Speeding-Up Elliptic Curve Cryptography Algorithms

Diana Maimuţ and Alexandru Cristian Matei

Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
{diana.maimut,alexandru.matei}@dcti.ro

Abstract. During the last decades there has been an increasing interest in Elliptic curve cryptography (ECC) and, especially, the Elliptic Curve Digital Signature Algorithm (ECDSA) in practice. The rather recent developments of emergent technologies, such as blockchain and the Internet of Things (IoT), have motivated researchers and developers to construct new cryptographic hardware accelerators for ECDSA. Different types of optimizations (either platform dependent or algorithmic) were presented in the literature. In this context, we turn our attention to ECC and propose a new method for generating ECDSA moduli with a predetermined portion that allows one to double the speed of Barrett's algorithm. Moreover, we take advantage of the advancements in the Artificial Intelligence (AI) field and bring forward an AI-based approach that enhances Schoof's algorithm for finding the number of points on an elliptic curve in terms of implementation efficiency. Our results represent algorithmic speed-ups exceeding the current paradigm as we are also preoccupied by other particular security environments meeting the needs of governmental organizations.

Keywords: Elliptic curve, elliptic curve cryptography, ECDSA, artificial intelligence, Schoof's algorithm, Barrett's algorithm.

1 Introduction

Elliptic curve cryptographic (ECC) was initially proposed in [25,31] as an alternative to the already established public key cryptographic schemes. As a side note, the credit for the first use of elliptic curves in a cryptology related context is given to Lenstra for his factorization algorithm [28]. ECC has received an increasing amount of attention in time not only for the high level provable security offered, but especially due to a desired property concerning the implementation efficiency: the cryptographic keys are significantly shorter compared to, e.g., the case of RSA [34].

For more than a decade now, ECC has become a central piece for the Blockchain technology. To be more specific, the Elliptic Curve Digital Signature Algorithm (ECDSA) [22] is widely adopted in the construction of cryptocurrency and, implicitly, blockchains. Thus, there has been a justified hype

with respect to efficient implementations of ECDSA and other ECC schemes in recent years, especially using Field Programmable Gate Arrays (FPGAs) [6–8,15,16,19,36,38,40]. Hence, FPGA-based hardware accelerators represent the main applied research topic when dealing with (permissioned) blockchains. To underline the importance of the subject, we have to mention that the main FPGA technology producer, Xilinx [4], organized a competition in 2021 [5], which encouraged R&D representants to propose, among other topics, blockchain-related projects [3].

Nonetheless, ECC grew compelling for another important emergent technology, i.e., the Internet of Things (IoT) [33]. Again, the interest for (hardware) optimized implementations of ECC algorithms is clearly shown in theory and practice [12, 13, 18, 20, 23, 24, 26, 27, 29, 30, 32, 35, 39]. Microprocessor, ASIC, and FPGA-based accelerators are proposed. Alongside hardware-specific optimizations, these cryptographic cores make use of different customary algorithmic improvements from the literature, mainly related to modular arithmetic.

Motivated by all the above and following the work of Géraud et al. from [14], this paper aims at building the foundation for new techniques of optimizing the implementation of ECDSA. As in the case of most public-key cryptosystems, the basic arithmetic operation used in ECC is the modular reduction. [14] describes a method allowing to double the speed of Barrett's algorithm [9] by using specific RSA moduli with a predetermined portion. The result is then applied in order to generate DSA [17] parameters. As an extension, our article presents a technique for generating ECDSA moduli with a predetermined portion that allows one to double the speed of Barrett's algorithm, a widely adopted efficient technique for performing modular reduction in a costly reduced manner. We also provide the reader with mathematical proof of our algorithm.

Moreover, we target a more general type of optimization suitable not only for ECDSA, but for various ECC algorithms. Thus, we propose an artificial intelligence (AI) based approach that enhances the speed of Schoof's algorithm for finding the number of points on an elliptic curve [37]. Schoof's method is the first deterministic polynomial time algorithm for counting points on elliptic curves defined over finite fields. The result represented, undoubtedly, a breakthrough in terms of designing ECC algorithms.

While the first result we propose is rather particular and can be applied for a certain digital signature scheme (ECDSA), the latter can be of general interest in terms of ECC algorithms. We underline that the AI-optimized variant of Schoof's algorithm is rather a proof of concept for a future series of results, with respect to this research direction.

We stress that all the related works regarding the ECC implementation optimizations [7, 8, 12, 13, 15, 16, 18–20, 23, 24, 26, 27, 29, 30, 32, 35, 36, 38–40] are targetting either specific curves (e.g., NIST P-256 [22], Secp256k1), curves over binary fields $GF(2^m)$, or curves over prime fields $GF(p)$. Note that our studies are different from the previously mentioned results, as we point out in Section 1.1.

Nonetheless, our methods can also be combined with already established algorithmic improvements to obtain even better implementation timings.

1.1 Specific Supplementary Motivation

The common practice when using ECC is to have specific curves [10, 22] rather than choose them every time when the algorithm is run in order to ease computations (by applying dedicated formulae for point addition and scalar multiplication). Nonetheless, speeding-up ECDSA in the general case can be advantageous, e.g., either for cryptographic implementations, requiring a higher level of security than the standard one, or simply for proprietary cryptographic algorithms (of course, given that a step consisting of checking the security of the generated curve is performed.). Such needs are customary, especially for governmental organizations. Moreover, in the aforementioned example, implementations for resource constrained devices and cryptographic hardware may be of great interest. Thus, when proposing our results, we do not seek to compare them with existing targeted ECC implementations in terms of speed, as we consider performing an initial costly step: the parameter generation.

In addition to the above, we believe that our AI-based strategy will benefit from the rapid advances in the field of AI in the near future.

1.2 Structure of the Paper

In Section 2, we introduce the notations and briefly describe Barrett's algorithm for modular reduction, Schoof's algorithm for point counting on elliptic curves, and ECDSA. The main results are discussed in Section 3, namely the algorithm for generating the ECDSA parameters and the method for finding the number of points of an elliptic curve. Details regarding the straightforward, unoptimized implementations of the previously mentioned algorithms are presented in Section 4. We conclude and provide the reader with future work ideas in Section 5. Moreover, we recall ECDSA in Appendix A and Schoof's algorithm in Appendix B.

2 Preliminaries

2.1 Notations

Throughout this paper, we denote by NextPrime(r) the smallest prime p, such that $p \geq r$. $\#S$ represents the cardinality of the set S. We let P be the bit-length of p, such that $P = ||p||$. The value of P is fixed from now onwards. The binary shift-to-the-right of x by y positions is further denoted by $x \gg y$.

2.2 Barrett's Algorithm

Let d and m be integer numbers. Barrett's algorithm (Algorithm 1) only uses two bit-shifts and one multiplication to produce an approximate value of the quotient obtained when d is divided by m. This approximation is denoted by c_3 and it satisfies the following inequality

$$\left\lfloor \frac{d}{m} \right\rfloor - 2 \leq c_3 \leq \left\lfloor \frac{d}{m} \right\rfloor,$$

which means that the whole loop is not repeated more than two times. The bit-lengths of d and m are represented by D and M. Algorithm 1 also requires a quantity that is denoted by L and which represents the maximal bit-length of the numbers that can be reduced. Barrett's algorithm works as long as the condition $D \leq L$ is satisfied. In most cases, these constants can be chosen such that $D = L = 2M$, provided that the reduction is performed after every operation. The constant k is computed only once, since it does not depend on the value of d. Further details regarding Algorithm 1 can be found in [9].

Algorithm 1: Barrett's algorithm for modular reduction.

Input: $m < 2^M, d < 2^D, k = \left\lfloor \frac{2^L}{m} \right\rfloor$ such that $M \leq D \leq L$
Output: $c_4 = d \pmod{m}$

```
1 c1 ← d ≫ (M − 1)
2 c2 ← c1 k
3 c3 ← c2 ≫ (L − M + 1)
4 c4 ← d − c3 m
5 while c4 ≥ m do
6 |   c4 ← c4 − m
7 return c4
```

2.3 Elliptic Curves in Cryptography

Schoof's Algorithm. The elliptic curves considered are of the form $y^2 = x^3 + ax + b$ and are defined over a finite field $\mathbb{F}_p$, where p is prime. An important result, which will be used throughout this paper, is the following theorem.

Theorem 1 (Hasse). *The number of points n of an elliptic curve defined over a finite field of size p satisfies the inequality*

$$|n - p - 1| \leq 2\sqrt{p}.$$

In [37], Schoof published the first deterministic and polynomial-time algorithm that computes the order of an elliptic curve, which is defined over a finite field. This algorithm starts off by using Theorem 1, which provides an interval of possible values for the order of the elliptic curve. That specific interval has the width $4\sqrt{p}$.

Since the order can be written as $\#E(\mathbb{F}_p) = p + 1 - t$, where t is the trace of the Frobenius endomorphism [41], the problem of finding the order reduces to that of finding the value of t. The next step involves computing the value of t modulo for some primes, such that their product is greater than $4\sqrt{p}$. Finally, the Chinese Remainder Theorem [41] produces the value of t, which is needed for finding the order.

The details of Schoof's algorithm are included in Appendix A as Algorithm 6.

ECDSA. ECDSA [22] is a digital signature scheme based on cyclic groups of elliptic curves defined over finite fields. Its security relies on the Elliptic Curve Discrete Logarithm Problem [21]. Details about setting-up the parameters of ECDSA, generating a signature and verifying it are included in Appendix A.

3 Main Results

3.1 Double-Speed Barrett for ECDSA

For the setup of ECDSA, two prime numbers p and n are required: p represents the size of the finite field and n is the order of the group $E(\mathbb{F}_p)$, since we are only considering the case when the order of this group is prime. Note that both the multiplications performed in Algorithm 1 are multiplications by constants, namely k and n.

Our aim is to generate the primes p and n such that their leading bits do not have to be computed. Moreover, we want this to happen also for the associated constants $k_p = \left\lfloor \frac{2^L}{p} \right\rfloor$ and $k_n = \left\lfloor \frac{2^L}{n} \right\rfloor$. The idea of Algorithm 2 is that if we choose the prime p in a convenient way, then we can control the most significant bits of n.

Algorithm 2: Generator for Barrett-compatible ECDSA parameters.

Input: P, the bit-length of the prime p, which has to be even and large
Output: (p, a, b, n), the parameters needed for ECDSA

1 $L \leftarrow 2P$
2 $U \leftarrow P/2$
3 Choose $r \in_R (0, 2^U)$
4 $\alpha = 2^{P-1} + 2^{U+1} + r$
5 $p = \text{NextPrime}(\alpha)$
6 **if** $p - \alpha \geq 0.7(P-1)$ **then** go back to step 3
7 Choose $a, b \in [0, p-1]$ such that $\#E(\mathbb{F}_p)$ is prime
8 $n \leftarrow \#E(\mathbb{F}_p)$
9 **return** (p, a, b, n)

Lemma 1. *Consider the notations used in Algorithm 2. Let* $Q = 2^{P-1} + 2^{U+1} + 2^U + 0.7(P-1)$. *Then* p, n, k_p *and* k_n *satisfy the following inequalities:*

1. $2^{P-1} < p < Q$;
2. $2^{P-1} < n < Q + 1 + 2\sqrt{Q}$;
3. $\left\lfloor \frac{2^{2P}}{Q} \right\rfloor \leq k_p < 2^{P+1}$;
4. $\left\lfloor \frac{2^{2P}}{Q+1+2\sqrt{Q}} \right\rfloor \leq k_n < 2^{P+1}$.

Proof.

1. Using the inequality $r < 2^U$ and Line 6 of Algorithm 2, we obtain that $p < Q$.
2. From Theorem 1 we have that $p+1-2\sqrt{p} < n < p+1+2\sqrt{p}$. Let $\epsilon = p-\alpha$. For the left side of the inequality, i.e., $2^{P-1} < n$, we obtain that

$$2^{P-1} + 2^{U+1} + r + \epsilon + 1 - 2\sqrt{2^{P-1} + 2^{U+1} + r + \epsilon} = p + 1 - 2\sqrt{p}.$$

   Thus,

$$2^{P-1} = (2^{P-1} + 2^{U+1}) - 2\sqrt{2^{P-1} + 2^{P-1}} <$$

$$< (2^{P-1} + 2^{U+1} + r + \epsilon + 1) - 2\sqrt{2^{P-1} + 2^{U+1} + r + \epsilon} = p + 1 - 2\sqrt{p}.$$

   For the right side of the inequality, i.e., $n < Q + 1 + 2\sqrt{Q}$, we know from Line 1 that $p < Q$. Therefore, we get $p + 1 + 2\sqrt{p} < Q + 1 + 2\sqrt{Q}$.
3. Using Line 2 of Algorithm 2, we can deduce that

$$\frac{2^{2P}}{Q} < \frac{2^{2P}}{p} < \frac{2^{2P}}{2^{P-1}}.$$

   Hence,

$$\left\lfloor \frac{2^{2P}}{Q} \right\rfloor \leq \left\lfloor \frac{2^{2P}}{p} \right\rfloor < 2^{P+1}.$$

4. Similarly, using Line 2 of Algorithm 2, we obtain that

$$\frac{2^{2P}}{Q + 1 + 2\sqrt{Q}} < \frac{2^{2P}}{n} < \frac{2^{2P}}{2^{P-1}}$$

   Therefore,

$$\left\lfloor \frac{2^{2P}}{Q + 1 + 2\sqrt{Q}} \right\rfloor \leq \left\lfloor \frac{2^{2P}}{n} \right\rfloor < 2^{P+1}.$$

Remark 1. In Line 6 of Algorithm 2, we do not allow the distance between α and the next prime p to be too large. Additionally, we choose α in a specific way so that we can control p's size. This implies that the probability of the first U bits of p being different than the first U bits of α is negligible. We performed 10^5 experiments with the value $P = 256$, and the success rate was 100%.

Example 1. This example illustrates Algorithm 2. For the values $P = 256$, $L = 512$, and $U = 128$, we get:

$r =$ 0xbba46de2b4b53e20b97d41941c01a6b0
$\alpha =$ 0x80000000000000000000000000000002bba46de2b4b53e20b97d41941c01a6b0
$p =$ 0x80000000000000000000000000000002bba46de2b4b53e20b97d41941c01a6ef
$n =$ 0x800000000000000000000000000000001b97deb5c74f4115fd2d9ec6ae5cb520f
$k_p =$ 0x1fffffffffffffffffffffffffffffff5116e48752d2b077d1a0af9af8ff9647f
$k_n =$ 0x1fffffffffffffffffffffffffffffff91a08528e2c2fba80b4984e5468d2b7db

3.2 Enhancing Schoof's Algorithm Using AI

Our aim is to modify Schoof's algorithm by replacing Hasse's interval with another one containing the order, such that the width of the new interval is smaller.

In order to obtain such a result, we can use a neural network that takes input triplets of the form $(p_i/2^P, a_i/2^P, b_i/2^P)$ and returns as output elements $\hat{y}_i$ given by

$$\hat{y}_i = \frac{\hat{n}_i - (p_i + 1) + 2\sqrt{p_i}}{4\sqrt{p_i}},\ 0 < \hat{y}_i < 1,$$

where we denote by $\hat{n}_i$ that the estimate of the actual order n_i. Here we use the sigmoid activation function for the output layer to ensure that the output is in the appropriate range. The labels used for training the neural network are written as

$$y_i = \frac{n_i - (p_i + 1) + 2\sqrt{p_i}}{4\sqrt{p_i}}, 0 < y_i < 1.$$

The elements of the training, validation, and test sets will be written in the form $(p_i^*/2^P, a_i^*/2^P, b_i^*/2^P, y_i^*)$, where instead of $*$, we will have the superscripts tr, v, and t, respectively. These three sets will have the cardinal numbers equal to N_{tr}, N_v, and N_t, respectively.

At training time, we choose to use as loss function the mean squared logarithmic error, since we want this to work well for large primes. This function is given by

$$\mathcal{L} = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \left[\log\left(\frac{1 + y_i^{tr}}{1 + \hat{y}_i^{tr}} \right) \right]^2 = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \left[\log\left(\frac{n_i^{tr} - p_i^{tr} - 1 + 6\sqrt{p_i^{tr}}}{\hat{n}_i^{tr} - p_i^{tr} - 1 + 6\sqrt{p_i^{tr}}} \right) \right]^2.$$

Let us denote by ϵ the average distance between the actual order and the estimate of the order, which is computed on the validation set,

$$\epsilon = \left\lfloor \frac{1}{N_v} \sum_{i=1}^{N_v} |n_i^v - \hat{n}_i^v| \right\rfloor.$$

Our approach is a probabilistic one, since we need to assume that the order n satisfies the inequality

$$\hat{n} - 2\epsilon < n < \hat{n} + 2\epsilon. \tag{1}$$

This leads to the following result involving t,

$$(p + 1 - \hat{n}) - 2\epsilon < t < (p + 1 - \hat{n}) + 2\epsilon.$$

Notice that in the above inequality, we have doubled ϵ in order to increase the probability that our assumption is true. Hence, if we manage to determine the value of $t_0 \equiv t \pmod{4\epsilon}$, then we can find t by replacing t_0 in the formula $t = (p + 1 - \hat{n}) - 2\epsilon + t_0$, and thus we know the order of the group. The benefit of using the estimate given by the neural network is that Schoof's algorithm can be applied for an interval of width equal to 4ϵ instead of one of width equal to $4\sqrt{p}$. This means that if the neural network is good at estimating the order, i.e., $\epsilon < \sqrt{p}$, then this approach will be faster than the standard one.

Remark 2. Since our algorithm is probabilistic, after obtaining a value of ϵ, which is significantly lower than $\sqrt{p}$, instead of assuming that n lies in the interval $(\hat{n}-2\epsilon, \hat{n}+2\epsilon)$, we can assume that it lies in an interval of greater width, for example $(\hat{n}-4\epsilon, \hat{n}+4\epsilon)$. By doing this, we are able to increase the success probability of the algorithm.

Remark 3. The difference between Schoof's algorithm and our proposed technique is that we choose the set of primes from Line 1 of Appendix B, such that the product of the elements is greater than 4ϵ. All the steps that follow remain unchanged.

4 Implementation

4.1 GitHub Implementation

We refer the reader to [1] for the source code representing the implementation of our proposed results.

Note that for simplicity, we overlooked the initial part of Algorithm 2 (i.e., from Line 1 to Line 6) in our implementation.

4.2 Implementation Results

We ran the code for our algorithm on a standard laptop using Ubuntu 20.04.5 LTS OS, with the following specifications: Intel Core i3-1005G1 with 2 cores and 8 Gigabytes of RAM. The programming language we used for implementing our algorithms was Python, and the AI library we chose was TensorFlow.

AI-Based Speed-Up. Our AI-based technique can speed up the search for an elliptic curve of prime order. This speed-up depends initially on the model architecture and then on the accuracy of the AI-model. Within the current section, we report our (proof of concept) results.

To achieve our proof of concept goal, in our implementation we initially considered primes p of length 32 bits. Thus, we generated 60,000 elliptic curves of the form (p, a, b, n) by means of Schoof's algorithm. Based on these examples, we trained, validated, and tested the neural network model we chose. This network was composed of 7 dense hidden layers with the number of units decreasing from 512 to 8. Note that decreasing the number of units, as stated before, is a straightforward technique used in AI algorithms. The reason we decided to have 7 hidden layers was to obtain the best compromise in terms of error rate and code optimization (especially with respect to time complexity). We provide the reader with a graphical representation of the relationship between the number of neural network layers and the error rate of our proposed algorithm in Figure 1. Note that the error rate stabilizes at 7%, starting with the use of 7 layers.

Hence, using the previously described neural network, we managed to replace Hasse's interval with another interval, with the width approximately 15%

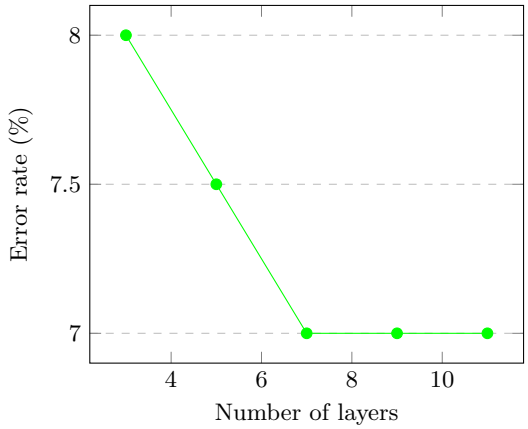

Fig. 1. The relationship between the number of neural network layers and the error rate of our proposed algorithm.

smaller than the original one. In this case, the probability that the order n satisfies Equation (1) was 93%, which was also the success rate of our probabilistic algorithm. The obtained probability was computed by finding the number of testing examples, which satisfied Equation (1).

We provide the reader with a graphical representation of the relationship between the number of neural network layers and the reduced Hasse interval of our proposed algorithm in Figure 2. Note that the percent by which the width of our reduced Hasse's interval is smaller than the original one stabilizes right after 7 layers at 16%. Note that the next value considered after 7 is 9 given that the AI models work better in the case of an odd number of layers.

Thus, given the results presented in Figures 1 and 2, we chose to use 7 layers in our implementation as a trade-off between accuracy and time complexity. Table 4.2 shows that the difference between the timing of the 7 layers implementation version and the 9 layers version is significant (the latter is 55% slower) and clearly sustains our choice of parameters.

Due to the fact that our proposed result is a particular algorithmic improvement, we compare it in terms of efficiency with the original Schoof algorithm (see Remark 3) and provide the reader with precise timings in Table 4.2. The average timings we report are given for 32, 48, and 64 bits prime numbers. It is straightforward for any other publicly available implementation optimization to be applied in our case too, and, thus, we can obtain the best timings.

Barrett-Based ECDSA Speed-Up. Based on the results in [14], we showed that using Barrett-compatible ECDSA parameters doubles the speed of Barrett's algorithm when performing the modular reductions required for generating (Algorithm 4) and verifying (Algorithm 5) ECDSA signatures. Thus, in such an optimized ECDSA implementation, the steps including modular reduction are performed two times faster than in a standard ECDSA implementation.

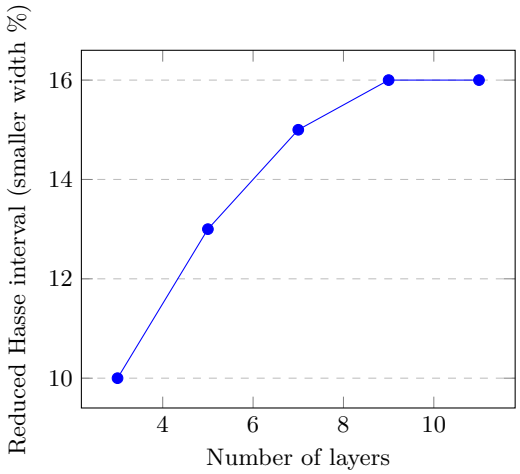

Fig. 2. The relationship between the number of neural network layers and the reduced Hasse interval.

Number of Layers	**Timing (seconds)**
7	1032
9	1597

Table 1. Timing comparison between the implementation of our proposed algorithm using 7 and 9 layers, respectively.

The authors of [7] report the fastest implementation of the ECDSA verification algorithm in FPGA compared to the already established results in the literature so far. The majority of papers presenting hardware optimizations for blockchain applications are only discussing the verification algorithm of ECDSA. Nonetheless, we are interested in optimizing the complete ECDSA scheme as our proposed speed-ups can also be applied for the signature generation algorithm (already stated in Section 4.2), not only for the verification algorithm.

Given that our FPGA implementation is work in progress, at this point, our target is to make a software implementation comparison of both the signing and the verification ECDSA algorithms. Thus, we considered the fastest (lightweight) ECDSA implementation available online [2] and modified it to include our proposed optimization from Section 3.1. The average time differences we obtained after 100 runs are presented in Table 4.2. Note that in the implementation at [2], the modular reduction steps are performed in a straightforward manner as opposed to our proposed double-speed Barrett optimization. Hence, the speed-up is obvious both in theory (see Section 4.2) and in practice (see Table 4.2).

Prime Length (bits)	Algorithm	Timing (seconds)
32	Enhanced Schoof	14.97
	Original Schoof	33.42
48	Enhanced Schoof	205
	Original Schoof	448
64	Enhanced Schoof	1926
	Original Schoof	3635

Table 2. Timing comparison between the implementation of our proposed algorithm and the original Schoof algorithm.

ECDSA Algorithm	Timing (milliseconds)
Enhanced Signature	4
Signature	4.6
Enhanced Verification	7.5
Verification	9

Table 3. Timing comparison between a lightweight ECDSA implementation and our enhanced version of it.

5 Conclusions and Future Work

We briefly described Barrett's algorithm for modular reduction, Schoof's algorithm for point counting on elliptic curves, and ECDSA as an example for applying our proposed speed-ups. We presented as main results an algorithm for generating implementation-friendly ECDSA parameters and a method for finding the number of points of an elliptic curve, representing an enhancement of Schoof's algorithm. We also gave details regarding the unoptimized implementations of the previously mentioned algorithms.

Future Work. We consider that timing comparisons between our enhanced Schoof algorithm and already established implementations of SEA [11] represent an interesting idea to be tackled in the near future.

Next, a valuable idea is looking into more sophisticated AI optimizations for the mathematical computations inside Schoof's algorithm.

Another interesting research direction is the implementantion of ECDSA in cryptographic hardware while using our proposed optimizations, together with a complexity analysis of other implementations in the literature. We are currently working on such an approach using FPGA-based equipment.

References

1. https://github.com/cryptocrew601/schoof
2. https://github.com/starkbank/ecdsa-python
3. https://www.hackster.io/contests/xilinxadaptivecomputing2021
4. https://www.xilinx.com
5. https://www.xilinx.com/developer/adaptive-computing-challenge/contest-2021.html
6. https://www.xilinx.com/products/intellectual-property/1-175rk99.html
7. Agrawal, R., Yang, J., Javaid, H.: Efficient FPGA-based ECDSA Verification Engine for Permissioned Blockchains. CoRR **abs/2112.02229** (2021), https://arxiv.org/abs/2112.02229
8. Awaludin, A.M., Larasati, H.T., Kim, H.: High-speed and unified ecc processor for generic weierstrass curves over gf(p) on fpga. Sensors **21**(4) (2021)
9. Barrett, P.: Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In: CRYPTO'86. Lecture Notes in Computer Science, vol. 263, pp. 311–323. Springer (1986)
10. Bernstein, D.J.: Curve25519: New Diffie-Hellman Speed Records. In: PKC'06. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer (2006)
11. Dewaghe, L.: Remarks on the Schoof-Elkies-Atkin Algorithm. Mathematics of Computation **67**(223), 1247–1252 (1998)
12. Dhillon, P.K., Kalra, S.: Elliptic curve cryptography for real time embedded systems in IoT networks. In: 5th International Conference on Wireless Networks and Embedded Systems (WECON). pp. 1–6 (2016)
13. Di Matteo, S., Baldanzi, L., Crocetti, L., Nannipieri, P., Fanucci, L., Saponara, S.: Secure Elliptic Curve Crypto-Processor for Real-Time IoT Applications, journal = Energies **14**(15) (2021)
14. Géraud, R., Maimuţ, D., Naccache, D.: Double-Speed Barrett Moduli. In: The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, vol. 9100, pp. 148–158. Springer (2016)
15. Gövem, B., Järvinen, K., Aerts, K., Verbauwhede, I., Mentens, N.: A Fast and Compact FPGA Implementation of Elliptic Curve Cryptography Using Lambda Coordinates. In: AFRICACRYPT'16. Lecture Notes in Computer Science, vol. 9646, pp. 63–83. Springer (2016)
16. Güneysu, T., Paar, C.: Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In: CHES'08. Lecture Notes in Computer Science, vol. 5154, pp. 62–78. Springer (2008)
17. Hoffstein, J., Pipher, J., Silverman, J.: An Introduction to Mathematical Cryptography. Undergraduate Texts in Mathematics, Springer (2008)
18. Hu, X., Zheng, X., Zhang, S., Cai, S., Xiong, X.: A Low Hardware Consumption Elliptic Curve Cryptographic Architecture over GF(p) in Embedded Application. Electronics **7**(7) (2018)
19. Javaid, H., Yang, J., Santoso, N., Upadhyay, M., Mohan, S., Hu, C., Brebner, G.: Blockchain Machine: A Network-Attached Hardware Accelerator for Hyperledger Fabric (2021)
20. Ji, J.H., Kima, H.: ASIC implementation for an ECC processor. IDEC Journal of Integrated Circuits and Systems **4**(2), 1–5 (2018)
21. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007)

22. Kerry, C.F., Romine, C.: FIPS PUB 186-4 Federal Information Processing Standards Publication Digital Signature Standard (DSS) (2013)
23. Kieu-Do-Nguyen, B., Pham-Quoc, C., Tran, N.T., Pham, C.K., Hoang, T.T.: Low-Cost Area-Efficient FPGA-Based Multi-Functional ECDSA/EdDSA. Cryptography **6**(2) (2022)
24. Kneevi, M., Nikov, V., Rombouts, P.: Low-Latency ECDSA Signature VerificationA Road Toward Safer Traffic. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **24**(11), 3257–3267 (2016)
25. Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation **48**(177), 203–209 (Jan 1987)
26. Kudithi, T., Sakthivel, R.: High-performance ECC processor architecture design for IoT security applications. The Journal of Supercomputing **75**, 447–474 (2018)
27. Lara-Nino, C.A., Diaz-Perez, A., Morales-Sandoval, M.: Lightweight elliptic curve cryptography accelerator for internet of things applications. Ad Hoc Networks **103**, 102159 (2020)
28. Lenstra, H.W.: Factoring Integers with Elliptic Curves. Annals of Mathematics **126**(3), 649–673 (1987)
29. Liu, Z., Huang, X., Hu, Z., Khan, M.K., Seo, H., Zhou, L.: On Emerging Family of Elliptic Curves to Secure Internet of Things: ECC Comes of Age. IEEE Transactions on Dependable and Secure Computing **14**(3), 237–248 (2017)
30. Liu, Z., Seo, H., Großschädl, J., Kim, H.: Efficient Implementation of NIST-Compliant Elliptic Curve Cryptography for Sensor Nodes. In: ICICS'13. Lecture Notes in Computer Science, vol. 8233, pp. 302–317. Springer (2013)
31. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Proceedings of the CRYPTO '85. Lecture Notes in Computer Science, vol. 218, pp. 417–426. Springer (1986)
32. Park, D., Chang, N.S., Lee, S., Hong, S.: Fast Implementation of NIST P-256 Elliptic Curve Cryptography on 8-Bit AVR Processor. Applied Sciences **10**(24) (2020)
33. Poschmann, A.: Lightweight Cryptography - Cryptographic Engineering for a Pervasive World. IACR Cryptology ePrint Archive **2009**, 516 (2009)
34. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
35. Salarifard, R., Bayat-Sarmadi, S.: An Efficient Low-Latency Point-Multiplication Over Curve25519. IEEE Transactions on Circuits and Systems I: Regular Papers **66**(10), 3854–3862 (2019)
36. Sau, S., Baidya, P., Paul, R., Mandal, S.: Binary Field Point Multiplication Implementation in FPGA Hardware. In: Intelligent and Cloud Computing. pp. 387–394. Springer Singapore (2021)
37. Schoof, R.: Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p. Mathematics of Computation pp. 483–494 (1985)
38. Sghaier, A., Zeghid, M., Massoud, C., Mahchout, M.: Design And Implementation of Low Area/Power Elliptic Curve Digital Signature Hardware Core. Electronics **6**(2) (2017)
39. Simon Francia, A., Solis-Lastra, J., Papa Quiroz, E.A.: Elliptic Curves Cryptography for Lightweight Devices in IoT Systems. In: Emerging Research in Intelligent Systems. pp. 71–82. Springer (2022)
40. Tachibana, S., Araki, S., Kajihara, S., Azuchi, S., Nakajo, Y., Shoda, H.: FPGA implementation of ECDSA for Blockchain. 2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW) pp. 1–2 (2019)

41. Washington, L.C.: Elliptic Curves: Number Theory and Cryptography, Second Edition. Chapman & Hall/CRC, 2 edn. (2008)

A ECDSA

Algorithm 3: ECDSA Setup

Input: q, a, b, defining the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_q$
Output: g, n, d, R

choose the subgroup $G \subset E(\mathbb{F}_q)$ such that the order of G is prime and $G = (g)$
$n \leftarrow \#G$
the private key d given by $d \in_R [1, n-1]$
the public key R is given by $R \leftarrow dg$
return g, n, d, R

Algorithm 4: ECDSA Signature

Input: m, q, a, b, g, n, d
Output: the signature generated is the pair (r, s)

$h \leftarrow \text{hash}(m) \in [0, n-1]$
$k \in_R [1, n-1]$
$(x, y) \leftarrow kg$
$r \leftarrow x \pmod{n}$
if $r = 0$ **then** go back to line 2
$s \leftarrow (h + rd)/k \pmod{n}$
if $s = 0$ **then** go back to line 2
return r, s

Algorithm 5: ECDSA Verification

Input: $m, q, a, b, g, n, R, r, s$
Output: the signature is valid if and only if the algorithm returns 1

$w \leftarrow s^{-1} \pmod{n}$
$u \leftarrow hw \pmod{n}$
$v \leftarrow rw \pmod{n}$
$(x_Q, y_Q) \leftarrow ug + vR$
if $x_Q \equiv r \pmod{n}$ **then return** 1
else return 0

B Schoof's Algorithm

Algorithm 6: Schoof's Algorithm

Input: q, a, b, defining the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_q$
Output: $\#E(\mathbb{F}_q)$

choose a set of primes $S = \{2, 3, \ldots, L\}$ such that $\text{char}(\mathbb{F}_q) \notin S$ and $\mathcal{P} = \prod_{\ell \in S} \ell > 4\sqrt{q}$
if $\gcd(x^3 + ax + b, x^q - x) \neq 1$ **then** $t \equiv 0 \pmod 2$
else $t \equiv 1 \pmod 2$
for $\ell \in S \setminus \{2\}$ **do**
    $q_\ell \leftarrow q \pmod \ell$ such that $|q_\ell| < \ell/2$
    compute the x-coordinate of the point (x', y'), where

$$(x', y') = (x^{q^2}, y^{q^2}) + q_\ell(x, y) \pmod{\psi_\ell}$$

    for $j = 1, 2, \ldots, (\ell - 1)/2$ **do**
        compute the x-coordinate of the point (x_j, y_j), where

$$(x_j, y_j) = j(x, y)$$

        if $x' - x_j^q \equiv 0 \pmod{\psi_\ell}$ **then**
            compute y' and y_j
            if $(y' - y_j^q)/y \equiv 0 \pmod{\psi_\ell}$ **then** $t \equiv j \pmod \ell$
            else $t \equiv -j \pmod \ell$
    if all values of j have been tried without success **then**
        if q is not a quadratic residue modulo ℓ **then** $t \equiv 0 \pmod \ell$
        else
            Let w be such that $w^2 \equiv q \pmod \ell$
            if $\gcd(\text{numerator}(x^q - x_w), \psi_\ell) = 1$ **then** $t \equiv 0 \pmod \ell$
            else
                if $\gcd(\text{numerator}((y^q - y_w)/y), \psi_\ell) \neq 1$ **then** $t \equiv 2w \pmod \ell$
                else $t \equiv -2w \pmod \ell$
solve the system of congruences to find $t \pmod{\mathcal{P}}$
choose the value of t such that $|t| \leq 2\sqrt{q}$
return $q + 1 - t$