Comparative Analysis between Fast and Basic Scalar Multiplication Used in Elliptic Curve Cryptography

Research Proposal

Twan Boeve
t.boeve@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

Nowadays, more and more physical devices that are used in everyone's daily life are connected to the internet in order to exchange information with other devices. This is commonly known as the Internet of Things, or IoT for short, which includes increasingly more small devices with limited computation power and communication capacity. The problem here arises when there is a need to secure the data, which needs to be done using a cryptographic algorithm that is as efficient as possible for the IoT devices to be able to handle this. In this paper, the research into such an algorithm, called Elliptic Curve Cryptography (ECC), is shown. The paper focuses on the differences between this algorithm and the currently most-used RSA algorithm, and mainly on the differences between two methods of scalar multiplication used in ECC: basic and fast scalar multiplication operation.

KEYWORDS

Cryptography, Internet of Things, RSA, ECC, Scalar Multiplication

1 INTRODUCTION

The Internet of Things (IoT) has been rapidly expanding in the past few decades, especially now that even fridges and water kettles are connected to the internet [17]. However, being connected to the internet does have its risks and consequences since this means that (often sensitive) data is transferred which could be intercepted by outsiders [20].

For this reason, it is important to secure the transmission of data to be accessed by another device or service [10]. To be able to do this, a cryptographic algorithm is needed that can be used to secure the transmission of private/sensitive data via IoT devices [26]. Rivest–Shamir–Adleman (RSA) [21] is considered one of the main cryptographic algorithms that are used to secure the transmission of data between different parties in the network, but the problem with RSA and other traditional schemes is that they require a relatively high amount of computation power and communication capacity [7]. Nowadays most IoT end devices are resource-constrained [23]. For that reason, a lightweight algorithm with simpler and more efficient calculations is needed. This is where Elliptic Curve Cryptography (ECC) comes into play [16].

Both RSA and ECC make use of a *trapdoor* function [4]. This is a function that is easy to compute one way, but very difficult to compute the other way [1]. Within ECC, this trapdoor function is the scalar multiplication of a point. When encrypting a message using ECC, this trapdoor function is often executed many times, which brings the need for an efficient way of performing this scalar multiplication to again reduce the needed computation power and communication capacity of an IoT device [18]. This claimed quicker method is called 'fast scalar multiplication', which is used for both encryption and decryption in ECC.

To be able to better understand the differences between RSA and ECC, and also to get a better understanding of why ECC is considered better for resource-constrained devices, the first research question is as follows:

(1) "How does the ECC algorithm differ from the RSA algorithm?"

Since the need for a more efficient way of performing scalar multiplication is quickly rising with the rapidly increasing amount of small IoT devices, the second (and main) research question is the following:

(2) "What is the difference in computation and communication costs between basic and fast scalar multiplication in the context of ECC?"

To answer the first research question, both qualitative and quantitative data are gathered. The qualitative data is gained by extracting information from already existing scientific papers, as mentioned in Section 3. Since the exact differences between the algorithms are purely based on facts, analysing this data is unnecessary. The quantitative data is gathered by executing both algorithms on the Raspberry Pi and analysing the differences.

Research Paper, June 2022, Enschede

To answer the second research question, only quantitative data is gathered on how much different the computation and communication costs are between basic and fast scalar multiplication. The data is obtained by running Python scripts on the Raspberry Pi and then analyzed using Python libraries and a Volt/Ampere meter.

In this paper, first a literature review will be done on related work in section 3. Then the scenario will be discussed in section 4, including the hardware and software used for the research. After that, the results will be given from researching existing literature and performing the algorithms in section 5. Following will be a discussion on the given results in section 6. And finally, a conclusion is given on the research questions in section 7.

2 BACKGROUND

In this section, background is given on the separate algorithms used in this research. The two encryption algorithms are RSA and ECC, where first RSA is analyzed in subsection 2.1, then ECC is analyzed in subsection 2.2, and finally the differences between the two are analyzed in subsection 2.3.

2.1 RSA

To start off, the RSA algorithm will be analyzed. The development of the RSA algorithm already began in 1977 [9], making it one of the oldest public-key cryptosystems. Even though the algorithm was described over 40 years ago (as of the time of writing), it is still the most widely used encryption algorithm as it provides enough security for most use cases while still being manageable by most devices.

The reason that the RSA algorithm can be used by most devices nowadays is that it makes use of a so-called *trapdoor* function, making the encryption calculations relatively lightweight. A trapdoor function is a function that is easy to compute one way, but very difficult to compute the other way [1]. A trapdoor function is also used by the ECC algorithm but in a different way, as will be discussed later. In RSA, the trapdoor function is based on the difficulty of factoring a large integer into two large prime numbers. This is known as the Integer Factorization Problem. For example, it is very easy for a computer to calculate that 3 times 17 equals 51, whereas it is a lot more difficult for this same computer to calculate that the number 51 is a product of the two prime numbers 3 and 17. Applying this to a product of two large prime numbers, often consisting of over hundreds of digits, makes it very difficult for most devices to calculate what the original prime numbers are.

As told before in this section, RSA is a public-key cryptosystem. This means that it makes use of both a public key and a private key, which are both needed to be able to send and receive messages without them being able to be dismantled easily.

In RSA, the public key (PubKey) consists of a pair of numbers. The first of these is the number e, which is a prime number between 1 and the least common multiplier of the product of $(p-1)$ and $(q-1)$ (called r), where p and q are randomly generated prime numbers. e has to be co-prime to r, too. The second part of the PubKey is the number n, which is simply the product of p and q.

This same number n is also the second part of the private key (PrivKey) in RSA. The first part of the PrivKey is the number d, which is used for decrypting messages. d is calculated using the number e from the PubKey and the number r, which was calculated before. The formula for d is $(d * e) \mod r = 1$.

The key size of the RSA algorithm determines the size of the modulo r, and because of that essentially also the size of e and d used in the PubKey and PrivKey (since they make use of this modulo r). The greater the key size, the harder it becomes to calculate p and q when knowing the PubKey and thus the better the security.

2.2 ECC

Now the Elliptic Curve Cryptography (ECC) algorithm will be examined. The ECC algorithm was described in 1985, about 8 years later than the RSA algorithm. Even though it has been around for over 35 years now, it is only just becoming relatively popular. Its rise began in 2004, when the need for cost-efficient algorithms began to rise because of smaller IoT devices. The type of data they share varies, but examples are activity data of someone living in a house (for example to automatically turn lights on when the person is awake and at home) or signals (for example to be able to turn on a water kettle from a smartphone) [17].

As opposed to the RSA algorithm, the trapdoor function of the ECC algorithm is based on the difficulty of finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point [3], which is known as the *Elliptic Curve Discrete Logarithm Problem* (ECDLP). In other words, it is easy to compute the next point given a starting point (the *multiplicand*) and a multiplication, even when doing this many times, but (nearly) impossible to compute this multiplicand given the final product point.

The public key in an ECC cryptosystem essentially consists of all coordinates the dot calculations pass through, although these coordinates can be compressed into one end coordinate (called point Z) which is most often used as the single public key of the ECC algorithm.

The private key in ECC is the number n, which is the number of times the dot function is applied to reach the end coordinate from the starting coordinate. A single dot function consists of a starting point (A) and a line from A that passes through a point B on the same - in the case of ECC 'elliptic' - curve. The thing that is special about elliptic curves though is that such a line through A and B always goes through a third point on the curve (C). This point C is the result of the dot function applied to A and B. This is a single dot calculation which, again, is relatively easy to compute but very difficult to do backwards (retrieving A when knowing C)

The size of both the public and private keys determines the max value of the point on the elliptic curve for each calculation. If the

value of this point exceeds this max value, its modulus with this value is taken and used for the calculation. This means that with larger keys, more points are possible on the elliptic curve making it harder to find the private key when knowing the public key.

2.3 ECC vs RSA

Finally, the major differences between ECC and RSA will be discussed. The first and most important difference between the two is the key sizes required for equal amounts of security provided. In Table 1 below, the differences in key sizes between the RSA and ECC algorithms can be compared for different Security Bit Levels (source: [19]).

Security bits	RSA Key Length	ECC Key Length
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Table 1: Difference in key sizes between RSA and ECC

Since the major problems of IoT devices nowadays are the limited sizes of the computation and communication capacities, it is very important that lightweight algorithms can be used. As can be seen from the Table, the RSA algorithm requires far larger key sizes as the Security Bit Level increases. For example, for a Security Bit Level of 128 bits, RSA requires a key size of 3072 bits whereas ECC requires a key size of 256 bits while providing equal (if not better) security at the same time [19]. This was already clear in 2014, when it was proved that 1024-bit RSA is more unsafe than 160-bit ECC [2].

As mentioned before, ECC requires smaller key sizes in order to provide the same amount of security, allowing the memory size required by IoT devices to be smaller. However, RSA does prove to have more speed when generating private and public key pairs [15], meaning that it requires less computation power. This is due to the fact that performing product calculations, which are used in the RSA algorithm, is easier for a processor to do than performing dot calculations, which are used in the ECC algorithm.

However, the ECC algorithm clearly outperforms the RSA algorithm when it comes to decrypting messages, especially when it comes to larger key sizes. This is partially due to the fact that the required key sizes are smaller, but also because the calculation is easier [15].

All in all, the ECC algorithm proves far more useful than the RSA algorithm for resource-constrained devices. This is not only because the required key sizes are smaller for the same level of protection, but also because the average calculation difficulties are way lower, consequently making the computation costs lower and the requirement for good processors less.

3 RELATED WORK

In this section, it is discussed what is presented in the literature regarding the RSA and ECC algorithms and the usage of scalar multiplication in ECC algorithm.

ECC is not at all a new concept. Its first use was in 1985, when Koblitz [12] and Miller [22] proposed to use the group of points on an elliptic curve defined over a finite field in a cryptographic system. In their report [13] authors explained why ECC has not been used broadly and what the state of ECC is. It is obvious here that the report is slightly outdated, but it explains very well the issues that arise when attempting to implement the algorithm.

In 2012, Mike Hamburg released a report on the difference between fast and compact elliptic-curve cryptography [11]. Although this report does not have the exact same subject as the main research question proposed in Section 1, it is interesting that there is an entirely different approach in attempting to improve cryptosystem speeds, in this case by introducing a new implementation on curve signature and key agreement.

Another report, made by Matthieu Rivain from CryptoExperts [24], covers a subject very close to our main research problem: it compares fast and regular algorithms for scalar multiplication over elliptic curves. This paper is very interesting in this case since it explains clearly what the different fast algorithms are and how they are different from each other, which could very well help in the research towards the speed differences between these algorithms which we are planning to discover in this paper.

Finally, in their report, Benson and Ballard propose a framework for practical parallel fast matrix multiplication [5]. In this report, they do not only show how their framework works, but they also compare it to regular matrix multiplication algorithms along with the difference in results. Although their paper proposes a very swift algorithm, the authors do state that one of the most serious implementation challenges is that it requires a relatively big amount of communication capacity.

4 SCENARIO

In this section, the scenarios considered to answer the performance evaluation of both research questions are discussed. After that, the hardware and software used to perform these scenarios are given.

4.1 Research Question 1

For research question 1, the algorithms for both RSA and ECC (with basic scalar multiplication) need to be implemented. Since RSA is relatively basic to implement when compared to ECC, this can be implemented without the use of any non-standard (meaning: not official) library. All that is required to perform a full key generation, encryption, and decryption algorithm are some mathematical functions (such as the greatest common divisor) and a few standard libraries.

As stated in section 3, however, ECC appears to be relatively difficult to implement. For this reason, a library will be used that is

able to do basic scalar multiplication. The rest of the full algorithm is possible to be implemented without the use of other non-standard libraries.

4.2 Research Question 2

For the second research question, we encounter the same issue as described above for research question 1: ECC is very difficult to implement when compared to (for example) RSA. This problem extends even further with the implementation of the fast scalar multiplication algorithm if this would also be implemented from scratch. For this reason, a library is also used to implement the fast scalar multiplication portion of the ECC algorithm.

4.3 Hardware

To start off, all hardware will be listed. The computations and algorithms are run on a Raspberry Pi 4 Model B with 2GB of RAM. The reason a Raspberry Pi is used is that it can handle any software that was planned to be used in the research, and so others could reproduce the results easier since each Raspberry Pi of the same model should produce the same results when running the algorithms [8].

Furthermore, a Volt/Ampere meter is used that is connected to the Raspberry Pi. This is the USB Charger Doctor, which shows the voltage and current flowing through it, switching between the two every three seconds. This device is used to compute the computation costs for the algorithms. The connection with the Raspberry Pi can be found in Figure 1 below and consists of only a power adapter, the USB Charger Doctor, a power cable and the Raspberry Pi.

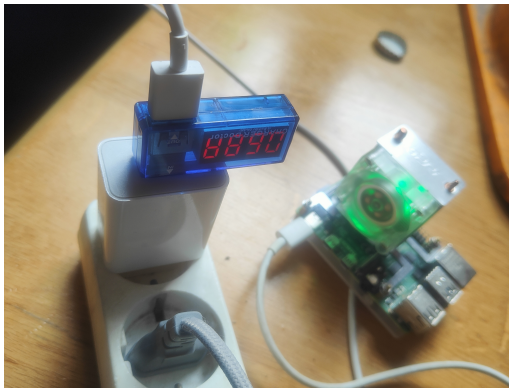

Figure 1: The setup used to measure the power consumption of the Raspberry Pi while running the different algorithms

4.4 Software

4.4.1 Operating Software. Since these two devices are all that is needed, now the software used on the Raspberry Pi will be discussed. The Raspberry Pi used to run the algorithms was installed with (at the time of writing) the latest version of Raspberry Pi OS, released on 04/04/2022.

4.4.2 Programming language. The programming language used for the algorithms is Python. There are several reasons for this, of which the main one is that there were already several libraries existent that were useful when implementing the different algorithms. On top of that, the programming language is often seen as easy to use and pick up.

4.4.3 Python libraries. Now the different Python libraries used will be listed. First off, some default Python libraries are used. These are the following:

- **time**: Timer used to compute the time it takes for an algorithm to finish
- **math**: Used in the RSA algorithm to easily compute the gcd (greatest common divisor) of two numbers.
- **random**: Random number generator used to generate random private keys for both RSA and ECC.
- **os and psutil**: Both used only once to measure the memory used when executing the algorithms.

On top of that, a few other libraries are used. The first of these is called *ecdsa* [25], which is a library used for key generation and basic scalar multiplication for the basic ECC algorithm.

The second library used is called *fastecdsa* [14], which is a library that is able to perform scalar multiplication operations (such as point addition or multiplication on an elliptic curve) relatively fast when compared to the basic ecdsa library. This library is very easy to use, since it does not change how points are added or multiplied; this is done almost exactly the same way this would normally be done, but now relatively quicker.

4.4.4 Produced Python scripts. Lastly, the produced Python scripts written for this research are used. These are six different files, consisting of a normal version of the three algorithms and looped versions of the algorithms, the latter of which were used to more accurately measure the computation power. The scripts can be found on the author's GitHub [6].

5 RESULTS

In this section, the results from performing the research will be given. This includes the results for both research questions, where the results from the first research question are derived from both existing literature and the algorithms, and the results for the second (and main) research question are derived solely from running the algorithms.

5.1 Research Question 1

The first research question is: "How does the ECC algorithm differ from the RSA algorithm?". As stated before, the results of this research question consist of both reviewing existing literature and of results from running the algorithms. The first of these two parts was discussed in section 2. For the second part, however, results are available on both the time difference and time difference when executing the two different algorithms and on the computation and communications costs needed for both algorithms.

5.1.1 Algorithms. To start off, the difference in time between both algorithms will be discussed. In Table 2 below, the time (in seconds) between the RSA algorithm and the basic ECC algorithm can be

found when executing them both fully several times; this includes key generation, encryption, and decryption.

Loops	RSA (in seconds)	ECC (in seconds)	Difference
1	7.287	0.0405	~180x
2	11.963	0.0842	~142x
10	89.84	0.3801	~236x
100	> 900	3.9891	>225x

Table 2: Difference in execution time between RSA and ECC

Now, the difference in memory usage between the same two algorithms will be shown. In Table 3 below, the memory usage can be seen after executing key generation, encryption, and decryption for both algorithms. Since the amount of loops did not have any influence on how much memory is used, a table is used instead of a graph. Do note that in the table RSA uses a keysize of 16 bits here whereas the ECC algorithm uses a keysize of 256 bits. This was unfortunately not possible to change for the test.

Algorithm	Memory used (in MB)
RSA (keysize 16 bits)	38.0273
Basic ECC (keysize 256 bits)	23.7

Table 3: Difference in memory usage between RSA and ECC

Finally, results are available on the power consumption of both algorithms. These can be found in Table 4 below. Also for the power consumption there was no influence on the amount of loops, so again a table is used instead of a graph.

Algorithm	Average current (in amperes)
RSA	0.80
Basic ECC	0.69

Table 4: Difference in power usage between RSA and ECC

5.2 Research Question 2

The second and main research question was stated as follows: "What is the difference in computation and communication costs between basic and fast scalar multiplication in the context of ECC?". As stated before, the results of this research question consist solely of results from running the algorithms. This includes both the computation and communication costs of the algorithms, although the time taken is also measured.

To start off, a graph is given below showing the differences in speed when encrypting the same message using equal-length encryption keys between basic and fast scalar multiplication when used in ECC. Results are included on when the algorithm is run once, 10 times, 100 times, 250 times, 500 times, 750 times, and finally 1000 times. The results can be found in Figure 2

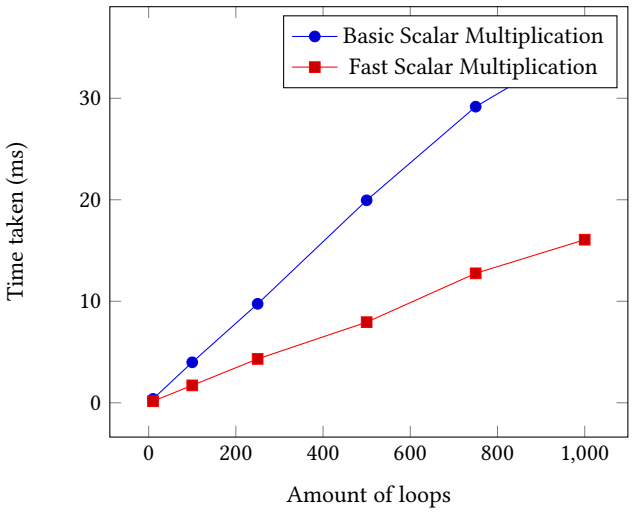

Figure 2: Speed difference between fast and scalar multiplication used in ECC

Next, the difference in memory usage between basic and fast scalar multiplication in ECC can be found in Table 5 below, the memory usage can be seen after executing key generation, encryption, and decryption for both algorithms. Again, the amount of loops did not have any influence on how much memory is used.

Algorithm	Memory used (in MB)
Basic ECC	23.7
Fast ECC	21.5

Table 5: Difference in memory usage between basic and fast ECC

Finally, results are available on the power consumption of both algorithms. These can be found in Table 6 below. Also here the amount of loops has no influence on the results.

Algorithm	Average current (in amperes)
Basic ECC	0.69
Fast ECC	0.76

Table 6: Difference in power usage between RSA and ECC

6 DISCUSSION

In this section, a discussion is given on the results. This includes whether the results are equal to the expected results, and if not, why they might differ.

6.1 Research Question 1

For the first research question, only the results from running the algorithms will be discussed since the literature review is based on facts and allows for little discussion. The results from the literature

review, however, do give a clear view of what is expected from the algorithms.

Looking at the speed differences between RSA and ECC given in Table 2, it is clear that ECC is much faster than RSA in the combination of key generation, encryption, and decryption. As mentioned in section 2 about the background of the project though, this was to be expected. One thing that is apparent is the fact that the difference seems to increase as more loops are performed. This most likely has to do with the fact that any potential startup delays when starting the loops have less effect on the total time when more loops are performed.

Secondly, the memory difference between the two algorithms will be discussed as given in Table 3. The difference in memory usage for both algorithms is not extreme, although it is lower for ECC, but the measurements were performed with different key sizes for the two algorithms: RSA used a key length of only 16 bits whereas ECC used a key length of 256 bits. From this it is clear that when equal key lengths would be used, ECC uses less memory on average than RSA.

Finally, results on difference in power consumption are available in Table 4. From the Table it is clear that basic ECC uses less power than RSA: on average, 0.69A is used by ECC whereas 0.80A is used by RSA while running the algorithms. Also this result was expected, as stated in section 2.

6.2 Research Question 2

Now the results from research question 2 will be discussed, as given in section 5. This research question covers the differences between basic and fast scalar multiplication used in ECC. For research question 2, there is only quantitative data to look at.

In Figure 2, the speed difference between the two algorithms can be found. Since the differences between the two algorithms were small enough (as opposed to RSA versus ECC), it was possible to put the results into a graph. From this graph, it is clear that the algorithm using fast scalar multiplication is faster than the algorithm using basic scalar multiplication. This, as the methods of scalar multiplication might suggest, is to be expected.

Next, the differences in memory usage between the two algorithms can be found in Table 5. Here it can be seen that the fast scalar multiplication algorithm uses around 2MB less than the basic one when executed. This, however, is most likely due to the fact that two libraries had to be used to decrypt messages for the basic algorithm, which resulted in a few extra variables that had to be used. All in all, there is almost no difference in memory usage between the two algorithms.

Finally, the power usage of the scalar multiplication operations can be found in Table 6. It can be seen that fast scalar multiplication used more power than basic scalar multiplication. Although it is not explicitly clear what the result here should be, it is expected that the power consumption is slightly higher for fast scalar multiplication since the multiplications performed are more advanced, although they are quicker than basic scalar multiplication.

7 CONCLUSION

In this final section a conclusion is given on the research questions and the final result. On top of that, any ideas for future work are listed.

To start off, research question 1 has been answered in full, and there is an obvious reason ECC is slowly becoming more popular than RSA. Smaller IoT devices require easy calculations and low computation and communication costs, which ECC excels at in comparison with RSA.

On top of that, research question 2 has also been answered: both the computation and communication costs are put into tables in section 5. From this we can conclude that fast scalar multiplication in ECC makes quite a big difference in speed, while only costing a little more memory to process (again, this most likely has to do with the fact that an additional library and additional libraries were required for basic ECC to work in this case).

All in all, ECC should definitely be used in (small) IoT devices, since its calculations are much quicker while its computation and communication costs are also more favourable than RSA. Fast scalar multiplication is also very desirable, especially when speed and power usage are very important. In terms of memory usage, there is only a small difference with basic scalar multiplication, if any.

What could still be studied to further specify the differences between basic and fast scalar multiplication in ECC is for example whether the same results are found for different key sizes, since it might be possible that the fast algorithm makes even more difference when used with even greater key sizes.

REFERENCES

[1] Mansoor Ahmed. What is a Trapdoor Function? https://dev.to/ahmedmansoor012/what-is-trapdoor-function-pb6, 2021. Accessed: 03/05/2022.

[2] Mohsen Bafandehkar, Sharifah Md Yasin, Ramlan Mahmod, and Zurina Mohd Hanapi. Comparison of ecc and rsa algorithm in resource constrained devices. In *2013 international conference on IT convergence and security (ICITCS)*, pages 1–3. IEEE, 2013.

[3] Liantao Bai, Yuegong Zhang, and Guoqiang Yang. Sm2 cryptographic algorithm based on discrete logarithm problem and prospect. In *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1294–1297. IEEE, 2012.

[4] Mihir Bellare and Silvio Micali. How to sign given any trapdoor function. In *Conference on the Theory and Application of Cryptography*, pages 200–215. Springer, 1988.

[5] Austin R. Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. *SIGPLAN Not.*, 50(8):42–53, jan 2015.

[6] Twan Boeve. Research Project ECC. GitHub, 2022. https://github.com/TwanBoeve/ResearchProjectECC.

[7] Kyung Jun Choi and Jong-In Song. Investigation of feasible cryptographic algorithms for wireless sensor network. In *2006 8th International Conference Advanced Communication Technology*, volume 2, pages 3 pp.–1381, 2006.

[8] Mohammed El-Haii, Maroun Chamoun, Ahmad Fadlallah, and Ahmed Serhrouchni. Analysis of cryptographic algorithms on iot hardware platforms. In *2018 2nd Cyber Security in Networking Conference (CSNet)*, pages 1–5. IEEE, 2018.

[9] S.L. Garfinkel. Public key cryptography. *Computer*, 29(6):101–104, 1996.

[10] Alan Grau. Can you trust your fridge? *IEEE Spectrum*, 52(3):50–56, 2015.

[11] Mike Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. https://ia.cr/2012/309.

[12] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
[13] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography*, 19(2):174, 2000.
[14] Anton Kueltz. fastecdsa. PyPi, 2021. https://pypi.org/project/fastecdsa/.
[15] Vivek B Kute, PR Paradhi, and GR Bamnote. A software comparison of rsa and ecc. *Int. J. Comput. Sci. Appl*, 2(1):43–59, 2009.
[16] Carlos Andres Lara-Nino, Arturo Diaz-Perez, and Miguel Morales-Sandoval. Elliptic curve lightweight cryptography: A survey. *IEEE Access*, 6:72514–72550, 2018.
[17] Joseph Lindley, Paul Coulton, and Rachel Cooper. Why the internet of things needs object orientated ontology. *The Design Journal*, 20(sup1):S2850–S2851, 2017.
[18] Julio López and Ricardo Dahab. An overview of elliptic curve cryptography. pages 1–2, 2000.
[19] Dindayal Mahto and Dilip Kumar Yadav. Rsa and ecc: a comparative analysis. *International journal of applied engineering research*, 12(19):9053–9061, 2017.
[20] Ulf T Mattsson. Database encryption-how to balance security with performance. *Available at SSRN 670561*, 2005.
[21] Evgeny Milanov. The rsa algorithm. *RSA laboratories*, pages 1–11, 2009.
[22] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.
[23] Effy Raja Naru, Hemraj Saini, and Mukesh Sharma. A recent review on lightweight cryptography in iot. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 887–890, 2017.
[24] Matthieu Rivain. Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338, 2011. https://ia.cr/2011/338.
[25] Brian Warner. ecdsa. PyPi, 2021. https://pypi.org/project/ecdsa/.
[26] Jinbo Xiong, Lei Chen, Md Zakirul Alam Bhuiyan, Chunjie Cao, Minshen Wang, Entao Luo, and Ximeng Liu. A secure data deletion scheme for iot devices through key derivation encryption and data analysis. *Future Generation Computer Systems*, 111:741–753, 2020.