http://doi.org/10.35784/iapgos.897

ELLIPTIC-CURVE CRYPTOGRAPHY (ECC) AND ARGON2 ALGORITHM IN PHP USING OPENSSL AND SODIUM LIBRARIES

Mariusz Duka
Silesian University of Technology, Faculty of Automatic Control, Electronic and Computer Science, Gliwice, Poland

Abstract. *This paper presents the elliptic-curve cryptography (ECC) and Argon2 algorithm in PHP using OpenSSL and Sodium cryptographic libraries. The vital part of this thesis presents an analysis of the efficiency of elliptic-curve cryptography (ECC) and the Argon2 hashing algorithm in the Sodium library, depending on the variation of initiation parameters.*

Keywords: elliptic-curve cryptography, ECC, OpenSSL, Argon2, Sodium

KRYPTOGRAFIA KRZYWYCH ELIPTYCZNYCH (ECC) I ALGORYTM ARGON2 W JĘZYKU PHP Z WYKORZYSTANIEM BIBLIOTEKI OPENSSL I SODIUM

Streszczenie. *Celem niniejszej pracy jest analiza wydajności kryptografii krzywych eliptycznych (ECC) i wskazanie optymalnej krzywej dla kryptosystemu wykorzystującego język PHP wraz z biblioteką OpenSSL, a także analiza wydajności algorytmu haszującego Argon2, wchodzącego w skład biblioteki Sodium, w zależności od zmienności parametrów inicjujących.*

Słowa kluczowe: kryptografia krzywych eliptycznych, ECC, OpenSSL, Argon2, Sodium

Introduction

The information security, its storage and dissemination is a crucial element of any IT system, regardless of what data the system processes and on what technology it is based on. The Internet users of these IT systems expect the information they provide, especially sensitive information, to be adequately protected.

Currently available programming languages, such as Java, C, Python, Ruby, or PHP are used to build modern IT systems. The choice for a particular programming language depends on many factors, such as the requirements set for programmers, available programming libraries, technical support or its popularity at a time.

The scripting programming language, PHP, was designed to create web applications and generate websites on the server side. For the last 20 years PHP has been extensively developed and is currently the leader (almost 80% [4]) amongst the programming languages, on basis of which all websites are built. The popularity of the PHP language among the programmers around the world is enormous, as evidenced by the ninth position in the TIOBE index [5] (October 2019) and the sixth position in the SPECTRUM [6] ranking (2018). The PHP potential has also been recognised by such Internet giants as Google, Facebook, Yahoo!, Wikipedia or WordPress.

When designing a PHP based IT system to process information, a web developer is equipped with a number of modules and features to facilitate the information security, e.g. the OpenSSL library, that also uses, cryptography of elliptic curves or currently, the RSA algorithm. With PHP 7.2 version, the Sodium cross – platform library and the new Argon2 hashing algorithm are available to programmers. It makes PHP language a safe environment that implements the latest cryptography solutions.

This paper presents the findings related to the efficiency of generating cryptographic key using a cryptography of elliptic curves and efficiency of the Argon2 algorithm, depending on the variability of initiating parameters.

The conducted experiments related to the elliptic curve cryptography in IT systems based on PHP 7.2 were aimed at analysing which elliptic curves were the most efficient and could be recommended for using in the production environment.

1. Elliptic – Curve Cryptography (ECC)

Since 1985, the theory of ecliptic curves over finite bodies has been applied to several cryptographic issues, such as the distribution of integers into prime factors, primality tests, and the construction of cryptosystems. One of the main reasons for the interest in the cryptology in elliptic curves is that they are the source of a vast number of finite groups equipped with a rich additional algebraic structure. The groups of elliptic curve points are in many aspects similar to multiplicative finite body groups. However, they have two advantages over them – there are more of them and it seems that they provide the same degree of security with a short key length [2, 3].

The cryptosystems using elliptic curves were first suggested in 1985 by Victor Miller and Neal Koblitz. Initially, they did not expect that their idea would be used in practical terms, in any case, not earlier than in the distant future. Today, a few years after they had submitted their project, many useful implementations have been in use [2].

Table 1. NIST recommendations for the public key length

	Recommended minimum length (in bytes) of the public key			Ratio
Security level in bytes	DSA	RSA	ECC	ECC to RSA/DSA
80	1024	1024	160–223	1:6
112	2048	2048	224–255	1:9
128	3072	3072	256–383	1:12
192	7680	7680	384–511	1:20
256	15360	15360	512+	1:30

The Table 1 shows the recommendations of the National Institute for Standards and Technology (NIST) regarding the public key length. As we can see, the conventional keys of 1024-byte and 2048-byte size (value often found in the case of RSA) corresponds to cryptosystems based on elliptic curves with 160-223-byte and 224-255-byte key sizes, respectively. In a fair comparison, the complexity of implementing the cryptosystem should also be taken into account. In practice, however, shorter keys can translate into faster implementations, lower energy consumption, smaller surfaces of silicon, etc. [1].

1.1. The optimal elliptic curve for the cryptosystem

The use of elliptic curves increases the efficiency of cryptographic systems compared to other public key systems, such as RSA or Diffie – Hellman. It takes into account the difference in the key length (Table 1), while maintaining the same level of security. The selection of optimal elliptic curves requires measuring the generation time, taking into account the key length for the individual types of curves recommended by the international organisations, i.e. Brainpool curves (according to Brainpool), Prime, C2pnb, C2tnb (according to ANSI), Secp and Sect curves (according to SECG). As a result, Prime elliptic curves, which achieved a ratio of 2.03%, surpassed all other curves in terms of efficiency.

Table 2. Elliptic curves available in PHP

Name of the organization	Elliptic Curve Name	The length of key in bytes
Brainpool Group	brainpoolP160r1	160
	brainpoolP160t1	160
	brainpoolP192r1	192
	brainpoolP192t1	192
	brainpoolP224r1	224
	brainpoolP224t1	224
	brainpoolP256r1	256
	brainpoolP256t1	256
	brainpoolP320r1	320
	brainpoolP320t1	320
	brainpoolP384r1	384
	brainpoolP384t1	384
	brainpoolP512r1	512
	brainpoolP512t1	512
ANSI	prime192v1	192
	prime192v2	192
	prime192v3	192
	prime239v1	239
	prime239v2	239
	prime239v3	239
	prime256v1	256
ANSI	c2pnb163v1	163
	c2pnb163v2	163
	c2pnb163v3	163
	c2pnb208w1	208
	c2pnb272w1	272
	c2pnb304w1	304
	c2pnb368w1	368
ANSI	c2tnb191v1	191
	c2tnb191v2	191
	c2tnb191v3	191
	c2tnb239v1	239
	c2tnb239v2	239
	c2tnb239v3	239
	c2tnb359v1	359
	c2tnb431r1	431
SECG	secp112r1	112
	secp112r2	112
	secp128r1	128
	secp128r2	128
	secp160k1	160
	secp160r1	160
	secp160r2	160
	secp192k1	192
	secp224k1	224
	secp224r1	224
	secp256k1	256
	secp384r1	384
	secp521r1	521
SECG	sect113r1	113
	sect113r2	113
	sect131r1	131
	sect131r2	131
	sect163k1	163
	sect163r1	163
	sect163r2	163
	sect193r1	193
	sect193r2	193
	sect233k1	233
	sect233r1	233
	sect239k1	239
	sect283k1	283
	sect283r1	283
	sect409k1	409
	sect409r1	409
	sect571k1	571
	sect571r1	571

The experiment was carried out on a Dell PowerEdge 1950 server – with a typical workday load (Web server), with the following parameters:

- processor – Intel Xeon E5430 @2.66 GHz, 8CPU; 8GB RAM;
- operating system – FreeBSD 11.1 – RELEASE-p10;
- Software:
  - PHP 7.2; php72-openssl-7.2.5;
  - OpenSSL 1.0.2k.

Cryptography based on elliptic curves, i.e. Brainpool, Prime C2pnb, Secp and Sect, is available in PHP since its 7.1 version, and the OpenSSL library. There are many types of elliptical curves recommended by the international standardisation organisations or communities. The American National Standard Institute (ANSI) recommends Prime, C2pnb, C2tnb elliptic curves, Standard for Efficient Cryptography Group (SECG) recommends Secp and Sect elliptic curves or Brainpool Group, which recommends Brainpool elliptic curves.

1.2. The course of the experiment

In order to measure the efficiency of the key generation by using elliptic curves, a PHP 7.2 programming language, using the OpenSSL cryptographic library, has been developed. The experiment involved the measurement of time to generate 100 keys for each elliptic curve separately, during a typical operation of the production server.

The efficiency, marked as a performance ratio and expressed as a percentage for each elliptic curve, was calculated by dividing the average generation time by the key length of the elliptic curve. At the end of the experiment, the key performance for each group of elliptic curves was determined by dividing the performance ratio by the number of elliptic curves for that group.

1.3. The outcome of the experiment

We can conclude, from the data in Tables 3-8, that the larger the key size, the more time it takes to generate it. Specifically for curves with the key length of 384-511 bytes, the time needed to generate it increases significantly, e.g. for the elliptic curve sect571r1 the time needed was almost 56 milliseconds. PHP Group, which manages the implementation and development of the PHP programming language, based on the NIST recommendation, suggests the use of elliptic curves prime256v1 (NIST P-256) and secp384r1 (NIST P-384).

Table 3. Average performance ratio for Brainpool elliptic curves

Elliptic Curve Name	The length of key in bytes(A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
brainpoolP160r1	160	4.03	2.52
brainpoolP160t1	160	3.95	2.47
brainpoolP192r1	192	4.38	2.28
brainpoolP192t1	192	4.30	2.24
brainpoolP224r1	224	5.26	2.35
brainpoolP224t1	224	5.10	2.28
brainpoolP256r1	256	5.68	2.22
brainpoolP256t1	256	5.50	2.15
brainpoolP320r1	320	7.36	2.30
brainpoolP320t1	320	7.01	2.19
brainpoolP384r1	384	9.70	2.53
brainpoolP384t1	384	9.17	2.39
brainpoolP512r1	512	14.40	2.81
brainpoolP512t1	512	13.46	2.63
Average ratio in %			**2.38**

Table 4. Average performance ratio for Prime elliptic curves

Elliptic Curve Name	The length of key in bytes(A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
prime192v1	192	4.20	2.19
prime192v2	192	4.21	2.19
prime192v3	192	4.21	2.19
prime239v1	239	5.18	2.17
prime239v2	239	5.18	2.17
prime239v3	239	5.19	2.17
prime256v1	256	2.85	1.11
Average ratio in %			**2.03**

Table 5. Average performance ratio for C2pnb elliptic curves performance

Elliptic Curve Name	The length of key in bytes (A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
c2pnb163v1	163	5.85	3.59
c2pnb163v2	163	5.84	3.58
c2pnb163v3	163	5.83	3.58
c2pnb176v1	176	5.79	3.29
c2pnb208w1	208	6.74	3.24
c2pnb272w1	272	12.15	4.47
c2pnb304w1	304	13.61	4.48
c2pnb368w1	368	16.93	4.60
Average ratio in %			**3.85**

Table 6. Average performance ratio for C2tnb elliptic curves

Elliptic Curve Name	The length of key in bytes(A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
c2tnb191v1	191	6.23	3.26
c2tnb191v2	191	6.19	3.24
c2tnb191v3	191	6.18	3.23
c2tnb239v1	239	7.51	3.14
c2tnb239v2	239	7.49	3.13
c2tnb239v3	239	7.45	3.12
c2tnb359v1	359	16.02	4.46
c2tnb431r1	431	26.38	6.12
Average ratio in %			**3.71**

Table 7. Average performance ratio for Secp elliptic curves

Elliptic Curve Name	The length of key in bytes(A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
secp112r1	112	3.29	2.94
secp112r2	112	3.30	2.95
secp128r1	128	3.36	2.62
secp128r2	128	3.38	2.64
secp160k1	160	4.02	2.51
secp160r1	160	3.92	2.45
secp160r2	160	3.92	2.45
secp192k1	192	4.35	2.27
secp224k1	224	5.22	2.33
secp224r1	224	4.99	2.23
secp256k1	256	5.66	2.21
secp384r1	384	8.94	2.33
secp521r1	521	16.92	3.25
Average ratio in %			**2.55**

Table 8. Average performance ratio for Sect elliptic curves

Elliptic Curve Name	The length of key in bytes(A)	Average time to generate keys (in milliseconds) (B)	Ratio in % (B/A)
sect113r1	113	3.48	3.08
sect113r2	113	3.47	3.07
sect131r1	131	4.81	3.67
sect131r2	131	4.88	3.72
sect163k1	163	5.66	3.47
sect163r1	163	5.81	3.57
sect163r2	163	5.83	3.58
sect193r1	193	6.56	3.40
sect193r2	193	6.55	3.39
sect233k1	233	7.14	3.07
sect233r1	233	7.36	3.16
sect239k1	239	7.09	2.97
sect283k1	283	12.24	4.33
sect283r1	283	12.99	4.59
sect409k1	409	24.22	5.92
sect409r1	409	26.57	6.50
sect571k1	571	50.14	8.78
sect571r1	571	55.96	9.80
Average ratio in %			**4.45**

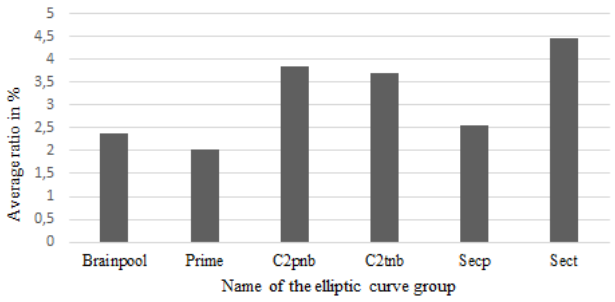

Fig. 1. Average performance ratio for all analysed groups of elliptic curves

The interesting conclusion of the experiment was the demonstration that Brainpool curves do not differ in efficiency from those recommended by the PHP Group as the most efficient elliptic curves Prime and Secp, and are in the second position between Prime and Secp in the ranking of average performance.

2. Argon2 algorithm

The Argon2 algorithm became the winner in the contest organised by the Password Hashing Competition [7], lasting from 2013 to 2015. The idea of the contest was to select a hashing algorithm that would be in line with the requirements of the National Institute for Standards and Technology (NIST) in the United States, for AES and SHA-3 algorithm, but more effective and sophisticated. Of the 24 projects submitted, it was the project by Alex Biryukov, Daniel Dinu and Dmitry Khovratovich from the University of Luxembourg in Luxembourg that won the most recognition. The algorithm has been made available on the Internet [7] in the form of the source code in C language.

Argon2 protects against brute-force attacks by using a predefined memory size, GPU usage time and an appropriate degree of Instruction Level Paralleism. It uses three parameters to control: the memory required, execution time and the number of threads used. There are two variants of this algorithm: Argon2i and Argon2d. Argon2i is effective against side-channel attacks because it uses data independent access to memory, which is why it is well suited for password hashing. Argon2d provides better protection against GPU-based attacks.

The Argon2i hashing algorithm has been implemented in PHP, starting from the 7.2 version, as part of the Sodium library. A range of Argon2 algorithms is used in tasks requiring massive storage capacity. It is optimised for x86 architecture and works efficiently on older processors. The key feature of Argon2 is the simultaneous use of multiple processor cores, which, consequently, is an additional protection that prevents a time/memory/date/trade off attack type of intrusion.

Argon2 uses optimally all available memory, Argon2i uses the memory at a speed of two processor cycles per byte, while Argon2d is three times faster. These features, among others, are included in the software used for cryptocurrencies. Argon2 is exceptionally scalable, both in terms of memory and CPU usage, because it can use up to 224 threads simultaneously.

2.1. The efficiency of Argon2 algorithm

The efficiency of Argon2 algorithm depends on the technical conditions in which it functions. Primarily, on the speed of the processor and the memory acquired. The performance evaluation will be basically to measure the algorithm's workload time dependent on the initialised parameters. The experiment is carried out on a Dell PowerEdge 1950 server – with a typical workday load (Web server), with the following parameters:

- processor – Intel Xeon E5430 @2.66 GHz, 8CPU; 8GB RAM;
- operating system – FreeBSD 11.1 – RELEASE-p10;
- Software:
  - PHP 7.2; php72-sodium-7.2.5;
  - Sodium library – libsodium – 1.0.16.

2.2. The course of the experiment

In the tcsh shell, the Argon2 program was cyclically called with parameters determining the memory acquired, the number of threads and the number of iterations, with the output key length set for all tests to 32 bytes.

The testing procedure had the following pattern:

echo -n "PSAEI_w_Gliwicach" | argon2 MySaltPSAEI -v 13 -m 14 -l 32 -i -t 1 -p 1

where MySaltPSAEI – the first parameter specifying grain (salt), v – algorithm version number (default 13), m – memory usage in kilobytes, calculated as 2^N, l – length of the shortcut at the output, i or d – Argon2i algorithm used or Argon2d, t – number of iteration, p – number of threads.

Figure 2 shows the results of a typical application of the Argon2 algorithm. As we can see, the algorithm's running time in a single iteration was 0.055 seconds, using 16MB of memory, with the number of threads being one.

Fig. 2. Application of the Argon2 algorithm in the tcsh shell

The next stage of the experiment was the cyclical calling of the hash function from the PHP programming language, using the Argon2i algorithm. The number of iterations for each type of measurement was 10, with the results divided by the number of threads and the amount of the memory used.

Table 9. Performance test for Argon2d and Argon2i algorithms, on the systems (console) level.

Thread	Memory in MB	Argon2d (1 iteration) in seconds	Argon2i (3 iterations) in seconds
1	16	0.031	0.094
1	32	0.078	0.172
1	64	0.172	0.383
1	128	0.359	0.773
1	256	0.719	1.594
1	512	1.445	3.227
1	1024	2.914	6.469
2	16	0.039	0.125
2	32	0.109	0.219
2	64	0.211	0.453
2	128	0.445	0.914
2	256	0.914	1.828
2	512	1.828	3.688
2	1024	3.594	7.492
4	16	0.086	0.172
4	32	0.180	0.328
4	64	0.375	0.664
4	128	0.773	1.328
4	256	1.664	2.773
4	512	3.344	5.516
4	1024	6.805	11.172
8	16	0.164	0.258
8	32	0.414	0.516
8	64	0.633	0.984
8	128	1.297	2.156
8	256	2.648	4.461
8	512	5.148	9.078
8	1024	10.758	18.383

2.3. The outcome of the experiment

The results of the tests covering the Argon2d and Argon2i algorithms, performed from the systems (console) level, are presented in the Table 9.

The results of the tests covering the Argon2i algorithm, performed from the PHP language level, for one, three and six iterations are presented in the Table 10.

Table 10. Performance test for Argon2i algorithm in PHP.

Thread	Memory in MB	Argon2i (in seconds)		
		1 iteration	3 iterations	6 iterations
1	16	0.0404	0.0808	0.1613
1	32	0.0826	0.1840	0.3359
1	64	0.1690	0.3781	0.6922
1	128	0.3471	0.7737	1.4176
1	256	0.7076	1.5748	2.8905
1	512	1.4336	3.1983	5.8553
1	1024	2.9056	6.4867	11.8148
2	16	0.0318	0.0692	0.1349
2	32	0.0636	0.1267	0.2332
2	64	0.1279	0.2465	0.4410
2	128	0.2620	0.4949	0.8553
2	256	0.5318	0.9959	1.6977
2	512	1.0665	1.9977	3.3970
2	1024	2.1589	4.0598	6.8337
4	16	0.0324	0.0544	0.0918
4	32	0.0632	0.1043	0.1636
4	64	0.1270	0.2048	0.3116
4	128	0.2585	0.4082	0.6034
4	256	0.5235	0.8305	1.2202
4	512	1.0610	1.6618	2.4375
4	1024	2.1318	3.2923	4.9209
8	16	0.0311	0.0556	0.0807
8	32	0.0611	0.1020	0.1454
8	64	0.1223	0.2005	0.2804
8	128	0.2472	0.4016	0.5534
8	256	0.5051	0.7974	1.1246
8	512	1.0222	1.5885	2.2446
8	1024	2.0625	3.0723	4.5112

3. Conclusion

The first aim was to analyse the performance of the elliptic curve cryptography in a programming environment, using the PHP 7.2 version and the OpenSSL library. The findings indicate that the optimal elliptic curve to use in the research environment constructed in this way is the prime256v1 curve with a 1.11% efficiency factor, which corresponds to a security level of RSA 3072 bytes.

Table 11. The most efficient elliptic curves in respective groups (bytes)

Key length in bytes	Name of elliptic curve	System
160–223	prime192v1 (v2, v3)	ANSI
224–255	prime239v1 (v2, v2)	ANSI
256–383	prime256v1	ANSI
384–511	secp384r1	SECG
512+	brainpoolP512t1	Brainpool

In the 160 – 223 bytes group, the most efficient (with the same ratio) elliptic curves are prime192v1, prime192v2 and prime192v3, in the group of 224 – 255 bytes – prime239v1, prime239v2 and prime239v3, in the group of 256 – 383 bytes – prime256v1, in group 384 – 511 bytes - secp384r1, and in the 512+ bytes group – brainpoolIP512t1.

The second aim was to analyse the performance of the Argon2 hashing algorithm, which is part of the Sodium cryptographic library and utilised both from the system console and the PHP 7.2. version. The time required to perform the hashing operations in the console version are similar to those in the PHP version, with the increase in time and depending on the memory acquired for use by the Argon2 algorithm.

It is also important to note that the algorithm efficiency is largely influenced by the number of threads used. With 1 GB of memory used, six iterations and one thread, the time taken to perform this operation was almost 12 seconds. With four threads, it dropped by almost half and with eight threads it reached just over 4.5 seconds. Of course, the above mentioned example is only a limiting example of the Argon2 algorithm in the production environment, as it is charged with a heavy use of processor power and the memory. The use of 16MB or 32MB memory at three iterations and two threads is the optimal solution in terms of time and the use of hardware resources.

References

[1] Blake I., Seroussi G., Smart N.: Krzywe eliptyczne w kryptografii. WNT, Warszawa 2004.
[2] Koblitz N.: Algebraiczne aspekty kryptografii. WNT, Warszawa 2000.
[3] Koblitz N.: Wykład z teorii liczb i kryptografii. WNT, Warszawa 2006.
[4] https://w3techs.com/technologies/details/pl-php/all/all (available: 14.11.2019).
[5] https://www.tiobe.com/tiobe-index/ (available: 14.11.2019).
[6] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018 (available: 14.11.2019).
[7] https://password-hashing.net (available: 25.04.2019).

M.Sc. Eng. Mariusz Duka
e-mail: mariduk781@student.polsl.pl

PhD student at the Faculty of Automatic Control, Electronic and Computer Science of the Silesian University of Technology. Professionally associated with the IT industry since 1993, in particular in software design and its implementation. Author of many Web projects, including the ISOWQ (International Studies of Website Quality) ranking system. His scientific interests related to the issues of data exploring, concurrent programming and IoT.

http://orcid.org/0000-0001-5773-0459

otrzymano/received: 16.12.2019 *przyjęto do druku/accepted: 15.09.2020*