Software Tool for Ensuring Data Integrity and Confidentiality Through the Use of Cryptographic Mechanisms

Ruslana Aleksieieva¹, Andriy Fesenko¹, Andriy Dudnik^{1,2}, Yerlan Zhanerke³

¹Taras Shevchenko National University of Kyiv, 60 Volodymyrska str., Kyiv, 01601, Ukraine ²Interregional Academy of Personnel Management, 2 Frometivska str, Kyiv, 03039, Ukraine ³Almaty Technological University, 100 Tole bi str, Almaty, 050012, Kazakhstan

Abstract

Ensuring the security of online communication is paramount, and this can be achieved by implementing robust data protection measures such as encryption and electronic signatures. To further enhance the cryptographic strength of these measures, the symmetric block algorithm AES, combined with elliptic cryptography, can be utilized. Moreover, using ECDSA offers a level of security comparable to RSA but with the added benefit of a shorter key length, thereby improving overall efficiency.

This paper comprehensively presents the algorithms employed within the information security system, providing a detailed account of their implementation.

Keywords 1

Confidentiality, integrity, cryptography, digital signature, ECC, AES, ECDSA, cryptographic security.

1. Introduction

People rely heavily on technology and digital communications, which has led to an increasing need for secure information exchange.

Cryptography plays an essential role in ensuring the secure exchange of information. Encrypting data using cryptographic algorithms protects information from unauthorized access, interception, and modification. Cryptography also enables the assurance of data integrity and authenticity through the use of digital signatures and hash functions.

To achieve this, appropriate software is employed to elevate the level of data security. These software programs encompass cryptographic information security tools, which involve creating certificates or keys, encrypting and decrypting messages or files, and generating and verifying qualified digital signatures to maintain the integrity and authenticity of documents.

2. Analysis of cryptographic mechanisms

The article titled "Comparative analysis of block key encryption algorithms" examines the comparative characteristics of block encryption algorithms (DES, 3-DES, AES, IDEA, Blowfish) based on criteria such as architecture, scalability, security, flexibility, and resistance to attacks. Based on the study, the AES and Blowfish algorithms outperform other algorithms in most aspects. For instance, DES and 3-DES exhibit lower security levels. IDEA, on the other hand, lacks flexibility and resistance to attacks.

ORCID: 0000-0001-5154-5324 (A. Fesenko); 0000-0003-1339-7820 (A.Dudnik); 0000-0002-3853-7239 (Y. Zhanerke)



^{© 2023} Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

MoMLeT+DS 2023: 5th International Workshop on Modern Machine Learning Technologies and Data Science, June 3, 2023, Lviv, Ukraine EMAIL: alexeeva02y@gmail.com (R. Aleksieieva); aafesenko88@gmail.com (A. Fesenko); a.s.dudnik@gmail.com (A.Dudnik); zhanerkeyerlan0@gmail.com (Y. Zhanerke)

CEUR Workshop Proceedings (CEUR-WS.org)

While Blowfish demonstrates relatively high security and performance indicators, it suffers from longer decryption time and weaker cryptanalysis resistance. AES, being a universal solution, compensates for its disadvantages with its overall advantages. Consequently, AES will be used in further research [1-3]

AES (or Rijndael) emerged as a robust replacement for DES during a competition conducted by the National Institute of Standards and Technology (NIST). Rijndael, the algorithm developed by Daemen and Rijmen, was recognized as the best and subsequently declared the new AES. NIST selected Rijndael due to its simplicity, high performance, fast execution, compactness, and straightforward mathematical structure. AES is a symmetric block cipher with a block size of 128 bits [4]

In the AES algorithm, the length of the input block, the output block and the State is 128 bits. For the AES algorithm, the length of the Cipher Key, K, is 128, 192, or 256 bits. The principle of the rounds is illustrated in Figure 1. [5] The AES algorithm employs a round function comprising four distinct byte-oriented transformations, shown in Figure 2:

- 1. SubBytes: nonlinear byte substitution using a substitution table (S-box);
- 2. ShiftRows: cyclically shifts the rows of the State array by a different number of bytes;
- 3. MixColumns: mixes data in each column of the state array;
- 4. AddRoundKey: adds a round key to the state using a simple bitwise XOR operation.







Figure 2: AES conversion

AES is a widely accepted standard in US government organizations, which attests to its high security and protection levels. However, the cryptographic strength of AES also relies on the complexity of the encryption key. Hence, in future research, we will consider enhancing the cryptographic strength of keys using elliptic cryptography.

3. Elliptic-curve cryptography

Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC allows smaller keys than non-EC cryptography (based on plain Galois fields) to provide equivalent security. Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme. [6]

The security of elliptic curve cryptography relies on the ability to compute multiplication by a point and the inability to compute the multiplicand given the starting and ending points of the product. The size of the elliptic curve, which is determined by the total number of discrete integer pairs satisfying the curve equation, determines the complexity of the problem. The main advantage offered by elliptic curve-based cryptography is a smaller key size, resulting in reduced storage and transmission requirements. [7] This means that an elliptic curve group can provide an equivalent level of security compared to an RSA-based system with a larger modulus and, consequently, a larger key size. For instance, a 256-bit elliptic curve public key should offer. A comparison of the cryptographic strength of these algorithms was considered in character 4. Elliptic curve scalar multiplication is the operation of successively adding a point along an elliptic curve to itself repeatedly. It is used in elliptic curve cryptography (ECC) to produce a one-way function. (Figure 3).[8]

Given a curve, E, defined by some equation in a finite field $(E: y^2 = x^3 + ax + b)$, point multiplication is defined as the repeated addition of a point along that curve. Denote as $nP = P + P + P + \cdots + P$ for some scalar (integer) n and a point P = (x, y) that lies on the curve, E. This type of curve is known as a Weierstrass curve. The security of modern ECC depends on the intractability of determining n from Q = nP given known values of Q and P if n is large (known as the elliptic curve discrete logarithm problem by analogy to other cryptographic systems). This is because the addition of two points on an elliptic curve (or the addition of one point to itself) yields a third point on the elliptic curve whose location has no immediately obvious relationship to the locations of the first two, and repeating this many times over yields a point nP that may be essentially anywhere. Reversing this process, i.e., given Q = nP and P, and determining n, can only be done by trying out all possible n - an computationally intractable effort if n is large.



Figure 3: Elliptic curve scalar multiplication

Create keys based on elliptical curves. Elliptic curve domain parameters over F_p are a sextuple: T = (p, a, b, G, n, h)

consisting of an integer p specifying the finite field F_p , two elements $a, b \in F_p$, specifying an elliptic curve $E(F_p)$, defined by the equation:

$$E: y^2 \equiv x^3 + ax + b(modp),$$

a base point $G = (x_G, y_G)$ on $E(F_p)$, a prime *n* which is the order of *G*, and an integer *h* which is the cofactor $h = E(F_p)/n$.

Elliptic curve domain parameters over F_p precisely specify an elliptic curve and base point. This is necessary to precisely define public-key cryptographic schemes based on ECC.[9,10]

Accordingly, upon completing the algorithm, a pair of keys should be obtained: a private key d (a randomly selected integer from the interval [1, n - 1]) and a public key on the elliptic curve Q = (xQ, yQ), represented by the point Q = dG. (Figure 4)



Figure 4: Create keys based on elliptical curves.

Points on an elliptic curve follow three standard rules: addition, doubling, and negation.

1. Negation of points

Rule to add two points with the same x-coordinates when the points are either distinct or have y-coordinate 0:

$$(x, y) + (x, -y) = 0$$

- i.e. the negative of the point (x, y) is -(x, y) = (x, -y). [6] The point negation algorithm is depicted in Figure 5.



Figure 5: The point of negation

2. Addition of points

Rule to add two points with different x-coordinates: Let $(x_1, y_1) \in E(F_P)$ and $(x_2, y_2) \in E(F_P)$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} (modp)$$
$$x_3 = \lambda^2 - x_1 - x_2 (modp)$$
$$y_3 = \lambda (x_1 - x_3) - y_1 (modp)$$

3. Doubling a point

If the points P and Q have the same coordinates, the addition process is slightly modified. Since there is no well-defined straight line through P, the operation is closed using the limit case, which involves considering the tangent to the curve E, at point P.

The calculation of this tangent is similar to the previous method, using the derivatives (dE/dx)/(dE/dy):

$$\lambda = \frac{3x_p^2 + a}{2y_p},$$

where a is derived from the defining equation of the curve E mentioned earlier.

The algorithm for adding and doubling the points of an elliptic curve is depicted in Figure 6.



Figure 6: Adding and doubling points on an elliptic curve

4. Digital signature

A *digital signature* is a mathematical technique utilized to verify the authenticity and integrity of a digital document, message, or software. It serves as the digital counterpart of a handwritten signature or stamped seal, providing enhanced security measures. Digital signatures address issues of tampering and impersonation in digital communications.

Electronic digital signatures are commonly employed to sign important electronic documents, and utmost care is taken to protect the confidentiality of sensitive data, which is an inherent attribute of every document. Therefore, improving the signature algorithm becomes a crucial aspect of electronic signatures. The utilization of elliptic curve-based cryptography is gaining traction due to the development of more efficient asymmetric cryptographic algorithms.

The practical requirements drive the advancement of asymmetric cryptography, presenting new challenges in information security and cryptographic algorithm design. [11]

The Elliptic Curve Digital Signature Algorithm (ECDSA) is an asymmetric public key algorithm specifically designed for creating digital signatures. While sharing a similar structure with DSA, ECDSA is defined over a group of points on an elliptic curve, rather than a ring of integers. [12] Like other asymmetric algorithms, ECDSA operates in a way that makes it easy to compute in one direction but significantly difficult to reverse. In the case of ECDSA, a number on the curve is multiplied by another number to generate a point on the curve. Even if the initial point is known, finding the resulting point is challenging.

The strength of the encryption algorithm relies on the discrete logarithm problem within a group of points on an elliptic curve. Unlike the discrete logarithm problem in a simple setting or the factorization problem in integers, there is no subexponential algorithm available for solving the discrete logarithm in a group of points on an elliptic curve. This characteristic leads to significantly higher "per-bit key security" in algorithms that employ elliptic curves. [13]

Comparison of ECDSA with RSA

RSA is one of the most widely used algorithms for creating digital signatures, relying on the computational complexity of factoring large integers.

Considering the popularity of RSA, an analysis is conducted to assess the efficiency and cryptographic strength of these algorithms. The security level of a symmetric key algorithm is directly linked to the chosen key size, meaning that larger keys offer greater security. However, the security level of an asymmetric key algorithm is lower than the key size employed. Notably, there is a considerable difference between the key size and the security level of various asymmetric key algorithms. Table 1 provides a comparison of security levels for different RSA and ECDSA key sizes. [6]

Cryptographic strength (in bits)	RSA public key length (bits)	ECDSA public key length (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Table 1

Comparison of key length concerning cryptographic strength

Compared to RSA, ECDSA offers increased security against modern cracking methods due to its inherent complexity. Despite providing an equivalent level of security, ECDSA employs shorter key lengths. Consequently, longer ECDSA keys require significantly more time to be cracked through brute force attacks.

Another advantage of ECDSA over RSA is performance and scalability. Because ECC provides optimal security with a shorter key length, it requires less computing power. This is useful for devices with limited memory and processing power.

Elliptic curve cryptography proves to be an effective approach to ensuring data integrity and confidentiality. It demonstrates resistance against attacks from quantum computers and exhibits higher processing speeds compared to traditional encryption methods. It also provides greater security when dealing with large amounts of data. [14]

The implementation of ECDSA involves three primary functions: key generation (previously discussed), creation of digital signatures, and verification of digital signatures. [15-17]

Digital signature generation

To sign a message *m*, an entity A does the following:

1) Select a random or pseudorandom integer $k \in [1, n - 1]$.

2) Compute k * g = (x, y) and r = xmodn. If $r \equiv 0modn$, then go to the step 1.

3) Compute $k^{-1}modn$ and $s = k^{-1}(hash + d * r)modn$, where hash is a value of the hash function of the message m to be signed. If s = 0, then the value $s^{-1}modn$ does not exist, so go to the step 1.

4) The result of the signing process is two integers r and s - a digital signature. The document signing algorithm is shown in Figure 7.





Digital signature verification. In order to verify the signature of sender A on a message, recipient B must do the following [13]:

- Get a copy of the public key Q of the sender A.
- Verify that r and s are integer in the interval [1, n 1] and compute hash value.
- Compute $u_1 = s^{-1}hash \mod n$ and $u_2 = s^{-1}r \mod n$.
- Compute $u_1g + u_2Q = (x, y)$.
- Accept the signature if and only if $x \mod n = r \mod n$.

The signature verification process is depicted in Figure 8.



Figure 8: Signature verification algorithm

5. Software implementation of cryptographic mechanisms

Based on the research data, the objective was to implement an enhanced AES encryption algorithm using elliptic cryptography key generation and an algorithm for signing and the ECDSA algorithm for signing and verifying electronic digital signatures. To achieve this, software was developed in the Python programming language.

Interface for creating certificates. The application's main window is illustrated in Figure 9. To create a certificate (a pair of elliptic curve keys), users can click on the "Create keys" button, which prompts a window to appear (Figure 10). In this window, users can enter their email and name, and additional settings allow them to choose the elliptic curve for key generation and the AES mode for subsequent encryption.

		•			ELL	ICE			
	File	e View Certificates	Account	settings					
	С	Create keys Encr		Decrypt		Verify	Import		
				(Certificates	Notepad)		
-		Name		Email	IsImp	orted	User-ID	к	ey-ID
	1	Alekseeva Ruslana	alexee	wa02y@gmail.c	True		c07318fd2bb279112	28 5RMv54iv	/HnvHTgjSe
	2	alexeeva02y@gmail.c	alexee	wa02y@gmail.c	True		c07318fd2bb279112	28 gZWD2z9	nDhybLsi5
	3	Ruslana test	alexee	wa02y@gmail.c	True		c07318fd2bb279112	28 4sBEqA8t	epWjeUKA
	4	Ruslana	alexee	wa02y@gmail.c	True		c07318fd2bb279112	28 dQX777nl	Hr7kowS3Y
	5	Ruslana	alexee	wa02y@gmail.c	False		c07318fd2bb279112	28 JsdQcjUT	XqytUqDaA
	6								
l									

Figure 9: The main window of the application

Implementation of the key generation algorithm based on elliptic cryptography. The implementation includes the ECCKeyGenerator class, which facilitates the generation of elliptic curve cryptographic keys. This class encompasses various methods, such as generating a public-private key pair, checking for a point on the curve, negating a point, adding two points, and scalar multiplication of an elliptic curve point.

Here is a description of each method:

- - __init__(self, private_key=None): Initializes a new instance of the ECCKeyGenerator class with an additional private key and an elliptic curve. If the private key is not provided, a random number is generated in the range of the curve order.
- - gen_keypair(self): Generates a public-private key pair using the instance's private key and the curve generator point. The algorithm for this process is depicted in Figure 4.
- $extended_gcd(k, p)$: Implements the extended Euclidean algorithm to calculate the greatest common divisor of k and p and the modular inverse of k to p. This method is utilized in the addition and scalar multiplication operations.

• - is on curve(self, point): Verifies if a given point lies on an elliptic curve.

• - point_neg(self, point): Negates a point on the curve by changing the sign of its y coordinate. Figure 5

• - add_point (self, point1, point2): Adds two points on the curve using the elliptic curve addition formulas. If either point is *None*, it returns the other point. If the *x*-coordinates of the points are the same, but their *y*-coordinates its, it returns *None*. Otherwise, it calculates the slope of the line between the points, determines the *x* and *y* coordinates of the resulting sum point, and returns them (see Figure 6).

• - scalar_mult(self, k, point): Performs scalar multiplication of a point on the curve by repeatedly adding the point to itself k times using the binary expansion k. This method takes two arguments: k (the scalar) and *point* (a point on the elliptic curve).

• If the scalar is zero or the point is *None*, it returns *None*. If the scalar is negative, the method recursively calls itself with the absolute value of the scalar and the negation of the point. The generated result is then included in a loop that performs a binary decomposition of the scalar. In each iteration, it checks the lowest bit of the scalar, and if it is 1, it adds it to the result using the add_point() method. It then doubles the addition by adding it and shifting the scalar bits to the right. This process continues until the scalar becomes zero, and the result is a point representing the public key (Figure 3).

In this window to create a certificate, you must enter mandatory data such as name and postal address. There is also a button for additional settings (Figure 13), where you can choose an elliptic curve for creating keys and, accordingly, the AES operation mode for encryption.

If you do not change additional settings, then by default the curve will be secp265k1, and the AES operating mode will be CBC.

) Ci	reating Certificate				
Enter Details						
Please enter personal details below.						
Name:	1					
E-mail:						
Additional settings:						
Choose	curve name:	secp256k1 🜔				
Choose AES mode: CBC						
Addi	tional settings	Cancel				

Figure 10: Additional certificate settings

Figure 11 demonstrates an example of a created certificate in an unencrypted format.



Figure 11: Certificate in unencrypted form

Implementation of AES. During the encryption process, a new AES object is created with the provided key, mode, and initialization vector values as function arguments. The input data is encoded, and the AES object is used to encrypt the encoded data, resulting in a ciphertext.

During decryption, the ciphertext is decoded, and a new AES object is created using the key, mode, and initialization vector values. A portion of the data is decrypted using the AES object and the specified block size. The decrypted text is returned as the output of the function.

The encryption and decryption algorithm operates by encrypting files and text using the generated keys. The process is illustrated in Figure 12.



Figure 12: Algorithm for creating certificates and, accordingly, encrypting and decrypting data

Interface for encrypting and decrypting files. Once the users have created a certificate, they can proceed to encrypt data. To encrypt a file, they need to follow these steps:

1. In the program's main window, select the previously created certificate by clicking on it. This step is illustrated in Figure 13.

2. After selecting the certificate, the "Encrypt" button becomes active. Click on the "Encrypt" button.

3. Choose the file that needs to be encrypted.

4. Upon successful encryption of the file, a message confirming the encryption process will be displayed, as shown in Figure 14.

After the file has been encrypted, it is possible to view the encrypted file. The encrypted file appears as a set of values and symbols, as depicted in Figure 15. It is important to note that the original file selected for encryption is not deleted or modified in any way. Instead, the program creates a new file with the encrypted information and adds the ".enc" extension to the file name. This ensures that both the original file and the encrypted version coexist separately.

•	● ● ● ELLICE								
F	-ile	e View Cer	tificates A	ccount settings					
Create keys Encrypt Decrypt		Sign	Verify	Import	Export	Share cert via Email			
					Certificates	Notepad)		
L.		Nar	ne	Email	IsImp	orted	User-ID	•	(ey-ID
ŀ	1	Alekseeva R	uslana	alexeeva02y@gmail.c	True		c07318fd2bb279112	28 5RMv54iv	/HnvHTgjSe
	2	alexeeva02y	@gmail.c	alexeeva02y@gmail.c	True		c07318fd2bb279112	28 gZWD2z9	nDhybLsi5
	3	Ruslana test		alexeeva02y@gmail.c	True		c07318fd2bb279112	28 4sBEqA8	tepWjeUKA
•	4	Ruslana		alexeeva02y@gmail.c	True		c07318fd2bb279112	28 dQX777nl	Hr7kowS3Y
	5	Ruslana		alexeeva02y@gmail.c	False		c07318fd2bb279112	28 JsdQcjUT	XqytUqDaA
	6	Ruslana		alexeeva02y@gmail.c	False		c07318fd2bb279112	28 hPf6qdtri	MJB9YVwYR
	7								
Γ									

Figure 13: Selecting a certificate for encryption



Figure 14: Successfully encrypted file

test >	≣ test123.asc.enc
	2T6/pV64Y5D1kg2v/rTaHouqKqmI019a6dWW5yIGqnfgh7qgclT77llWFWICuLuIuRrNZ/fLZm5W0Lu5AwX8X2PyILD6H6tRl0eIC9T5jMoSdQ9VnVHfFjzRcmSa4ZvwlHaGehm2e9atXt11aXA8x8Q5JpLuKYDfw

Figure 15: Encrypted file

The encryption algorithm is shown in Figure 12.

The decryption process follows similar steps to the encryption process and can be summarized as follows:

1. Select the previously created certificate that was used to encrypt the file and click on it in the main window.

2. Click on the "Decrypt" button.

- 3. Choose the encrypted file that you want to decrypt.
- 4. Upon successful decryption, you will receive a message confirming the decryption process.

It is important to note that using an incorrect certificate to decrypt a file may result in a failed encryption attempt, as indicated by the notification shown in Figure 16. Conversely, when the decryption process is successful, a corresponding message will confirm this. The decrypted file will be saved in the same location as the original encrypted document, as depicted in Figure 17.



Figure 16: Failure message





The decryption algorithm is shown in Figure 12.

Implementation of ECDSA. The implementation of ECDSA involves three main functions: hash_message(), sign_message(), and verify_signature(). These functions are used to obtain the message hash, create a signature, and verify the signature, respectively. It is assumed that the keys used in the code have been generated earlier.

The hash_message() function takes a message as input and uses the SHA-512 algorithm to hash the message, resulting in a 64-byte hash (512 bits). This hash is then converted to an integer e using the int.from_bytes() function with the parameter 'big', which reads the bytes in a large byte order (most significant bytes first) The resulting integer e is then reduced by a bitwise shift, shifting it by the

difference between the bit length of e and the bit length of the curve parameter n (the curve modulus). The resulting number becomes the message hash value.

The sign_message() function implements the process of signing a message using elliptic cryptography. It first hashes the message using the hash_message() function, resulting in the message hash value. The function then enters a loop that continues until valid values of r and s are generated. In each iteration, a random value k is generated and used to calculate the coordinates (x, y) of a point on the elliptic curve using the scalar_mult() function. The r value is calculated as x modulo n, and the s value is calculated using a formula specified in Figure 7.

The output of this function is the values of r and s, which can be sent along with the message as the signature.

The verify_signature() function verifies the authenticity of a signature. It first calculates the message hash using the hash_message() function. Then, it extracts the values of r and s from the signature. The function calculates w, which is the multiplicative inverse of s modulo n, and also calculates intermediate values u1 and u2 used in the verification calculation. Using the the scalar_mult() and add_point() functions, the function calculates the (x, y) points on the elliptic curve, derived from the public key g and the intermediate values u1 and u2. The function compares the calculated value of x with the value of r from the signature. If they match, the function returns "signatures match", indicating that the signature is valid. Otherwise, the function returns "invalid signature". The algorithm is illustrated in Figure 8.

Interface for signing and verifying documents. The ELLICE application allows users to securely sign and verify documents electronically using the ECDSA algorithm, as detailed in Section 4. To generate an electronic signature, users can easily follow the instructions provided within the app's interface:

- 1. Create a certificate and select it
- 2. Click the "Sign" button
- 3. Select a document to sign
- 4. Receive a message confirming the successful signing of the document. The signing process is shown in Figures 18-20.



Figure 18: File to sign

Name	Size	Date Added	∽ Kind
test123.asc	Zero bytes	3 бер. 2023 р., 19:10	Documer
test_cert_1234.asc	408 bytes	3 бер. 2023 р., 18:18	Documer
new_test_cert.asc	279 bytes	3 бер. 2023 р., 15:24	Documer
Certificate_to_alex02y@gmail.com.asc	408 bytes	3 бер. 2023 р., 15:16	Documer
Certificate_to_alexgmail.comsec.asc	344 bytes	3 бер. 2023 р., 15:16	Documer
text_cert_with_passphrase.asc	408 bytes	3 бер. 2023 р., 12:02	Documer
🔤 results.doc	7,9 MB	23 лют. 2023 р., 14:20	Microst
🗈 usercert.csv	26 KB	30 жовт. 2022 р., 20:25	CSV Doc
🔜 text.txt	26 bytes	11 жовт. 2022 р., 10:00	Plain Text
📼 ПР5.pptx	1,4 MB	11 жовт. 2022 р., 09:59	PowerP
📄 ecdsa.py	2 KB	3 лист. 2022 р., 11:12	Python se
📄 config.py	414 bytes	30 жовт. 2022 р., 18:00	Python s
New Folder		Cancel	Open

Figure 19: Selecting a file to sign



Figure 20: Message about the successful signing of the document

After signing, a signature file can be generated and used to verify the integrity of the document. This signature file should be transmitted along with the document and checked upon receipt. The structure and content of the signature file can be referred to in Figure 21.



Figure 21: Signature file

The ECDSA algorithm is used to create a signature using the "sign" function and the "hash_message" function. The private key is utilized for signature creation, while the public key is required for verification. Therefore, it is crucial to prioritize the protection of the private key.

To verify a signed document, follow these steps:

- Select a certificate
- Click the "Verify" button
- Choose the document that contains the signature.

• After verification, a success message will display the signer's name. If the document has been modified, the signature will be deemed invalid.

The electronic signature verification process is illustrated in Figures 22-25.

Name	Size	Date Added	Kind
∽ 🚞 test		21 груд. 2022 р., 12:57	Folder
text.txt.sgn	156 bytes	Сьогодні, 12:36	Documen
test123.asc.enc	576 bytes	Сьогодні, 11:06	Documen
test123.asc	Zero bytes	3 бер. 2023 р., 19:10	Documen
test_cert_1234.asc	408 bytes	3 бер. 2023 р., 18:18	Documen
new_test_cert.asc	279 bytes	3 бер. 2023 р., 15:24	Documen
Certificate_to_alex02y@gmail.com.asc	408 bytes	3 бер. 2023 р., 15:16	Documen
Certificate_to_alexgmail.comsec.asc	344 bytes	3 бер. 2023 р., 15:16	Documen
text_cert_with_passphrase.asc	408 bytes	3 бер. 2023 р., 12:02	Documen
፼₌ results.doc	7,9 MB	23 лют. 2023 р., 14:20	Microst
🗈 usercert.csv	26 KB	30 жовт. 2022 р., 20:25	CSV Doci
🔜 text.txt	26 bytes	11 жовт. 2022 р., 10:00	Plain Text
TDE naty	1 / 1/0	11 WART 2022 5 00.50	DoworD
New Folder		Cancel	Open

Figure 22: Selecting the signature file to verify the signature



Figure 23: Receive a message about the signature match, which means that the file has not been modified



Figure 24: Change the text in the signed document.



Figure 25: notified that the signatures do not match.

6. Conclusion

In this article, we have explored the key aspects of ensuring data confidentiality and integrity by utilizing cryptographic mechanisms. Specifically, we have discussed the process of key generation using elliptic cryptography, presented an enhanced AES encryption algorithm, and outlined the ECDSA algorithm for digital signature creation. To practically implement these algorithms, a dedicated software application has been developed.

7. References

- [1] S. Mukta, R.B. Garg, Comparative analysis of block key encryption algorithms, International Journal of Computer Applications (2016), 145(7), 26–35. doi:10.5120/ijca2016910656.
- [2] M. Iavich, S. Gnatyuk, E. Jintcharadze, Y. Polishchuk, A. Fesenko and A. Abisheva, Comparison and Hybrid Implementation of Blowfish, Twofish and RSA Cryptosystems, Proceedings of 2019 IEEE 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON), Lviv, Ukraine, 2019, pp. 970-974.
- [3] M. Nehrey, I. Voronenko, A-B. M. Salem, Cybersecurity Assessment: World and Ukrainian Experience, 12th International Conference on Advanced Computer Information Technologies (ACIT), Ruzomberok, Slovakia, 2022, pp. doi:335-340. 10.1109/ACIT54803.2022.9913081.

- [4] Gary C. Kessler, An Overview of Cryptography, 2023. URL: <u>https://www.garykessler.net/library/crypto.html</u>
- [5] National Institute of Standards and Technology, Federal Information Processing Standards (FIPS) Publication 197-upd1, 2023. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197upd1.pdf.
- [6] The Case for Elliptic Curve Cryptography, 2009. URL: https://archive.ph/o/G7TQq/https://www.nsa.gov/business/programs/elliptic_curve.shtml%23ski ptopmenus.
- [7] M. Iavich, A. Gagnidze, G. Iashvili, S. Gnatyuk, V. Vialkova, Lattice based Merkle, CEUR Workshop Proceedings, Vol. 2470, pp. 13-16, 2019.
- [8] A. Corbellini, Elliptic Curve Cryptography: Elliptic Curve Cryptography: finite fields and discrete logarithms, 2015. URL: https://andrea.corbellini.name/2015/05/23/elliptic-curve-cryptography-finite-fields-and-discrete-logarithms.
- [9] Standards for Efficient Cryptography Group. URL: https://www.secg.org/sec1-v2.pdf
- [10] A. Razaque, V. Alexandrov, M. Almiani, B. Alotaibi, M. Alotaibi, A. Al-Dmour, Comparative Analysis of Digital Signature and Elliptic Curve Digital Signature Algorithms for the Validation of QR Code Vulnerabilities, in Proceedings of the 2021 Eighth International Conference on Software Defined Systems, SDS, IEEE, 2021, pp. 1-7.
- [11] E. Faure, A. Shcherba, Y. Vasiliu, A. Fesenko, Cryptographic key exchange method for data factorial coding, CEUR Workshop Proceedings, 2020, Vol.2654, pp. 643–653.
- [12] D. Hankerson, S. Vanstone, A. Menezes, Guide to Elliptic Curve Cryptography, Springer Professional Computing, New York: Springer. 2004, 311 p. doi:10.1007/b97644
- [13] S. Gnatyuk, Zh. Hu, V. Sydorenko, M. Aleksander, Yu. Polishchuk and Kh. Yubuzova, Critical Aviation Information Systems: Identification and Protection, Cases on Modern Computer Systems in Aviation, USA: IGI Global, pp. 423-448.
- [14] S. Gnatyuk, T. Okhrimenko, M. Iavich and R. Berdibayev, Intruder Control Mode Simulation of Deterministic Quantum Cryptography Protocol for Depolarized Quantum Channel, Proc. of 2019 Intern. Scien.-Pract. Conf. on the Problems of Infocommunications. Science and Technology, 2019, pp. 825-828.
- [15] D. Johnson, A. Menezes, S.Vanstone, The Elliptic Curve Digital Signature Algorithm (ECDSA), IJIS 1, 36–63 (2001). doi:10.1007/s102070100002.
- [16] N. Dakhno, O. Barabash, H. Shevchenko, O. Leshchenko, A. Dudnik, Integro-differential Models with a K-symmetric Operator for Controlling Unmanned Aerial Vehicles Using a Improved Gradient Method, Proceedings of 2021 IEEE 6th International Conference on Actual Problems of Unmanned Aerial Vehicles Development (APUAVD), Kyiv, Ukraine, 2021, pp. 61-65.
- [17] A. Dudnik, Y. Kravchenko, O. Trush, O. Leshchenko, N. Dakhno, V. Rakytskyi, Study of the Features of Ensuring Quality Indicators in Multiservice Networks of the Wi-Fi Standard, Proceedings of 2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, 2021, pp. 93-98.