HIGH SCHOOL EDITION
Journal of Student Research

Analysis on the Efficiency and Security of the RSA and ECDH Key Exchange Algorithms

Charles Wang

Millburn High School, USA

ABSTRACT

The Rivest-Shamir-Adleman (RSA) and Elliptic Curve Diffie-Hellman (ECDH) Key Exchange algorithms are examples of trapdoor algorithms, which use mathematical complexity to provide real life internet security. In this paper we will look at the efficiency and security considerations of the RSA and ECDH Key Exchange algorithms. These algorithms were coded in Java on Replit and ran with different message sizes and key sizes to compare their efficiency based on encryption time. Furthermore, the main attacks on these cryptosystems, the Quadratic Sieve algorithm for the RSA and the Baby Step Giant Step algorithm for the ECDH, were coded, and the run times of the attack algorithms were compared to determine the relative security. Regression analysis was performed to determine the curves of best fit to approximate the graphs of message sizes and key sizes compared to the runtime. According to industry standard, the ECDH has the advantage over the RSA for the same level of security given a smaller key size required. At the same 112-bit strength, the ECDH key size is 224 bits, and the RSA key size is 2028 bits. Based on the extrapolated data to this industry standard, my results showed that encryption for the ECDH was faster than the RSA but did not confirm the comparable security as hacking the RSA was slower. The Quadratic Sieve codes could be improved for faster attack. In the future, the development of quantum computers will necessitate quantum-resistant algorithms as the Shor's Algorithm can solve current problems faster.

Introduction

Public key cryptosystems use pairs of keys that are generated through various algorithms. One key of the pair, labeled the private key, must be kept secret, while the other key, the public key, can be openly distributed without reducing the security of the system. Anyone with this public key can encrypt a message that can be sent to the person, but only the person with the private key can decode the encrypted message. The algorithms that generate the keys are called trapdoor functions. Trapdoor functions are functions that are easy to compute in one direction, but the inverse of the function is asymptotically harder to compute.

An example of a trapdoor function is the prime factorization problem. It is trivial for any computer to calculate the product of two primes, while it is difficult for a computer to determine the two primes that are factors of a prime product. While there are more efficient methods of calculating the prime factorization of a prime product than trial division from the smallest prime up, these methods are still slow compared to the speed at which a computer can multiply two primes (Boneh 1999).

Another trapdoor function is based on the Elliptic Curve Discrete Logarithm problem. An elliptic curve is a function based on the graph of $y^2 = x^3 + ax + b$ where a and b are constants. To “add” points on the elliptic curve, the chord-tangent method is used. To illustrate, to add points P and Q, draw the line PQ and find the coordinates of the third intersection of PQ and the elliptic curve. Then flip this point over the x-axis and the resulting point is defined as the sum of P and Q. To find $P + P$, draw the tangent to the elliptic curve at point P, and find the point of the third intersection between this line and the elliptic curve. Finally, flip this point over the x-axis and the resulting point is defined as the sum of P and P. $P + P$ is also defined as 2P, so multiplication is the same as repeated addition. Furthermore, each elliptic curve also contains a point at infinity, where the sum of any point with infinity is the point itself, and the sum of two points with the same x-coordinate is the point at infinity. All these operations are also computed modulo p, where p is a prime (McAndrew 2016).

This trapdoor function is based on the fact that multiplying a point by a constant is fast for a computer, while if a computer is given a point P and a point Q, it is hard for it to find out what multiple of P is equal to Q. In other words, if $Q = n * p$, a computer given point Q and P will take a significantly longer time to find the constant n (Barker et al., 2017).

Public key cryptosystems use trapdoor functions to provide the keys, so that it is easy to generate the keys, but someone without the private key cannot decrypt a message easily even if they have all the public information.

All computer algorithms have a certain time complexity which is related to how long it takes for the computer to complete the algorithm. It is based on the number of operations, assuming each operation takes the exact same amount of time. Most of the time, this reflects the worst-case scenario, the maximum amount of time it would take for a computer to run the program. An example of a time complexity is O(n) or O(log n) where n stands for the size of the input into the function as the input gets large. A polynomial time complexity is a time complexity where the time it takes the computer to run an algorithm is correlated with a polynomial of (log n). Such an algorithm would be considered fast. Most encryption protocols are polynomial time or faster, while all attacks on current encryption algorithms are slower than polynomial time. As the size of the inputs gets larger, this difference between polynomial and non-polynomial time complexities becomes much bigger, leading to much faster encryption than attack.

As a result, current cryptographic algorithms are efficient because they have a faster time complexity than the time complexity of different attacks on the algorithms. For modern cryptographic industries, the key sizes used in the algorithms are several hundred digits long to prevent quick attacks. Although this leads to slightly longer encryption time, the attack time on these algorithms becomes much longer.

However, recent advances in quantum computers would lead to some of these cryptosystems being less secure. Shor's algorithm solves both the elliptic curve discrete logarithm problem and the prime factorization problem, so the algorithms that depend on these specific trapdoor problems will no longer be secure in the future (Kee 2021). Researchers are working to develop quantum-secure algorithms that are as efficient as current-day algorithms. Lattice-based cryptography holds a great promise for replacing existing algorithms in the post-quantum world. Three of the four National Institute of Standards and Technology (NIST) finalist cryptosystems are lattice-based cryptographic systems.

Background

Public key cryptosystems are currently used in industry to send messages between different people as well as storing information securely. These cryptosystems are often used to generate shared secret keys between two parties to enable more efficient secret-key cryptosystems such as the Advanced Encryption Standard (AES). An example of using public key cryptosystems is when users need to interact with a bank website through information transmission to withdraw or deposit money. Banks often use a combination of these public key cryptosystems, such as ECDH and RSA, with the AES to protect data. Current updates in the uses or attacks of these cryptosystems are critical to reduce vulnerability to any special attacks. The necessity for a strong encryption means that key sizes in these public key cryptosystems must be very large, as small key sizes allow for much quicker attack. Most key sizes in public key cryptosystems are hundreds to thousands of bits long.

The problem of prime factorization is one of the oldest problems in mathematics, dating back to the Sieve of Eratosthenes in ancient Greece. Prime factorization is the process of turning a large number into a product of prime integers, and this is considered a trapdoor problem. It is incredibly easy to multiply two numbers, but the factorization of a prime product is not efficient to compute. There are many different prime factorization methods that are faster than trial division, dividing every single prime from two up. Some prime factorization methods are special-purpose methods, while others are general-purpose methods. For example, Fermat's factorization is a special-purpose method, which factorizes a large odd prime product effectively. It relies on the fact that any large, odd number N can be written as $N = a^2 - b^2$. Fermat's factorization tests values for a and tries to find the smallest a and b values to factorize N. Once a and b are found, N can be written as $N = (a - b) * (a + b)$, factors of N. However, this relies on the fact that N can be easily written as the difference of two squares, otherwise, Fermat's factorization is just as slow as trial division.

A general-purpose factorization method that is faster than most other prime factorization algorithms is the Quadratic Sieve. This method is the fastest algorithm to factorize numbers less than one hundred digits long and is the second fastest algorithm for numbers greater than one hundred digits. This algorithm is an improvement on Fermat's factorization, where instead of only looking for squares that perfectly subtract to get the large number, it looks for a congruence of squares modulo a certain number, allowing for more cases to show up. Furthermore, the congruence of squares usually is not trivial and leads to a quick factorization. Once a congruence of squares $a^2 \equiv b^2 (modulo\ N)$ is obtained, the expression becomes $(a - b) * (a + b) \equiv 0\ (modulo\ N)$. Then, taking the greatest common divisor of either $(a + b)$ and N or $(a - b)$ and N, a factor of N is obtained if it is not either one or N (Pomerance 2008). Despite significant improvements, it is still slow and cannot factorize several hundred-digit numbers in any reasonable amount of time.

The RSA algorithm uses the difficulty of the prime factorization problem to encrypt data efficiently. To begin, a user must generate two very large prime numbers, p and q. Then, compute $N = pq$, which serves as the modulus for both private and public key calculations. Next, compute the number $A = (p - 1)(q - 1)$, and generate a random number e such that e and A are coprime. Finally, generate the number d such that d is the multiplicative inverse of e with respect to A. N and e will serve as the public key, while everything else will be kept secret. For someone who has the public key to encrypt their message m, they will simply compute $m^e\ (modulo\ N)$, and send this value to the person with the private key. The person with the private key will compute what they receive to the d power or find $m^{e^d} (modulo\ N)$. This returns the original message m, so the person with the private key can decrypt the encrypted version of the message (Rivest et al., 1978). To attack the RSA, someone must be able to factor N, which, as mentioned above, is an extremely difficult problem to complete efficiently. Other methods exist to attack the RSA, but when the RSA is run with high key sizes that are not special values, they are either not practical or do not function (Boneh 1999).

The ECDH (Elliptic Curve Diffie Hellman) Key Exchange is a public key cryptosystem that is based on the difficulty of the elliptic curve discrete logarithm problem. The algorithm starts between two people, Alice and Bob, with a shared starting point P and a shared elliptic curve $y^2 = x^3 + ax + b$ and a modulo N. Alice and Bob must also have their private keys, so let Alice have a private key a, and Bob have a private key b, where a and b are large integers. Alice will compute aP and send this value to Bob. Bob will compute bP and send this value to Alice. When Alice gets Bob's value, she will then compute a(bP), and this will be her version of the shared key. When Bob gets Alice's value, he will then compute b(aP), and this will be his version of the shared key. Multiplication is associative in elliptic curves, so a(bP) is equivalent to b(aP) and both Alice and Bob have a shared key to send secure messages between them (Barker et al., 2017).

There are many different attacks on the ECDH algorithm, but most of these have the main problem of finding the order of a point on the elliptic curve with a prime modulo. This is equivalent to asking, given a point P, what is the number x other than one such that $P \equiv xP$ on the elliptic curve.

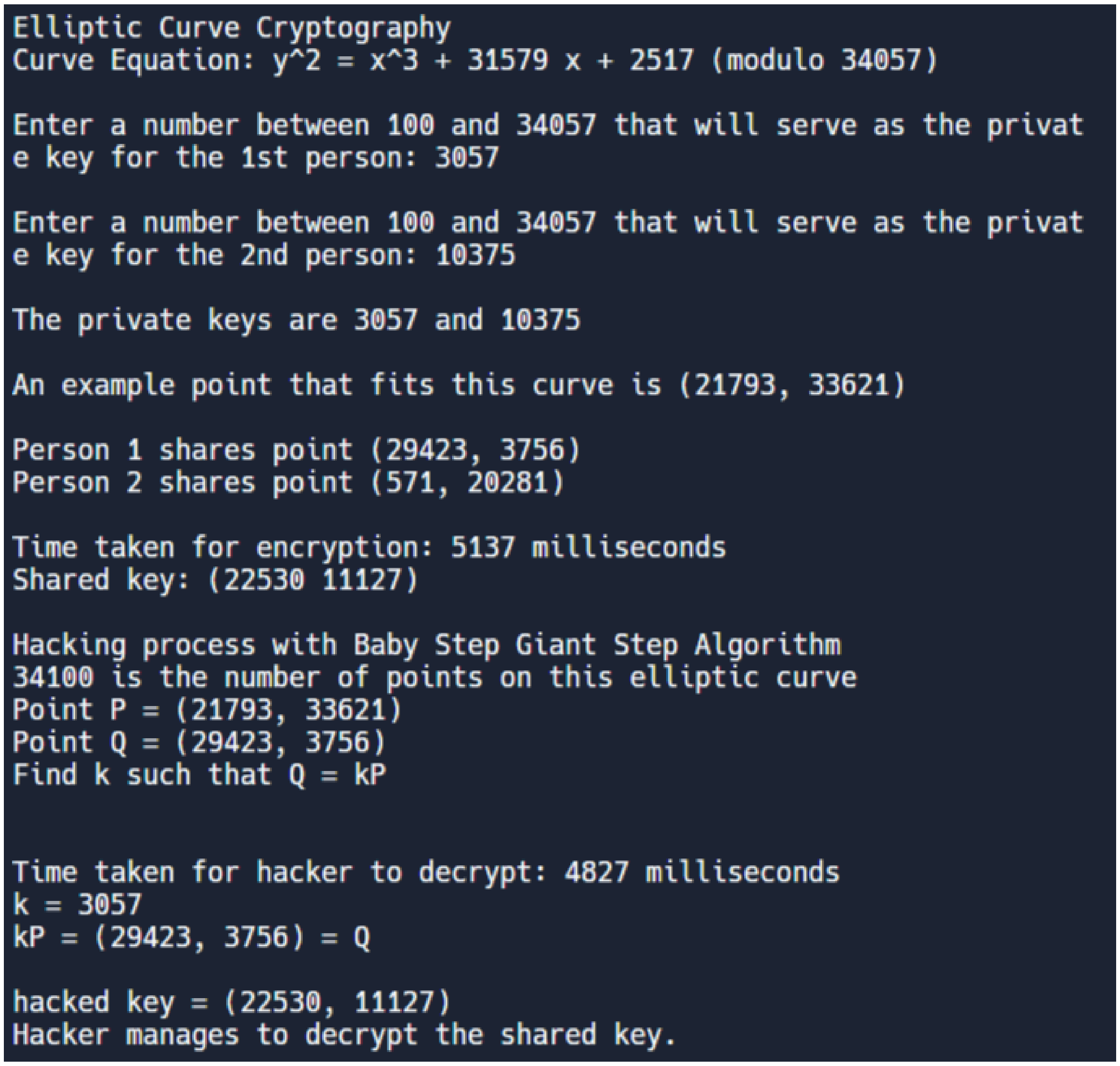
One major attack on the ECDH is the Baby-Step Giant-Step Method, which has an order of approximately O($\sqrt{N}$) where N is the prime order of the elliptic curve point. To start off, compute the value m such that $m = \lfloor\sqrt{N}\rfloor$ and compute the point mP on the elliptic curve. Then, compute a list of iP where $0 \leq i \leq m$ and store the points in a list. Next, compute $Q - jmp$ where $j = 0, 1, 2, 3, \ldots$, until one of these points matches a point in the list of iP. If the value $iP = Q - jmp$, then save the value $i + jm$, because $(i + jm)P = Q$ (Aung et al., 2017). This solves the elliptic curve discrete logarithm problem, as it allows a hacker to find one of the private keys that either Alice or Bob has and the shared private key. Finally, the hacker can use the shared private key to read all encrypted messages that Alice and Bob send. However, this method is still slow compared with encryption, so hacking remains difficult.

In comparison between the ECDH and the RSA, it is known that the ECDH has key sizes that are much smaller, but still provides the same level of security. The RSA is a much older algorithm, that some claim should be retired due to the multitude of new attacks that are constantly being developed against it. Furthermore, the ECDH algorithm is much more adaptable, given many variants such as using elliptic curves in higher dimensions. On the other hand, the RSA algorithm does not have many variations since it is based on the singular hard problem of prime factorization, and is more vulnerable to special attacks, resulting in the necessity of much bigger key sizes. Although the RSA is more simplistic and easier to use, the ECDH is more efficient with smaller key sizes, allowing for faster encryption and decryption time, while providing equal or greater security.

Methods

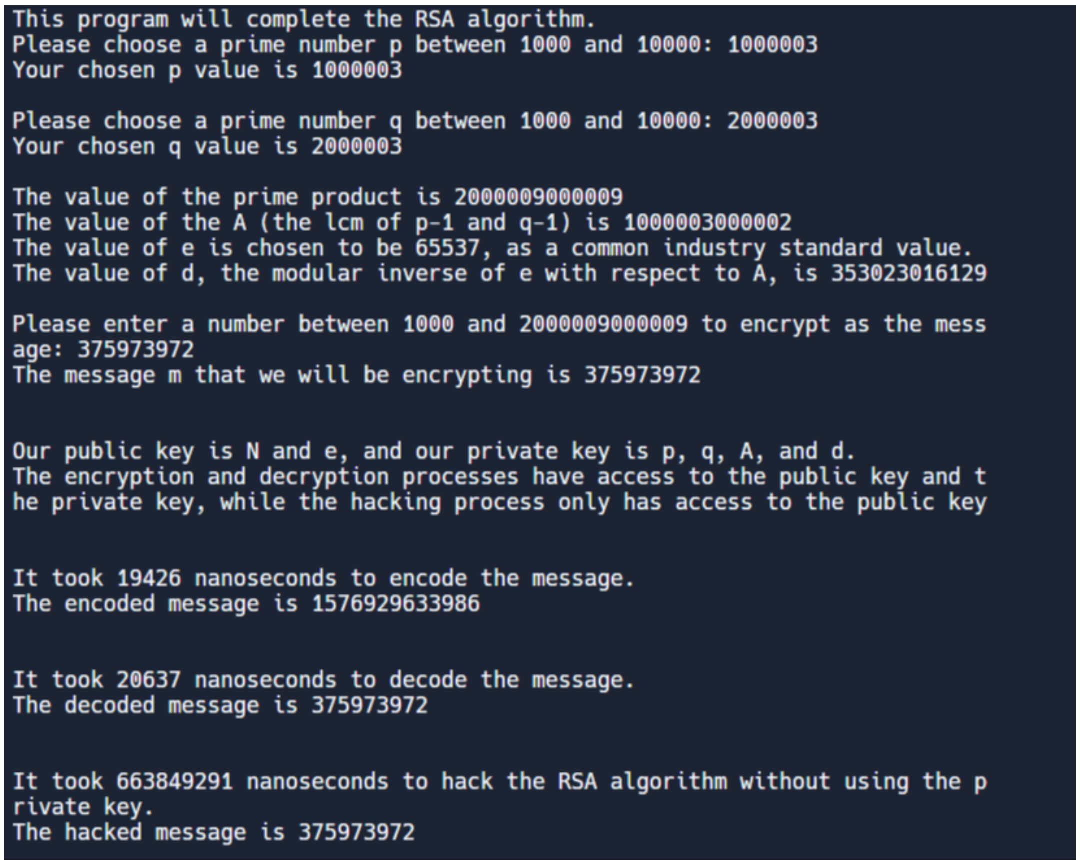
Both the RSA and ECDH Key Exchange algorithms as well as their respective attack methods, the Quadratic Sieve and the Baby Step Giant Step (BSGS) algorithms were coded in the programming language Java. These Java programs were coded on replit.com, an online integrated development environment.

Each program was run one hundred times to collect completion time versus message size for the RSA and the Quadratic Sieve algorithms and run time versus sum of private keys for the ECDH and the BSGS algorithms. Regression analysis was performed, and data were extrapolated to the industry standard. The industry standard key size is approximately 2^{2048} for the RSA and 2^{224} for the ECDH. The run example is shown in Figure.1 for the ECDH and BSGS and in Figure.2 for the RSA and Quadratic Sieve.



```
Elliptic Curve Cryptography
Curve Equation: y^2 = x^3 + 31579 x + 2517 (modulo 34057)

Enter a number between 100 and 34057 that will serve as the privat
e key for the 1st person: 3057

Enter a number between 100 and 34057 that will serve as the privat
e key for the 2nd person: 10375

The private keys are 3057 and 10375

An example point that fits this curve is (21793, 33621)

Person 1 shares point (29423, 3756)
Person 2 shares point (571, 20281)

Time taken for encryption: 5137 milliseconds
Shared key: (22530 11127)

Hacking process with Baby Step Giant Step Algorithm
34100 is the number of points on this elliptic curve
Point P = (21793, 33621)
Point Q = (29423, 3756)
Find k such that Q = kP


Time taken for hacker to decrypt: 4827 milliseconds
k = 3057
kP = (29423, 3756) = Q

hacked key = (22530, 11127)
Hacker manages to decrypt the shared key.
```

Figure 1. Online Replit run example of the ECDH and BSGS algorithms



```
This program will complete the RSA algorithm.
Please choose a prime number p between 1000 and 10000: 1000003
Your chosen p value is 1000003

Please choose a prime number q between 1000 and 10000: 2000003
Your chosen q value is 2000003

The value of the prime product is 2000009000009
The value of the A (the lcm of p-1 and q-1) is 1000003000002
The value of e is chosen to be 65537, as a common industry standard value.
The value of d, the modular inverse of e with respect to A, is 353023016129

Please enter a number between 1000 and 2000009000009 to encrypt as the mess
age: 375973972
The message m that we will be encrypting is 375973972


Our public key is N and e, and our private key is p, q, A, and d.
The encryption and decryption processes have access to the public key and t
he private key, while the hacking process only has access to the public key


It took 19426 nanoseconds to encode the message.
The encoded message is 1576929633986


It took 20637 nanoseconds to decode the message.
The decoded message is 375973972


It took 663849291 nanoseconds to hack the RSA algorithm without using the p
rivate key.
The hacked message is 375973972
```

Figure 2. Online Replit run example of the RSA and the Quadratic Sieve algorithms

Results

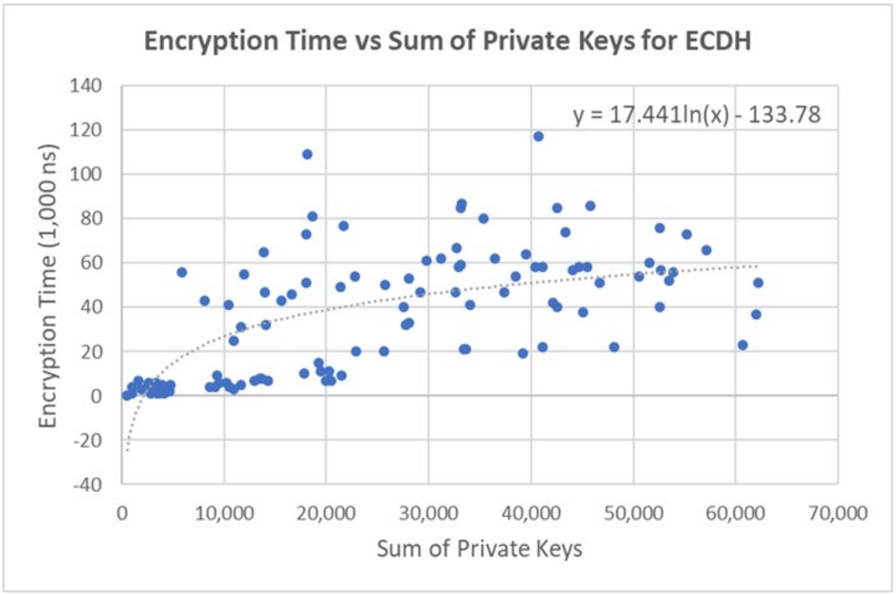


Figure 3. A logarithmic correlation can be seen between sum of private keys and run time

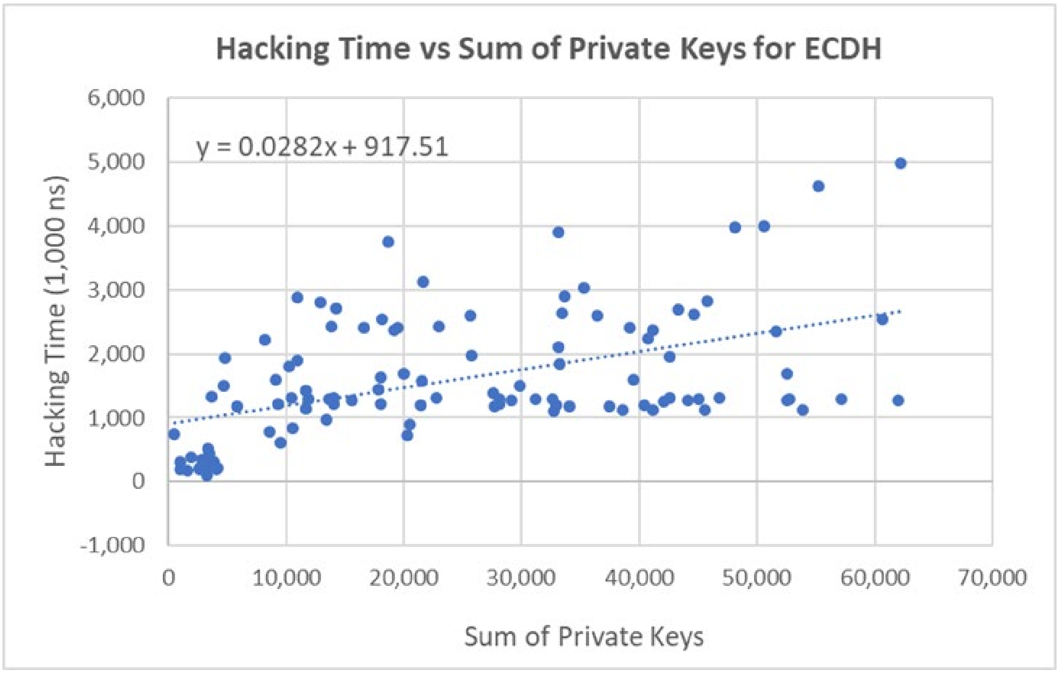


Figure 4. A linear correlation can be seen between sum of private keys and run time

The sum of private keys was used in Figure 3 and 4 to account for the cases where one of the private keys is large and the other is much smaller.

An example of a point on the two graphs above was with the elliptic curve $y^2 \equiv x^3 + 2579x + 2329\ (modulo\ 2647)$. The private keys were 2225 and 443. It took around 0.006 ms to encrypt with the ECDH algorithm, and 0.217 ms to hack using the Baby step Giant step algorithm.

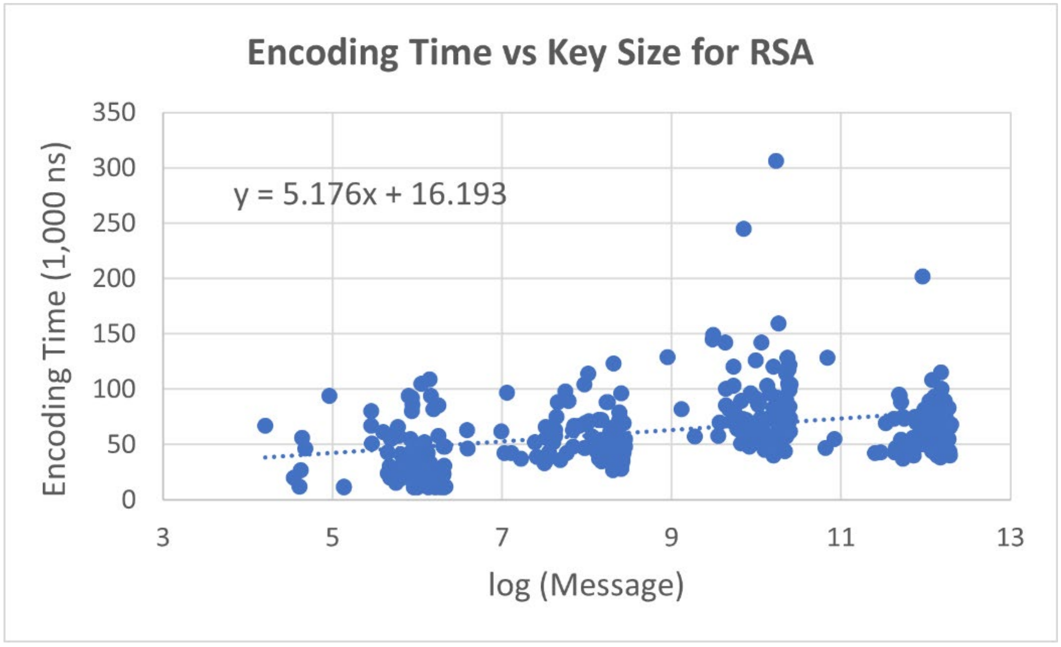


Figure 5. A logarithmic correlation can be seen between message size and run time

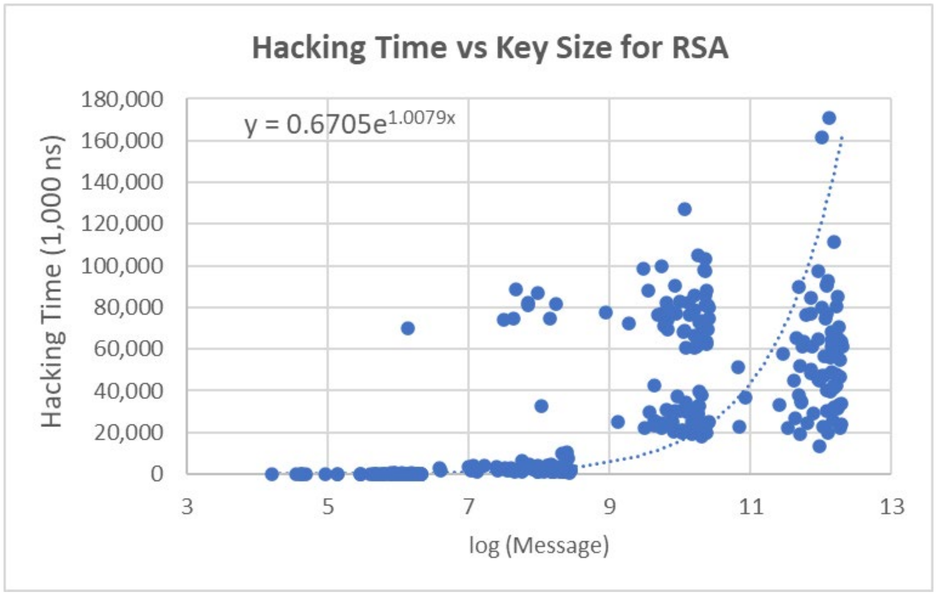


Figure 6. A linear correlation can be seen between message size and run time

The message size is used in Figure 5 and 6 because it accounts for the modulus size and how big the message is.

An example of a point on the two graphs above is with the modulus being 2000009000009 and the prime factors 1000003 and 2000003. The private key chosen was 49899262841 and the public key was 594260769599. The number chosen as the message to be sent was 1291964985154. It took around 0.082 ms to encrypt with the RSA algorithm and 170.733 ms to hack using the Quadratic Sieve algorithm.

Table 1. Extrapolated Results to Industry Standard

	RSA	ECDH
Industry standard key size	2^{2048}	2^{224}
Encryption time (seconds)	~0.07363	~0.00025741
Hacking time (centuries)	$5.0985 * 10^{605}$	$2.4108 * 10^{50}$

Discussion and Conclusion

Introduced as an alternative to the RSA in public key cryptography, the ECDH Key Exchange has an advantage over RSA in terms of key size at the same level strength or security. At industry standard of a 112-bit strength, the ECDH key size is 224 bits, and the RSA key size is 2048 bits. This means ECDH is more efficient given a smaller key size required at the same level of security.

At industry standard of 2048 bits for the RSA, my extrapolated data showed that it took about 0.07363 seconds to encrypt and $5.0985 * 10^{605}$ centuries to hack. Similarly, at the industry standard of 224 bits for the ECDH, it took about 0.00025741 seconds to encrypt and $2.4108 * 10^{50}$ to hack.

Compared with the ECDH, each at their respective industry strength, the RSA algorithm here took a lot longer to run, but the hacking time was also longer (Table 1). My extrapolated results demonstrated the efficiency of the ECDH algorithm as it was faster to encrypt but did not confirm the comparable security level at respective industry standard as hacking the RSA was slower. The Baby Step Giant Step algorithm to attack the ECDH was coded based on a relatively recent research paper (Aung, 2017) that appears to run efficiently, while the attack on the RSA was based on a basic version of the quadratic sieve that worked well for small prime products. More advanced steps are used in the industry to speed certain parts up, such as using partial relations and faster and better matrix processing. In addition, my code slowed down significantly in the step requiring linear algebra on Java to find subsets of vectors that would add to the zero vector, which I was unable to code.

Further, industry attacks on the RSA often involve specialized prime factorization attacks. These attacks may only function for key values that follow a certain pattern, but the attacks are very efficient. For example, the triple logarithmic factorization used by researchers in 2012 was able to crack around 0.2% of 6 million keys within 20 hours of computing. The RSA is very weak to such specialized attacks, while the ECDH is more resistant. Due to the risk of keys being cracked by specialized attacks, the RSA industry key size value needs to increase much more than the ECDH, making the algorithm much slower at the same level of security.

Additionally, for the ECDH, there are many more advanced variations on the algorithm providing a much larger degree of security. While the RSA has one main version and a couple small additions to allow for digital signatures, the ECDH has major variations, such as using hyperelliptic curves in higher dimensions, to provide more flexibility. Its many variations means that attacks will be less effective. The RSA algorithm is seen as a close to retiring algorithm, and is a relic of the past, while the ECDH algorithm will remain as the dominant industry cryptosystem (Kee, 2021).

In the upcoming decades, when quantum computers are developed fully and are operational, most current-day cryptosystems will need to change to remain quantum-secure. Elliptic curve cryptosystems and the RSA have a difficult order finding problem that ensure its security, but the Shor's Algorithm, a quantum-computer theoretical algorithm, will make this problem much faster to compute and will render these algorithms insecure. While current computers do discrete data calculations, quantum computers can compute analog data, and store probabilistic calculations, allowing it to compute many more options at the same instant. As long as the quantum computer is not observed or interfered with, it remains as a superposition of data. Though quantum computers are in theoretical stage, hackers can store important encrypted data, which could be decrypted later with quantum computers, necessitating the early development of quantum-secure algorithms ensure robust protection into the future.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

Aung, Tun Myat and Hla, Ni Ni, A Study of General Attacks on Elliptic Curve Discrete Logarithm Problem Over Prime Field and Binary Field (November 18, 2017). Available at http://dx.doi.org/10.2139/ssrn.3269714

Barker, E., Chen, L., Keller, S., Roginsky, A., Vassilev, A., I& Davis, R. (2017). Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography (No. NIST Special Publication (SP) 800-56A Rev. 3 (Draft)). National Institute of Standards and Technology.
https://doi.org/10.6028/NIST.SP.800-56Ar3

Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. Notices of the AMS, 46(2), 203-213.

Kee, L. (2021). Council post: RSA is dead - we just haven't accepted it yet. Forbes.
https://www.forbes.com/sites/forbestechcouncil/2021/05/06/rsa-is-dead—we-just-haventaccepted-ityet

McAndrew, A. (2016). Introduction to Cryptography with open-source software. CRC Press.

Menezes, A., Zuccherato, R., I& Wu, Y. H. (1996). An elementary introduction to hyperelliptic curves (pp. pp-155). Faculty of Mathematics, University of Waterloo.
https://api.semanticscholar.org/CorpusID:16341963

Pomerance, C. (2008). A tale of two sieves. Biscuits of Number Theory, 85, 175.
http://dx.doi.org/10.1090/dol/034/15

Rivest, R. L., Shamir, A., I& Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.
https://doi.org/10.7551/mitpress/12274.003.0047

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review, 41(2), 303-332. https://doi.org/10.1137/S0097539795293172