A Side-Channel Attack Resistive ECDSA

A.Samir Abo-Taleb[1], M. Shalaby[2], M. Nabil[3] and Salwa Elramly[1]

[1] Ain Shams University, Egypt
[2] Egyptian Armed Forces, Arab Academy for Science Technology and Maritime Transport, Egypt
[3]Armed Forces Research and Development Center, Egypt

10271@eng.asu.edu.eg, myousef73@hotmail.com, m_nabil1974@yahoo.com and salwahelramly@gmail.com

Abstract. Elliptic curve plays an important role in securing critical software applications as it is one of the most powerful and widely utilized encryption algorithms. It is used in many applications such as digital signature schemes. Elliptic curve algorithms are hard to break, however, attackers find another indirect and more efficient methods to estimate secret information behind any secured systems with the help of information leakages so that they can successfully reconstruct critical objects like private keys. These methods are called side-channel attacks. In this paper we propose a software implementation of ECDSA which can counteract cache memory side-channel attack using three techniques, namely, "Fisher Yates" algorithm, volatile memory objects, and thread locking.
Keywords: Elliptic curve cryptography, Digital Signatures, Side-Channel Attack

1. Introduction

Elliptic curves are used as an extension to other cryptosystems such as elliptic curve Diffie-Hellman key exchange (ECDH) which is a key agreement, pseudo-random generators and, elliptic curve digital signature algorithm (ECDSA). The advantage of ECC is it requires a smaller keys compared with other cryptographic algorithms to provide equivalent security [1][2]. Because of increasing usage of online authentications, digital signatures are developed to represent (in real-world) human being in his absence. There are many applications rely on digital signatures such as signing and message authentication [3]. Banking transactions are one of the most important processes that can take place between two parties far from each other and hence authentication is required. In this paper we present a secured authentication method based on ECDSA that can resist cache memory side-channel attacks.

2. Elliptic curve overview

In this section we present an overview of elliptic curve and some topics that are related to it.

2.1. Elliptic curve cryptography

Elliptic curve cryptography (ECC) is a public key encryption algorithm that can provide asymmetric cryptographic keys. ECC generates keys based on the properties of the elliptic curve equation and its input parameters including the selected prime fields, GF(p), which can yield a level of security with a smaller key compared with other systems that require a bigger one to achieve the same level of security. Furthermore, ECC establishes security with less computing power and battery resource usage

than its counterparts, and hence it is widely used for mobile applications [4]. EC was defined by Neal Koblitz and Victor Miller over Z_P, where Z_P= {0, 1,…,p-1}, as the set of points (x, y) including an imaginary point O which satisfy equations (1) and (2) [5].

$$y^2 \equiv x^3 + ax + b \quad mod(p) \qquad (1)$$

$$4a^3 + 27b^2 \neq 0 \quad mod(p) \qquad (2)$$

2.2. EC mathematics

EC mathematics are based on finite fields and modular arithmetic which make the operation easy to calculate and hard to reverse. There are many methods to generate points on elliptic curve [6][7]. Given two points on elliptic curve P and Q, there are two mathematical group operations, first one is point adding R = P + Q where P ≠ Q and the second is point doubling R = P + P = 2P. The calculated point R yield to the following:

$$x_3 = s^2 - x_1 - x_2 \qquad mod\ p \qquad (3)$$

$$y_3 = s(x_1 - x_3) - y_1 \quad mod\ p \qquad (4)$$

Where the calculation of *s* depends on the type of operation as follows:

1. For point addition:

$$s = \frac{y_2 - y_1}{x_2 - x_1} \quad mod\ p \qquad (5)$$

2. For point doubling:

$$s = \frac{3x_1^2}{2y_1} \qquad mod\ p \qquad (6)$$

2.3. Discrete logarithm problem

The EC is based on the hardness of a discrete logarithm problem (ECDLP) [5] to generate the public point. Given an elliptic curve E defined over a finite field F_p with order n and public point P = dG where d is the private key and G is the curve base point, it's very hard to calculate d from P & G since we deal with very big integers. One way to do so is to perform brute force attack that scans all generated points from the base point G and compares the results with the public key P. This operation takes many years and it depends on the key length and the number of instructions executed per second (IPS). ECDLP can be solved for small curve parameters so NIST defined some domain parameters which are standard and recommended to work with [8].

2.4. Elliptic curve digital signature algorithm (ECDSA)

In 1998, ECDSA was standardized by the American National Standards Institute (ANSI) in US. The ECDSA is defined for EC over prime fields Z_p and Galois fields GF(2^m). The signature of a message x is created by the following steps:

1. Choose a random integer d that represents the private key in which 0 < d < q where q is the curve order.
2. Calculate P = dG where P:is the curve public point, G: is the curve base point.
3. Set the generated point x-coordinate to a variable denoted by r and if it was equal to zero, one has to choose another private key.
4. Calculate $S \equiv [\ h(x) + d.r\ ]d^{-1}\ mod\ q$.
5. The generated signature for the message is (r, s).

The signed message along with its signature is sent to the recipient and the verification process is conducted as shown in the following steps:

1. Calculate $W \equiv s^{-1}\ mod\ q$.

2. Perform hashing to the message content with the same hash algorithm used in signing it, then convert the hash value from hexadecimal form to integer.
3. Calculate $U_1 \equiv W.h(x) \mod q$.
4. Calculate $U_2 \equiv W.r \mod q$.
5. Calculate $P = U_1 A + U_2 B$.
6. Integrity check:
   If $X_P \equiv r \mod q$ then, the signature is valid. Otherwise, signature is not valid.

3. The Related work

There are many researches that implement digital sinature using elliptic curve. In [9], the authors proposed an implementation of ECDSA using P-192 elliptic curve. They provided a multi-server authentication scheme that has a low cost for both computation and communication. In [10], the authers studied different set of elliptic curves, and presented a performance and security analysis of using these sets in cryptography.

However, [9, 10] did not address the attacks that might exploit the cache memory side-channel attack (SCA) or man-in-the-middle attack (MITM) to inspect the transferred packets and then analyze them to gain illegal access to personal computers.

4. The proposed work

In this section we propose our work as follows:

1. Random private key generation.
2. Performing a mitigation technique against SCA while signing a document.
3. Use case implementation.

4.1. Random private key generation

Here, we randomly generate private keys using "fisher-yates" algorithm which is used to shuffle array elements regardless its length. For example, if we want to generate a random integer number that consists of two digits then, we have to create two arrays with length 10 each (figure 1). Each array holds elements with value range from 0 to 9. These primitive elements can create any number by appending another array that contains shuffled elements. The output is an integer value of 2 digits including zero element. The proposed generation function takes the curve order as a parameter in which $0 < d < n$ where d is the generated private key and n is the curve order.

Figure 2 illustrates the random generation technique of private key that yield to the selected elliptic curve prime field parameters. The array that is responsible for most significant digit creation, is contains only 9 elements instead of 10 elements. This is why that the most left digits can't be 0. After shuffled each array elements we have 9 distinct private keys then we choose a random number from 1 to 9 to select the random private key from the set. The generation technique is then, a random of random choices.

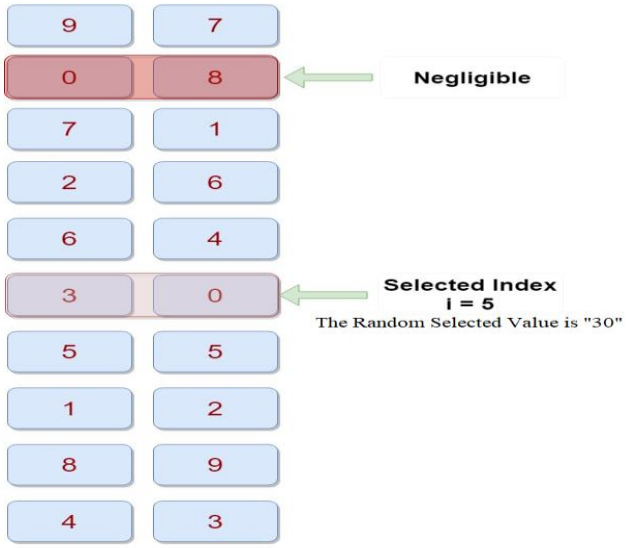

Figure 1. Integer number construction

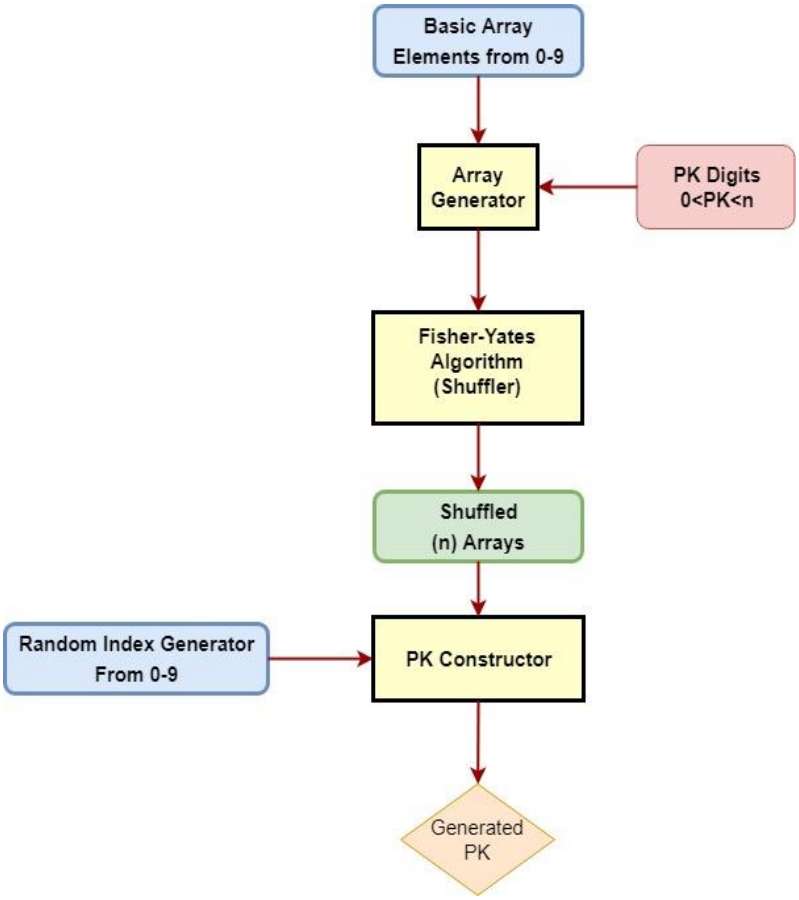

Figure 2. Random private key generation

4.2. Performing a mitigation technique against SCA while signing a document

Assume that we have an elliptic curve E_1 of order 19, where $E_1 = x^3 + 2x + 2 \; mod \; 17$. The points generated from the base point G = (5,1) are shown in table 1.

Table 1. All EC points including the neutral point

1P	(5,1)	11P	(13,10)
2P	(6,3)	12P	(0,11)
3P	(10,6)	13P	(16,4)
4P	(3,1)	14P	(9,1)
5P	(9,16)	15P	(3,16)
6P	(16,13)	16P	(10,11)
7P	(0,6)	17P	(6,14)
8P	(13,7)	18P	(5,16)
9P	(7,6)	19P	(infinity)
10P	(7,11)		

We can get 5P using two different ways.

1. Add G = (5, 1) to itself 5 times.
2. Convert 5 to binary form which is "0101" then we observe the next bit to the right of MSB and continue to reach the last bit (LSB). Finally apply the mathematical operation according to table 2.

Table 2. Mathematics representation in binary

0	1
Double Only	Double and Add

Table 3. Mathematical operation associated with each bit

Binary input (5)	**Steps**	**Calculations**	**# of steps**
0	negligible	**-**	**-**
1	We observe next bit to the right	**-**	**-**
0	Double only	**2P**	**1**
1	Double and Add	**2(2P)+P**	**2**
Total no. of steps = 3 steps			

Method (2) has fewer steps than method (1), and hence it is strongly considered when we deal with massive numbers. When the number 5 is converted into binary form the 0s and 1s can reveal the

mathematical operations to get the final point. By concatenation of those bits, one can construct the private key by appending 0s & 1s and convert them back to large decimal value. In figure 3, we propose a new method in the light of side-channel attack countermeasure to perform signing operations using an alternative technique but without changing the math behind elliptic curve algorithms (addition and multiplication). The private key is randomly generated with respect to curve order. We specify the number of digits allowed to form a large decimal number (private key, PK) which satisfies the following:

$$0 < PK < h \quad \text{where h is the curve order.}$$

Assume that we choose 5 digits. The software is then starting to shuffle one dimensional array of length 10 (from 0 to 9) 5 times and each time we have the next 10 different bits for 10 different private keys. For example, in the first iteration (i = 0), the MSBs for all private keys are created. In the last iteration index (i = 4), the LSBs for all private keys are created. Finally, we have 10 different private keys, each of them is 5 digits. The "fisher-yates" algorithm starts to choose random key from the set. The private key representation with 0s and 1s is defined by a random truth table as the system chooses whether the (Double only) and (Double & Add) are presented as "0" and "1", respectively or vice versa. This can be done by performing random complementary to bits many random times and the final results judge the bit representation (bit "1" may represent [Double only] or may represent [Double & Add]).

The idea behind this work is changing the truth table that represents which mathematical operation to perform. The attacker may append millions of bits as a side-channel attack, and then construct the private key however he/she does not know which operation "1" represents in this instance (Double only or Double & Add). The choice of double operation and double & add operation depends on random choice at runtime and hence this technique can countermeasure side-channel attack. Random binary complementary is used to randomly change the bitwise representation. "0" may represent "Double only" in an instance and in another instance may represent "Double and Add".

The single operation of doubling a point has approximately the same execution time as a single operation of adding a point. The "Double and Add" for a single point processing take delay time as "Double only", "Double and Add" combined. Table 4 illustrates the execution time of mathematical operations on point.

The only way to distinguish such current operation is to know two parameters:

1. Current bit wise representation ("0" represents "Double Only", "1" represents "Double and Add").
2. Bit wise itself.

Table 4. Average time for different mathematical operations

Mathematical operation on point	Average execution time (ms)
Doubling	0.0403654315349213
Adding	0.0405499200749979
Double and Add	0.0843728591517129

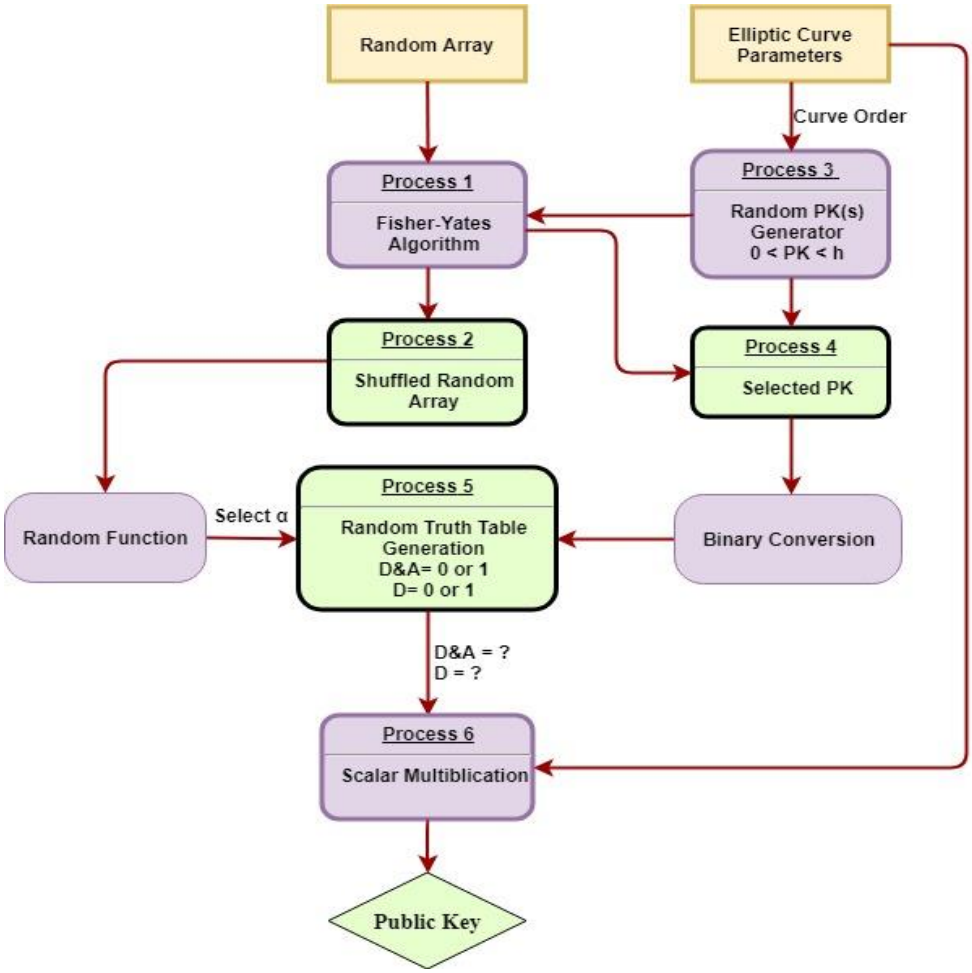

Figure 3. Secured generation of public key

The current bitwise representation is handled randomly by side channel countermeasure (SCC) model shown in figure 3. The private key is stored in binary format. If we have a PK = 5, it is stored in memory as 0101. The private key then can be constructed by sequences of ("Double only" + "Double and Add") respectively. Attackers may use one of the following two pieces of information to construct the PK:

1. integer binary representation.
2. execution time delay ("Double and Add" take more time than "Double only").

We overcome the first attack by SCC model, and we overcome the second attack as follows:

1. The randomly generated private key in string format is converted into a sequence of binary characters.
2. each character has a random mathematical representation which is equivalent to the original conventional PK.
3. We perform segmentation on string sequences for scaler multiplications.
4. The execution time delay may be equivalent to 0110 or 1001 or for the rest of its permutations as we don't use any integer representations.

By preventing PK of being stored in cache memory in either decimal or binary representation, the attacker cannot identify whether the current operation is point addition or point doubling. This will confuse the attacker as "Double and Add" operation may be thought as "Double and Double" or "Add and Add" because they have same execution time and PK has not any integer representation in cache memory. Also, the changes in the critical object cannot be monitored because it is stored as a volatile object [11, 12]. Moreover, we prevent the attacker from manipulating or even accessing the critical object address by applying thread lock to make the main thread exclusively access such memory address and after finishing all processes, we set the critical object value to null to be handled by the automated garbage collector then release the thread lock afterwards.

4.3. Use case implementation

Assume that Bob wants to transfer an amount of money to Alice. Bob has an account in Bank(A) which holds his personal information such as name, secret key and his biometric. Bob wants to transfer money to Alice so, he asks Alice about her account number which is represented by her finger print biometric. Bob has to send a message to his bank informing it to transfer money to Alice. He has to use a secured application which counteract SCA and perform the following steps:

1. Write the message including amount of money to transfer.
2. Take Alice account number (biometric).
3. Send the message with its signature to his bank.

The message is divided into segments of characters. Let the message Bob wants to send be "Transfer 1000$". The critical object in memory that hold the message is denoted by D and equal to {'T', 'r', 'a', 'n', 's', 'f', 'e', 'r', ' ', '1', '0', '0', '0', '$'}. Alice's biometric data is converted into hexadecimal form which is A= {'0xA', '0x3', '0x2', '0x2', '0x1', '0x3', '0xF', '0xB', '0x1', '0x3'}. The application performs mixing A and D based on Bob secret key (SK) which is stored safely in bank's database, for example SK= 14265. SK is converted into binary (S), so S = {'1', '1', '0', '1', '1', '1', '1', '0', '1', '1', '1', '0', '0', '1'}. The application scans Bob secret key in binary format (S) from MSB to LSB. Let M be an empty string, if the current bit is '1', then concatenate the first character from the message (‘T’) and the first hexadecimal digit of Alice biometric to M. If the current bit is '0', append the current message character to M, and so on. Therefore, the output is {'TAr3an2s2f1e3r F1B0100$3}. The expiration date might be added in the format {"dd":"mm":"yy"} to M. Then we convert M into a hash digest using SHA3-512bit hash algorithm. Finally, the hash digest is encrypted using ECDSA which we discussed earlier in section 2.4. The digital signature of the message content is "invalid" unless the bank has Bob SK and the receiving date of the sent message so the bank can authenticate the message properly.

5. Experimental results

We apply the proposed work to sign a document using different EC parameters. The proposed work has been implemented using Microsoft Visual C# Compiler version 4.6.1586.0, and personal computer with the following specs:

1. Processor: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
2. RAM: 16.0 GB.

Table 5 and table 6 illustrates the performance Analysis using different elliptic curves which are standard and recommended by NIST [8] without the countermeasure technique and with countermeasure technique respectively.

Table 5. Performance analysis using standard elliptic curves without SCC

Recommended EC by NIST	Selected PK	# of PK digits	Without SCC		
			Public key avg. calculation time (ms)	Signing Avg. calculation Time (ms)	Verifying Avg. calculation Time (ms)
secp192r1	Pk = n	58	15.58942214	Not applicable for PK = curve order n	
	random	54	12.33396952	12.7450389	28.0736595
secp224r1	Pk = n	68	22.30538886	Not applicable for PK = curve order n	
	random	64	18.29680023	18.6045201	38.0255423
secp256r1	Pk = n	78	28.15783606	Not applicable for PK = curve order n	
	random	74	23.17655532	23.7189029	53.3353985
secp384r1	Pk = n	116	73.43531671	Not applicable for PK = curve order n	
	random	112	59.83796577	61.5190690	128.077664
secp521r1	Pk = n	157	143.0427442	Not applicable for PK = curve order n	
	random	153	122.3799083	123.993550	249.852664

Table 6. Performance analysis using standard EC with SCC

Recommended EC by NIST	Selected PK	# of PK digits	Without SCC		
			Public key avg. calculation time (ms)	Signing Avg. calculation Time (ms)	Verifying Avg. calculation Time (ms)
secp192r1	Pk = n	58	15.68294931	Not applicable for PK = curve order n	
	random	54	12.62165712	12.8436590	12.8436590
secp224r1	Pk = n	68	22.60133937	Not applicable for PK = curve order n	
	random	64	18.49350565	18.9132195	18.9132195
secp256r1	Pk = n	78	28.38364508	Not applicable for PK = curve order n	
	random	74	23.44312591	24.2913875	24.2913875
secp384r1	Pk = n	116	74.06295258	Not applicable for PK = curve order n	
	random	112	60.49855714	63.2077771	63.2077771
secp521r1	Pk = n	157	145.8600671	Not applicable for PK = curve order n	
	random	153	123.0986465	124.897239	124.897239

Table 7 shows the time consumed to generate random private keys with respect to elliptic curve standards.

Table 7. PK average calculation time with different curves

Recommended EC by NIST	Random PK generation Avg. calculation Time (ms)
secp192r1	0.0599556987660316
secp224r1	0.071962462937078
secp256r1	0.0844619972092386
secp384r1	0.115521537058114
secp521r1	0.167619956467968

Using Intel VTune Amplifier 2019, we measured the cache memory bound and last level cache (LLC) miss count for the critical objects used for generating the random private key and for the object that holds PK binary string representation during the processes of public key generation, signing and verification. These results are illustrated in table 8.

Table 8. Volatile objects, no. of locks, memory bound and LLC miss count for different processes

	No. of volatile objects	No. of locks	Memory bound	LLC miss count
Random PK generation	1	1	0.9%	160,000
	0	0	45.6%	0
Public Key generation	1	1	18.5%	40,000
	0	0	31.2%	0
Signing	1	1	11.2%	80,000
	0	0	25.8%	0

Finally we Comparing our results to the related works in [9] and [10] with different types of elliptic curve domain parameters over a prime field F_P which is recommended by National Institute of Standards and Technology (NIST) [8].

Table 9. Comparison between related work in [9, 10] and proposed work.

Related work [9]					
Type of curve	**P-192**	**P-224**	**P-256**	**P-384**	**P-521**
Sign generation (ms)	47	79	110	251	542
Sign Verification (ms)	516	689	876	2123	4454
Proposed work					
Type of curve	**P-192**	**P-224**	**P-256**	**P-384**	**P-521**
Sign generation (ms)	12.843	18.913	24.291	63.207	12.843
Sign Verification (ms)	28.77	39.417	51.69	131.07	28.77
Related work [10]					
Type of curve	**P-192**	**P-224**	**P-256**	**P-384**	**P-521**
Sign generation	N/A	5.27 (μs)	10.03 (μs)	5.77 (μs)	N/A
Sign Verification (ms)	N/A	26.6	29.9	44.7	N/A
Proposed work					
Type of curve	**P-192**	**P-224**	**P-256**	**P-384**	**P-521**
Sign generation	N/A	18.913 (ms)	24.291 (ms)	63.207 (ms)	N/A
Sign Verification (ms)	N/A	39.417	51.69	131.07	N/A

6. Conclusion

In this work, we propose a method for creating secure digital signature that can countermeasure cache memory side-channel attacks. To achieve our goal, we use three techniques, first, we use "Fisher Yates" algorithm to randomly generate 10 private keys then randomly select one of them. Second, we use volatile memory objects to hold critical data and hence it is unlikely for attackers to get access to them. Third, we use thread locking so that only one thread (the administrator thread) can access the volatile objects while they are processed in cache memory, again this prevents attacker from accessing these critical objects. We analyze the performance of our proposed method using different parameters (the elliptic curve order and the number of private key digits) before and after applying SCC. Also, we make a comparative study between our proposed method and the related work against different elliptic curve domain parameters.

References

[1] D. Kleidermacher and M. Kleidermacher, Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development. Elsevier Science, 2012.
[2] A. J. Menezes, Elliptic Curve Public Key Cryptosystems. Springer US, 2012.

[3] H. Jahankhani, K. Revett, and D. Palmer-Brown, Global E-Security: 4th International Conference, ICGeS 2008, London, UK, June 23-25, 2008, Proceedings. Springer Berlin Heidelberg, 2008.
[4] L. C. Washington, Elliptic Curves: Number Theory and Cryptography, Second Edition. CRC Press, 2008.
[5] D. Hankerson, Guide to Elliptic Curve Cryptography. 2006.
[6] G. Wang, I. Ray, D. Feng, and M. Rajarajan, Cyberspace Safety and Security: 5th International Symposium, CSS 2013, Zhangjiajie, China, November 13-15, 2013, Proceedings. Springer International Publishing, 2013.
[7] D. Husemoller, Elliptic Curves. Springer New York, 2013.
[8] N. Telephone, T. Corporation, T. Kobayashi, A. Nagai, and W. Draft, "STANDARS FOR EFFICIENT CRYPTOGRAPHY SEC X . 2 : Recommended Elliptic Curve Domain Parameters," vol. 2, no. Sec 2, 2008.

[9] S. Manickam and D. Kesavaraja, "Secure multi server authentication system using elliptic curve digital signature," Proc. IEEE Int. Conf. Circuit, Power Comput. Technol. ICCPCT 2016, pp. 0–3, 2016.
[10] J. R. Shaikh, M. Nenova, G. Iliev, and Z. Valkova-Jarvis, "Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained E-commerce applications," 2017 IEEE Int. Conf. Microwaves, Antennas, Commun. Electron. Syst. COMCAS 2017, vol. 2017–Novem, pp. 1–4, 2018.
[11] R. C. Seacord, The CERT®C Coding Standard, Second Edition: 98 Rules for Developing Safe, Reliable, and Secure Systems. Pearson Education, 2014.
[12] G. R. Gao, D. Qian, X. Gao, B. Chapman, and W. Chen, Network and Parallel Computing: 13th IFIP WG 10.3 International Conference, NPC 2016, Xi'an, China, October 28-29, 2016, Proceedings. Springer International Publishing, 2016.