv m \pmod{n}$, the number of steps is equal to 1 less than the number of bits in d, which is approximately the same size as n. This means that decryption takes about 2047 steps for a 2048-bit key and increases based on the key size. This makes RSA inefficient.

4 Elliptic Curves

Elliptic Curve Cryptography (ECC) is another type of public-key cryptography. It is based on equations called elliptic curves, which have the form

$$y^2 = x^3 + ax + b,$$

where a and b are 2 constants (Silverman 7). Here is an example of an elliptic curve with equation $y^2 = x^3 - 3x + 6$.



The central trapdoor problem that ECC is based on is the discrete logarithm problem. This is analogous to normal logarithms in the real numbers that most people are familiar with: $\log_b a$ is the number x such that $b^x = a$. Discrete logarithms are similar in that $\log_b a$ is the integer x such that $xb = a$ over a group G, where the group operation is performed on b x times (Pass and Shelat 51).

A group is a mathematical set containing elements that are linked with an operation, called the group operation. Two elements can be combined using the group operation to form another element inside the group. An example of a group is the set of integers which is linked with the addition operation.

The trapdoor function in RSA is the ease of multiplying 2 numbers versus the difficulty in factoring the product. On the other hand, the trapdoor function in ECC is the ease in performing a group operation multiple times on an element to get another element versus the difficulty of finding the original element from the new element.

ECC differs from RSA in that it does not directly encrypt a message; there is no message or ciphertext. Instead, ECC generates a shared secret, which 2 parties can than use to securely communicate using another cryptographic method.

4.1 Domain Parameters

Elliptic Curve Cryptography has a much more complicated algorithm than RSA, with numerous restraints and potential weaknesses. Unlike RSA, ECC has additional parameters other than the public and the private key. The same elliptic curve can be used by multiple people, so the National Institute of Standards and Technology publishes standard elliptic curves. ECC has the following domain parameters (NIST 87):

- Generate 2 constants a and b, where $4a^3 + 27b^2 \neq 0$
- Select a prime p to be the modulo
  - All operations are done modulo p
  - Informally, this can be visualized as the curve wrapping around to the other side once it reaches the maximum
- Choose an integer point on the curve G to be the base point
- Let n be the order of G, the smallest integer such that $nG = \sigma$, the identity element
  - nG is the group operation performed on G n times
  - n is also the number of elements in the subgroup that contains G
- Let h be the number of elements in the curve divided by n (this is usually 1, 2, or 4)

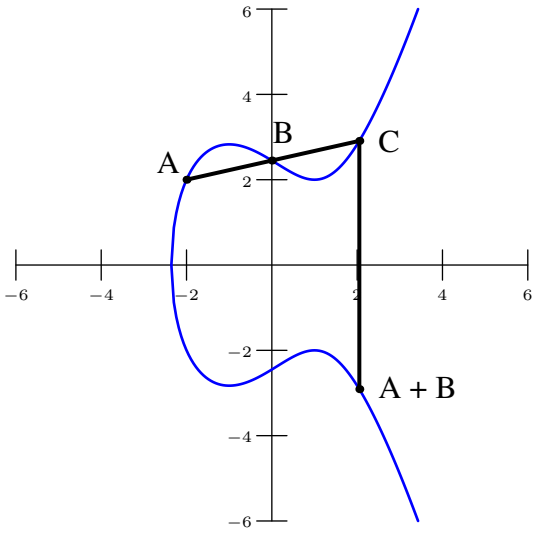
4.2 Group Operation

Elliptic Curve Cryptography operates over the group of integer points that satisfy the elliptic curve

$$y^2 \equiv x^3 + ax + b \pmod{p},$$

where a and b are integers and p is a prime. The group operation in ECC, +, can be defined in this way: draw a line through points A and B, find the intersection of that line with the elliptic curve at another point other than A and B, and let $A + B$ be that point reflected over the x-axis (Novotney 2).

Since the y in the equation is squared, we see that elliptic curves are symmetric over the x-axis, meaning that the point $A + B$ is also on the elliptic curve. We can visualize the group operation like so:



If we want to do $A + A$, we can use the tangent line to the elliptic curve at A as the initial line. As we noted above, elliptic curves are symmetric over the x-axis, meaning that if A and B are two points with the same x-coordinate and opposite y-coordinates, the line between them will be vertical and will never intersect the curve again. In this case, we define an identity element σ, where $A + \sigma = A$. We will also define nA as taking the group operation on A n times, so $nA = \underbrace{A + A + \cdots + A}_{n\ A\text{'s}}$.

4.3 Shared Secret Key

Elliptic Curve Cryptography is also a public-key cryptographic system, so it has a private and a public key. In order to perform encryption, two parties need to agree on the same domain parameters, which will probably be chosen from one of the standard curves. The private key is a random

number d, where $0 < d < n$, and the public key is a point $P = dG$, the group operation performed on the base point d times.

Let's say that person A has private key d_A and public key $P_A = d_A G$. Similarly, person B has private key d_B and public key $P_B = d_B G$. We see that person A can compute

$$d_A P_B = d_A d_B G$$

and person B can compute

$$d_B P_A = d_B d_A G.$$

These 2 values are the same and this can be called the secret key. Unless an attacker can compute d from $P = dG$, only person A and person B know this secret key, which can then be used to communicate securely. The problem of finding d from P is the discrete logarithm problem and the basis of Elliptic Curve Cryptography. It is computationally easy to compute P from d, but it is much harder to get d from P.

There needs to be an efficient way to calculate $P = dG$ other than simply performing the group operation on G k times. Otherwise, calculating the public key would take just as long as an attacker repeatedly performing the group operation on the base point until they find somebody's public key. We realize that this is very similar to the modular exponentiation part of decryption for RSA. However, instead of multiplying and exponentiating, we will perform the analogous taking one additional group operation and "squaring," which is basically performing the group operation on the point and itself.

Earlier, we computed x^{22}. Using a similar method, we will compute $22x$, where x is a point. From before, we have the base 2 form of 22: $22_{10} = 10110_2$. Reading from left to right, when there is a 1, we can perform the group operation of x and the square of itself. When there is a 0, we can take the "square."

$$
\begin{aligned}
1: &\quad \sigma \to x + \sigma + \sigma = x \\
0: &\quad x \to x + x = 2x \\
1: &\quad 2x \to x + 2x + 2x = 5x \\
1: &\quad 5x \to x + 5x + 5x = 11x \\
0: &\quad 11x \to 11x + 11x = 22x
\end{aligned}
$$

Thus, the number of steps to find dG is the number of bits in d minus 1, since the first step is always G. This can also be represented by $\lceil \log_2 d \rceil - 1$ steps.

4.4 Key Size

The security of Elliptic Curve Cryptography is based on how hard it is to find d from dG, the discrete logarithm problem. Similar to RSA, this is also an open problem in mathematics and faster algorithms may be found as time progresses. Currently, all of the fastest discrete logarithm algorithms, such as Pollard's rho, run with time complexity

$$O(\sqrt{n}),$$

where n is the order of G and the number of points in the subgroup (Pollard 331).

In this case, we can very clearly see the correlation between security strength in bits and key size in bits. To find the security strength in bits, we can plug in $n = 2^b$, so n has b bits. We then take the base 2 logarithm of the time complexity which gives us

$$\log_2 \sqrt{2^b} = \frac{1}{2} \log_2 2^b = \frac{1}{2} b.$$

So, the security in bits is half the key size in bits. To obtain 128-bits of security, we would need a 256-bit key, which is much less than in RSA, where we need 2538 bits. We compare the key sizes required in RSA and ECC to have the same security strength below.

Key Size Comparison		
Strength	RSA Key Size	ECC Key Size
87	1024	174
117	2048	234
128	2538	256
156	4096	312
256	13547	512

Another interesting point to note is that RSA's key size increases faster compared to ECC's key size as the security strength increases. From 128 to 256 bits of security, RSA's key size increases by a factor of $\frac{13547}{2538} = 5.34$, whereas ECC's key size increases by a factor of 2. We can also deduce this from the time complexity formulas. We can see that $\sqrt{n}$ increases slower than $O\left(\exp\left(\left(\frac{64}{9}\right)^{1/3}(\log n)^{1/3}(\log\log n)^{2/3}\right)\right)$, since the latter is an exponential function.

4.5 Security Issues

With RSA cryptography, each person generates their own unique primes and there are relatively few constraints, meaning that people have control over their own security. For ECC, the same elliptic curve can be used for multiple different people. Due to the relative difficulty in generating suitable elliptic curves and in order to create a standard for safety and efficiency, NIST publishes their own recommended elliptic curves for cryptography (NIST 87).

Unlike with generating one's own domain values, using a pregenerated curve requires trust that the curve was not built in with a backdoor that would enable a malicious actor to break all encryptions made with that curve. In 2006, NIST published the Dual Elliptic Curve Deterministic Random Bit Generator. However, this was later shown to potentially include a backdoor from the National Security Agency, which would break all encryptions that used this number generator (Shumow and Ferguson 4). This historical example reveals the real possibility of weak elliptic curves being purposely published, showing one of the weaknesses of ECC.

5 Conclusion

Understanding the security and the mathematics behind RSA Cryptography and Elliptic Curve Cryptography is crucial as these mechanisms secure the Internet in the modern world. RSA is the first major public-key cryptographic systems and is widely in use today, However, compared to others, it is slow and inefficient as it requires a larger key size for the same amount of security.

Elliptic Curve Cryptography, a newer and more complex system, is starting to become more widely used. In this paper, we analyzed the key sizes required to obtain the same amount of security and found that ECC required much smaller keys to be equally secure. However, there are downsides to ECC. Due to the fact that one needs to rely on others to generate the elliptic curve means that the security lies outside one's control, which raises security problems. In addition, it's complexity makes it more obscure and more prone to errors in implementation as it is less understood than RSA. While RSA concerns itself with multiplication and factoring, a problem people learn in elementary school, Elliptic Curve Cryptography is based on the discrete logarithm problem.

As all public key cryptography is based on a difficult problem, if there are more advances in mathematics that produce faster and more efficient algorithms to solve these trapdoor problems that the cryptographic systems are based on, they could be broken. If in the future, an algorithm is invented that solves either of these problems extremely efficiently, encryption around the world would be broken, which would have widespread ramifications. Thus, it is crucial to study these systems and understand how they work.

A future consideration that will become more prominent in the future is quantum computing. There are already quantum algorithms that can solve both the integer factorization problem and the discrete logarithm problem efficiently (Bäumer 3). The only barrier to this right now is the limitations in quantum technology that make these algorithms unusable. In the future, we may need to consider newer crytographic systems to secure the Internet. In addition, as computing power increases, key sizes need to be larger and larger in order to keep the same level of security.

6 Works Cited

Barker, Elaine, and Quynh Dang. "Recommendation for Key Management." National Institute of Standards and Technology, National Institute of Standards and Technology, Jan. 2015, nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf.

Bäumer, Elisa, et al. "Shor's Algorithm." Quatum Device Lab, 15 May 2015.

"Digital Signature Standard (DSS)." Federal Information Processing Standards, National Institute of Standards and Technology, July 2013, nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

The History of Cryptography, Stanford University, cs.stanford.edu/people/eroberts/courses/soco/projects/public-key-cryptography/history.html.

Novotney, Peter. Weak Curves In Elliptic Curve Cryptography. Mar. 2010, wstein.org/edu/2010/414/projects/novotney.pdf.

Pass, Rafael, and Abhi Shelat. "A Course in Cryptography." Department of Computer Science, Cornell University, 2010, www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf.

Pollard, J. M. "A Monte Carlo Method for Factorization." Bit, vol. 15, no. 3, 1975, pp. 331–334., doi:10.1007/bf01933667.

Rivest, R.L., Shamir, A., and Adleman, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM, Vol 21, No. 2, February 1978

Shumow, Dan, and Niels Ferguson. "On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng." Crypto 2007 Rump Session, International Association for Cryptologic Research, Aug. 2007, rump2007.cr.yp.to/15-shumow.pdf.

Silverman, Joseph H. "An Introduction to the Theory of Elliptic Curves." Department of Mathematics | Brown University. Summer School on Computational Number Theory and Applications to Cryptography, 2020, University of Wyoming, www.math.brown.edu/∼jhs/ Presentations/WyomingEllipticCurve.pdf.

Weisstein, Eric W. "Number Field Sieve." MathWorld, Wolfram, mathworld.wolfram.com/ NumberFieldSieve.html.