ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 2, 2024

ENCRYPTING THE FILE SYSTEM ON A SINGLE-BOARD COMPUTERS PLATFORM AND USING LINUX UNIFIED KEY SETUP WITH PHYSICAL ACCESS KEYS

***Bohdan Onishchenko*[1], *Roman Banakh*[1], *Abdallah A. Z. A. Ibrahim*[2]**

[1]Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013, Ukraine,
[2]Suez Canal University, El Sheikh Zayed, El Salam District, Ismailia 8366004, Egypt.
Authors' e-mails*: bohdan.onishchenko.kb.2021@lpnu.ua, roman.i.banakh@lpnu.ua, abdallah.ibrahim@uni.lu*

https://doi.org/10.23939/acps2024.02.128

Submitted on 15.10.2024

***Abstract*: The object of the research is the security of the file system of a single-board platform. As part of the research reported in this paper, a method has been proposed to protect the file system using encryption. Implementing a Linux Unified Key Setup paired with a password or Universal Serial Bus key has been demonstrated. The advantages of Linux Unified Key Setup for this task and the possibilities for system configuration and encryption method depending on the use case and hardware configuration has been outlined. As a result, the administrator of a single-board computer can store and work with sensitive information in a more secure environment. This will allow the user to be sure that their private information will not be accessible in case of theft or attempted hacking of the device.**

***Index Terms*: Encryption, Algorithms, Access control Authentication, IOT, Data protection, Passwords, Cybersecurity**

I. INTRODUCTION

Single board computers do not encrypt the file system by default, but this allows a cybercriminal to read or adjust the logic in their favor in case of access. Among the proposed solutions, there are external encrypted USB (Universal Serial Bus) drives or SSDs (Solid-State Drive) that support hardware-based encryption, but apart from the price issue, the user cannot use these devices as a place to store system data, nor can they edit the logic of these devices. To solve the problem, we chose LUKS (Linux Unified Key Setup) [1—2] as one of the most popular solutions for Linux systems and is extremely flexible, in addition, LUKS supports a large number of encryption algorithms. Which will allow the administrator to choose the algorithm that will satisfy the conditions in which the device is planned to be used and rely on critical factors for the system – encryption/decryption speed, encryption reliability. LUSK uses modern key derivation functions such as Argon2 [3] and PBKDF2 (Password-Based Key Derivation Function) [4], which allow you to store user passwords reliably, and under such conditions that a conditional criminal will not be able to pick up the password, for a satisfactory period of time – when the information has value. Coupled with reliable encryption algorithms – AES (Advanced Encryption Standard) in XTS (XEX Tweakable Block Cipher with Ciphertext Stealing), CBC (Cipher Block Chaining) [5] mode, TwoFish, Serpent. Users can also create their own USB keys and use them to distribute access. So, this article demonstrates the possibility of integrating LUKS and Single board computers, describes the algorithms that the user can use to configure encryption and all the important points and problems that can arise when working with an encrypted crypto container. This method can be used as a complete solution, as well as a basis for more complex scenarios.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Article [6] describes the mechanism of LUKS for IoT systems, which allowed us to understand the principles of encryption at a lower level since the memory mechanism on most IoT systems differs from that of a desktop computer. Still, this article does not disclose the mechanism of implementation and integration of LUKS into the system. Articles [7—8] describe the mechanisms of the Argon2 and PBKDF2 hashing functions, which allowed us to compare them with each other, understand the mechanism of these functions, and choose a configuration that is more suitable for the end user and more appropriate for the proposed scenario. Articles [9] describe the AES encryption algorithm in various modes; this information allowed us to understand better the AES encryption algorithm and the advantages and disadvantages of various AES encryption configurations. In the article [10], the authors describe the bypassing of hashing, which became the basis for studying hash functions and their analysis for cryptographic strength. It also helped analyze brute force attacks on hash functions used in LUKS.

III. SCOPE OF WORK AND OBJECTIVES

Even though a lot of research has already been done, the problem remains in demonstrating the possibility of integrating encryption for single-board computers, respectively, if there is a need to ensure the protection of information on a single-board computer by encrypting the file system and the lack of a scientific and technical solution, the problem arises in the study of encryption tools, taking into account the features of the single-board computer platform, the reliability of modern encryption algorithms, taking into account the loss of memory speed and configuration.

IV. SINGLE-BOARD COMPUTERS AND LINUX ENCRYPTION SPECIFIED

Modern Linux distributions offer a wide range of encryption tools, which are convenient and valuable, but we need to choose the one that will meet certain conditions. Therefore, tools such as GPG (GNU Privacy Guard), Age, and OpenSSL (Secure Sockets Layer) are unsuitable in our scenario because they aim to encrypt individual files or directories, not the entire file system. So, users have a choice between VeraCrypt and LUKS [9].

A. LINUX UNIFIED KEY SETUP

Among others, the most popular solution is LUKS (cryptsetup), and its advantages include: standardized and widely supported; command-line based; high performance for full-disk encryption.

Minimal overhead due to being part of the Linux kernel Device-mappers crypt (dm-crypt) is a Linux subsystem for transparent data encryption at the block device level. It is a part of the Linux kernel and is used to encrypt disks or partitions, providing data protection in the event of physical loss or theft of the media.

These advantages are the reasons for choosing LUKS over other encryption tools.

B. COMPARISON OF LUKS1 AND LUKS2

To better understand which configuration the user must choose, let us compare LUSK1 and LUKS2 and the possible dependencies on the algorithm, length of the key, etc. The data for the table was taken from the LUKS specification. The comparison of LUKS1 and LUKS2 features presented in the Table 1. LUKS2 is a more advanced version with more features for system integration and key management. The general trend is to use LUKS1 on legacy devices and LUKS2 in all other cases (Table 2). Each encryption algorithm supported by LUKS2 has its limitations and is being used depending on the users' systems and hardware configurations (Table 3). The security of cryptographic algorithms is usually assessed by the number of operations required to find a key by brute force and resistance to specific types of attacks, such as linear cryptanalysis, differential cryptanalysis, side-channel attacks, and other methods.

Table 1

Comparative table of LUKS1 and LUKS2 features

Feature	LUKS1	LUKS2
Metadata format	Limited	Flexible and redundant
Key slots	8	8 with better flexibility
Integrity protection	Not applicable	dm-integrity
KDF Algorithm	PBKDF2	PBKDF2 and Argon2
Snapshot support	Limited	Yes
Device size	Small to medium sized	Scalable and larger sized
Backward compatibility	Not applicable	With LUKS1

Table 2

Comparative table of LUKS1 and LUKS2 algorithms

Algorithm	LUKS1	LUKS2
KDF	Limited	Flexible and redundant
Encryption modes	8	8 with better flexibility
Authenticated encryption	Not applicable	dm-integrity
Integrity protection	PBKDF2	PBKDF2 and Argon2

Table 3

Comparative table of LUKS2 encryption algorithms

Algorit hm	Type	Key Size	Block Size	Strength
AES	Symmetr ic block cypher	128, 192, 256 bits	128 bits	High (Widely considered secure with 256-bit key)
Serpent	Symmetr ic block cypher	128, 192, 256 bits	128 bits	High (Strong alternative to AES)
Twofis h	Symmetr ic block cypher	128, 192, 256 bits	128 bits	High (Considered slower than AES)
PBKD F 2	Key Derivatio n	Varia bles Iterati ons	-	Medium to High (Customizable iterations)
Argon2 id	Key Derivatio n	Varia bles Iterati ons	-	High (Memory-hard, resistant to GPU attacks)

The length of the key and the number of encryption rounds are also considered, which determine the complexity of the algorithm's cracking [10]. No known practical attacks on AES can significantly reduce the attack complexity below 2128 operations for AES-128 or 2256 for AES-256. Linear and differential cryptanalysis have shown limited success against simplified versions of the algorithm but not against the complete AES scheme.

AES is the default cipher in LUKS2, providing strong encryption with high performance, especially in Xor–encrypt–xor-based tweaked-codebook mode with ciphertext stealing (XTS) mode. Serpent and Twofish are alternative block ciphers in LUKS2, often chosen for different security or performance characteristics. The password-based key derivation function (PBKDF2) is the default essential derivation function in LUKS2, though Argon2id is considered a more secure and modern alternative. Secure Hash Algorithms (SHA), SHA-256, and SHA-512 are standard hash functions used for hashing and key derivation, with HMAC-SHA256 ensuring data integrity. To compare the performance of different algorithms and lengths of the key, we used the artificial option cryptsetup benchmark. The benchmark data was performed on Raspberry Pi 5 [11] 8 GiB RAM and 4 Class 8 Gib SD-Card (Fig. 1-4).

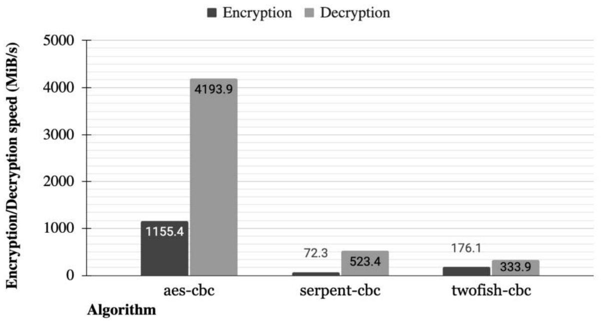


Fig. 1. Encryption and decryption speed using 128 bits key and CBC mode

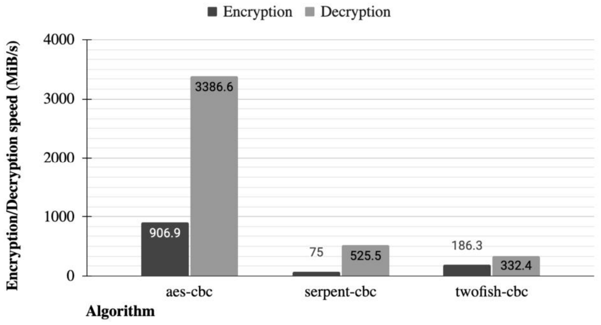
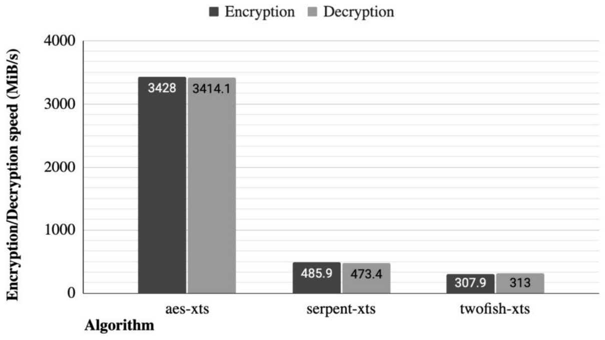


Fig. 2. Encryption and decryption speed using 256 bits key and CBC mode

This difference in read/write speeds between XTS mode and CBC mode is explained by the fact that XTS is mainly designed for disk encryption. At the same time, CBC is traditionally used for communication encryption and encryption of specific files.

Each mode has its own advantages, but XTS is more suitable for encrypting data, especially those stored in fixed-size sectors, as it is done on SD cards and SSDs. CBC uses chain encryption (each subsequent block of information depends on the previous one); that is, if there is a vulnerability to one of the blocks, it will affect the operation of other blocks. CBC mode is recommended for use with additional security methods (e.g., message authentication).



1) Fig. 3. Encryption and decryption speed using 256 bits key and XTS mode

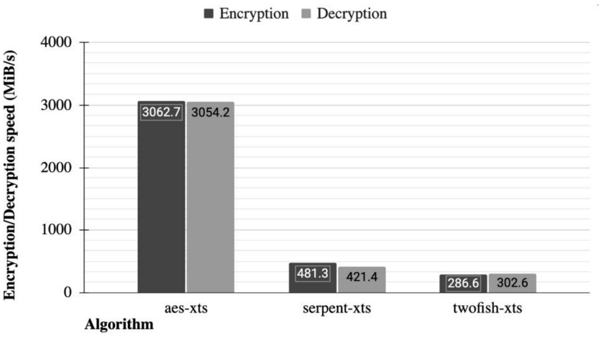


Fig. 4. Encryption and decryption speed using 512 bits key and XTS mode

After comparing, we conclude that performance depends on the algorithm, not the length of the key. Comparing CBC and XTS, we conclude that CBC loses in terms of encryption speed. Therefore, if the user is interested in the speed of encryption/decryption, then it is necessary to choose the AES algorithm. It is important to note that this benchmark was made on a single-board computer, Raspberry Pi 5, so the user should conduct benchmarks for their platform on their own, as the resistances may differ from the hardware.

C. SINGLE-BOARD COMPUTERS ENCRYPTION

Single-board computers are a popular IoT platform used for education, business solutions, and personal use. However, they still need ready-made solutions for encrypting the file system, which is critical if the user wants to store sensitive information with a certain level of confidentiality.

In this case, a conditional attacker can remove the SD card from the device and read all the data, so this research aims to fix this problem.

Full disk encryption is relatively easy to perform with modern Linux distributions. Single-board computers are an exception because the boot partition does not include most of the needed programs and kernel modules. Also, single-board computers do not have a

LiveCD, so it is impossible to encrypt the file system first and then install it.

V. SCENARIO OF THE PROBLEM SOLVING

Let us consider the following approach to solve the problem. Firstly, create an encrypted container in the file system to store sensitive information. Then, configure the system to automatically decrypt and mount the container if the user is legitimate. Finally, create a USB key that grants access to the encrypted container without requiring a password. In this case, a legitimate user who knows the password and/or has a USB key has the following options for working with the encrypted container, the system will wait for a USB key for 1 minute and 30 seconds. If the key is not found, the user will be promoted to enter a password. After the user has finished working with the encrypted container, in the case of a USB key, they simply remove it, and the container is encrypted and hidden in the disk hierarchy. If the user uses a password, he will need to either shutdown/reboot or use the command line to close the container.

VI. ABOUT USB KEY AND PASSWORD LENGHT

LUKS uses a key file as an additional method of user authentication. The key file can be any file, but there are better practices than this one. It is preferable to generate a personal file with random content, that cannot be easily generated. For this purpose, user can use a file with the extension .key or. lek in LUKS and fill them with random binary information. The user can apply any convenient tool to generate random binary information, preferably one that uses entropy to generate a random sequence. The most popular solutions are OpenSSL or the Linux module /dev/urandom. The latter uses the entropy of device to generate random data such as – mouse movements, keystroke timings, interval between interrupts from the hardware, timings of input and output operations, other random events at the kernel level.

A. PASSWORD LENGTH

The LUKS documentation specified that the password limit is found to be a 64-character. However, in practice, it works differently omitting the dependence between the limitations of the counter algorithm. If the user tries to enter a value of more than 64 characters, they will succeed, and it will work correctly. Regardless of the password length, it is evident that LUKS works only with a hash. Hash functions in LUKS are implemented through PBKDF2 or Argon2. Therefore, the user can use a password of any length, in any case, PBKDF2 or Argon2 will convert the encryption key to a hash in LUKS, the standard hash length is usually 256 bits (32 bytes).

VII. PROBABILITTY OF BRUTE FORCE

From a theoretical point of view [12], let us calculate the possibility of brute-forcing passwords of different lengths using (1):

$$T = N / S . \qquad (1)$$

For example, let us take a password of 8, 12, and 64 characters and use (1) to calculate the time it will take to achieve the goal. Firstly, let us calculate the password for an 8-character length which can be created from a combination of 62 different symbols. $N=62^{12}$, $T=62^{8}/10\cdot10^{5}\sim6.92$ years. Now, let us do the same using the length of 12 characters. $N=62^{12}$,$T=62^{12}/10\cdot10^{5}\sim102$ years, and the same using the length of 64 characters. $N=62^{64}$, $T=62^{64}/10\cdot10^{5}\sim5.16\cdot10^{118}$ years. For a more accurate comparison, let us compare the maximum time on two configurations – g5.4xlarge instance (1 GPU, 24GiB GPU memory, 16 vCPUs, 64 GiB memory) [13] and RTX6000 x4 for different hashing algorithms: MD5(Message-Digest), SHA1, SHA2-224, SHA3-224. This comparison is necessary to show the direct dependence of the password length on the time it takes to brute-force it, as well as the dependence of the brute-force attack on the computer's power. Therefore, the following conditions were chosen for comparison: a password of 8 characters, a password of 9 characters to show how even one character affects the statistics, and a password of 12 characters, the recommended length.

To understand the criticality of this issue, let us demonstrate the increase in time using passwords of 8 and 9 characters in percentage increase. Also, about the password format itself, it is essential, in addition to general recommendations on the type, not to use a one-syllable word or things that can be associated with the user and build a special brute-force dictionary based on these associations. The user password should contain uppercase and lowercase letters, numbers, and special characters. In our case, in the following sequence, a password was generated that contains 3 uppercase letters, 3 lowercase letters, and 2 special characters for a password of 8 characters (Fig 5).

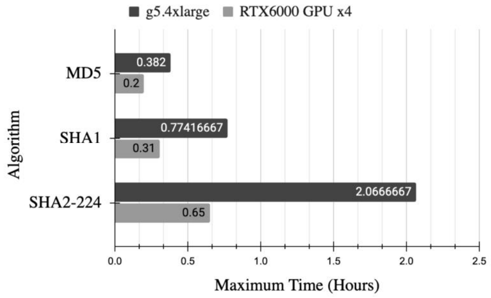


Fig. 5. Brute-force time for 8 password length g5.4xlarge and RTX6000 GPU x4 comparison

For a 9 characters password – 3 uppercase letters, 3 lowercase letters and 3 special characters (Fig 6).

For a 12-character password – 3 uppercase letters, 4 lowercase letters, 4 special characters and 1 number (Fig 7).

It is essential to understand that an attacker may have a much more powerful cluster than the user can expect. The process of guessing will be much faster, so if possible, users should use passwords that are as complex as possible. Although LUKS does not have a mechanism for timeout between password attempts (which would make it harder to crack a password), PBKDF2 spends more time on computation with each iteration, so in some cases, this mechanism makes brute-force attacks more difficult. By changing the number of iterations for PBKDF2 in the LUKS settings, the user can find the optimal values for the system and the delay between password requirements.

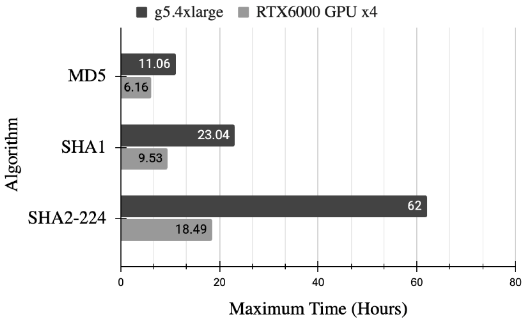


Fig. 6. Brute-force time for 9 password length on g5.4xlarge and RTX6000 GPU x4 comparison

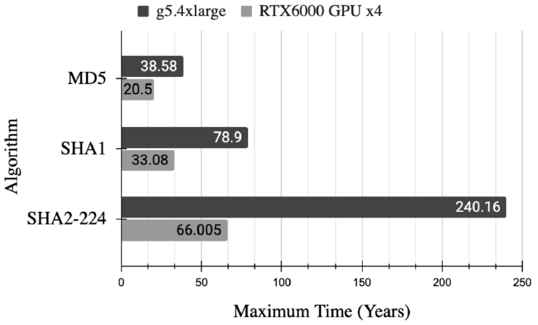


Fig. 7. Brute-force time for 12 password length on g5.4xlarge

Using passwords may not be convenient regarding transmission, storage, etc. Therefore, we recommend using two-factor authentication. The most common method is to use a password (the “knowledge” factor) paired with a critical file (the “possession” factor). This is another possible configuration option for LUKS, in which, even if the password is compromised, a cybercriminal cannot decrypt the content without a USB key with a key file. In addition to the standard LUKS tools, you can use additional modules, for example, for a Time-based one-time password (TOTP). This implementation is difficult but possible and will significantly increase security.

VIII. SYSTEM AND CRYPTSETUP CONFIGUTARION

After encryption and key file creation, you must focus on system configuration and cryptsetup. The main system file we will have to work with is /etc/fstab, which describes the configuration of disks and the rules that apply to them.

Example of a rule in fstab should be next:

```
UUID=xxxx-xxxx-xxxx /mnt/data ext4
defaults,noatime 0 2
```

The main points to be considered are the <options> <dump> <pass> items. Let us say about dump right away - this is the configuration of the file system backup, at the moment <dump> is not relevant, so its value is 0. <pass> allows 3 values — 0, 1, 2. <pass> defines the check of the file system where: 0 — do not check, 1 — check in the first phase (usually used for the root partition), 2 — check in another phase. But the main thing is <options> and its value. The value - defaults, although rw (read-write) and ro (read-only) parameters are also available for the user to use the container without restrictions.

In addition, it is necessary to select additional systemd parameters: x-systemd.automount — automatic file system mounting at the first access to it; x-systemd.nofail — allows the system to continue booting even if the file system cannot be mounted. Also, user needs to make changes to /etc/cryptsetup. Example of a rule in crypttab:

```
dm_crypt-0 UUID=xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx none luks
```

Also, user can add additional scripts in /etc/crypttab. These can be both self-written solutions and ready-made solutions of the /lib/cryptsetup/ scripts/passdev.

IX. DISCUSSION

We explored the topic of file system encryption for single board computers and demonstrated the possibility of solving the problem of easy access to the file system of single board computers. The demonstrated method used a utility for full encryption – LUKS, so the choice of this software was justified. We have analyzed the capabilities and available hashing functions of passwords and encryption algorithms and identified the advantages of a solution using the AES encryption algorithm. We also proposed a scenario using a USB key, which was a more convenient way to access an encrypted container and had its advantages over a password.

The problems of passwords and brute force attacks were outlined, which would allow the user to understand the need to follow the recommendations for their use and generation. Based on the research process, it should be emphasized that the result of encryption and the operation of individual system modules depended not only on the user configuration of the system, but also on the compiled system kernel and/or the limitations of the hardware on which encryption was performed.

The proposed option for solving the problem satisfied the modern conditions of information protection on single board computers, but the user must remember the development of technologies and Moore's law, that is, the passwords and encryption methods used at the moment may not be relevant after some time, because every year, the power of computer technology grows, and if at the moment the time to sort a 12-character password in compliance with the general recommendations on the sequence of characters is counted as tens of years, then after some time, it will take several minutes to select such a password.

X. CONCLUSION

The prototype of file system encryption on single board computers that meet modern security conditions using LUKS, with advantages presented among competitors. The validity of passwords against brute-force attacks was tested, allowing the user to understand the password requirements more accurately. The system's flexibility for creating and using a USB key was demonstrated in different scenarios. The paper also describes the corner cases encountered while reviewing the proposed solution. The result is a methodology that allows you to create and work with a crypto container to protect sensitive information.

References

[1] LUKS On-Disk Format Specification. Available at: https://gitlab.com/cryptsetup/cryptsetup/-/wikis/LUKS-standard/on-disk-format.pdf (accessed on 14 October 2024).

[2] Linux Unified Key Setup (LUKS) Disk Encryption Available at: https://wiki.archlinux.org/title/Dm-crypt/Encrypting_an_entire_system (accessed on 14 October 2024).

[3] Optimized Implementation of Argon2 Utilizing the Graphics Processing Unit. 10.48550/arXiv.1602.03097. Iuorio, Andrea & Visconti, Andrea. (2019). Understanding Optimizations and Measuring Performances of PBKDF2. 10.3390/app13169295

[4] Lee, Sung-Won & Sim, Kwee-Bo. (2021). Design and Hardware Implementation of a Simplified DAG-Based Blockchain and New AES-CBC Algorithm for IoT Security. Electronics. 10. 1127. 10.3390/electronics10091127.

[5] Cano Quiveu, German & Ruiz-de-clavijo-Vazquez, Paulino & Bellido, Manuel & Juan-Chico, Jorge & Viejo Cortes, Julian & Guerrero, David & Ostúa, Enrique. (2021). Embedded LUKS (E-LUKS): A Hardware Solution to IoT Security. Electronics. 10. 3036. 10.3390/electronics10233036.

[6] Performance Analysis of Encryption Algorithms in Named Pipe Communication for Linux Systems. 10.55549/epstem.1518789

[7] Saleh, Shaimaa & Al-Awamry, Amr & Taha, Ahmed. (2024). Tailoring AES for resource-constrained IoT devices. Indonesian Journal of Electrical Engineering and Computer Science. 36. 290. 10.11591/ijeecs.v36.i1.pp290-301.

[8] Parallel Implementations of ARX-Based Block Ciphers on Graphic Processing Units. 10.3390/math8111894

[9] Banakh, Roman. (2018). AUTHENTICATION METHOD WPA/WPA2 KEY PARAMETERS' DEFINITION FOR IEEE 802.11 BASED HONEYPOT. Radio Electronics, Computer Science, Control. 110-118. 10.15588/1607-3274-2018-1-13.

[10] Raspberry PI Official Site. Available at: https://www.raspberrypi.com/ (accessed on 14 October 2024).

[11] Byun, Hyeonsu & Kim, Jueun & Jeong, Yunseok & Seok, Byoungjin & Gong, Seonghyeon & Lee, Changhoon. (2024). A Security Analysis of Cryptocurrency Wallets against Password Brute-Force Attacks. Electronics. 13. 2433. 10.3390/electronics13132433.

[12] Proxy Nova - Password Brute Force Calculator. Available at: https://www.proxynova.com/tools/brute-force-calculator/ (accessed on 14 October 2024).

Bohdan Onishchenko was born in Chernihiv, Ukraine, in 2004, is a current fourth-year student at the Department of Information Technology Security at Lviv Polytechnic National University, anticipating to receive his bachelor's degree in 2024. His research and coursework focus on exploring the intersections of IoT and cloud infrastructures, emphasizing how embedded technologies can be fortified against cybersecurity threats.

Roman Banakh, the author, was born in Lviv, Ukraine, in 1992. He received his Ph.D. in cybersecurity at Lviv Polytechnic National University in 2024. He has been working in the cybersecurity scientific field since 2012 and has published more than 30 articles. His research interests include cloud computing, informational and communication systems on high load, security of IoT, and computer system architecture.

Abdallah Ibrahim earned a B.Sc. in Computer Engineering from Suez Canal University in 2010, followed by two M.Sc. degrees in Computer Sciences—one in 2012 from Suez Canal University and another in 2015 from the University of Luxembourg. In 2019 he completed a Smart ICT diploma in collaboration with ILNAS, Dell EMC, and the University of Luxembourg. His research interests include Cloud and Edge Computing, QoS, Industry 4.0, smart mobility, optimization, and machine learning.