

Using Hash Functions to Protect Critical Messages from Changes in Risky Computing Systems

Andrzej Smolarz¹, Nadiia Pasieka², Andrzej Kotyra¹, Vasyl Sheketa³, Mykola Pasieka³, Svitlana Chupakhina², Petro Krul² and Mykola Suprun⁴

¹ Lublin University of Technology, Nadbystrzycka 38D, Lublin, 20 – 618, Poland

² Vasyl Stefanyk Precarpathian National University, Ivano-Frankivsk, 76000, Ukraine

³ National Tech. University of Oil & Gas, Ivano-Frankivsk, 76068, Ukraine

⁴ National Pedagogical Dragomanov University, Kyiv, 01601, Ukraine

Abstract

Nowadays, information security has become an integral part of risky digital operations in cyberspace. To solve such problems associated with risky operations, the use of hash functions occupies one of the key places among information security technologies in modern cryptography. After all, these crypto-resistant functions process critical data of limited length while creating a small, fixed-size digital code called a hash or digest. The classic use of risk-dependent crypto functions is to check the integrity of the processed data, as well as authentication schemes for user messages. However, as the volume of risk-dependent processed critical data grows, the criterion of hash function throughput comes to the fore. Thus, SHA (Secure Hash Algorithm 3) is the latest member of the Secure Hash Algorithm family of standards, released by NIST on August 5, 2015. Although SHA-3 belongs to the same series of standards, it is internally different from the MD5-like structure of SHA-1 and SHA-2. The Secure Hash Algorithm is considered one of the first generally accepted hashing standards and, according to the currently published literature, is considered the fastest implementation of the SHA-512 (SHA-3 variant) cryptographic algorithm, providing acceptable bandwidth for protecting risk-dependent critical data.

Keywords

Hash functions, security risk, cryptography, critical data, secure hash algorithm.

1. Introduction

The use of cryptographic hash functions to protect risk dependent critical data, are related to the usual hash functions, which have additional properties, at the same time usual hash functions do not necessarily have them. These functions convert the final input of an information data stream to protect risk-dependent critical information into a small output of a fixed size, sometimes called

CITRisk'2021: 2nd International Workshop on Computational & Information Technologies for Risk-Informed Systems, September 16–17, 2021, Kherson, Ukraine

EMAIL: a.smolarz@pollub.pl (A.Smolarz); pasyekanm@gmail.com (N.Pasieka); a.kotyra@pollub.pl (A.Kotyra); vasylsheketa@gmail.com (V.Sheketa); pms.mykola@gmail.com (M.Pasieka); cvitlana2706@gmail.com (S.Chupakhina); krulp59@gmail.com (P.Krul); suprun62@ukr.net (M.Suprun)

ORCID: 0000-0002-6473-9627 (A.Smolarz); 000-0002-4824-2370 (N.Pasieka); 0000-0002-9442-6090 (A.Kotyra); 0000-0002-1318-4895 (V.Sheketa); 0000-0002-3058-6650 (M.Pasieka); 0000-0003-1274-0826 (S.Chupakhina); 0000-0002-5274-3215 (P.Krul); 0000-0002-4198-9527 (M.Suprun)



© 2021 Copyright for this paper by its authors.

Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

a summation. A good hash function has foreground image resistance, second foreground image resistance, and strong collision resistance. Resistance to the forward image means that it is computationally impossible to find any input that yields a digest equal to the predefined output. Second-image robustness means that it should be computationally infeasible to find any second input that has the same output as any given input. Anti-collision means that it is computationally infeasible to find two different input hashes with the same output.

Hash functions can be divided into two categories: keyed and unkeyed. Hash functions with a key are used in message authentication schemes. Such functions accept not only the hashed data, but also a second parameter, called a key. The hashing algorithm uses the actual data and the provided key to create the hash.

A function without a key is usually used to check the integrity of the data. The raw data is passed through the function and a summary of the results is stored. Later, a copy of the data is hashed and the two hashes are compared. If they are the same, the original data and the copy are the same for all intents and purposes. However, if they are different, the integrity of the copy cannot be trusted because the data could have been altered due to transmission errors or, even worse, by an external source for malicious purposes. A more detailed description and analysis of the hash function can be found in [1].

Thus hash functions require more and more computational resources. And because of the iterative nature of such functions, implementations of these algorithms cannot use computational techniques such as parallel processing to significantly improve performance. General-purpose processors are poorly suited to efficiently perform the basic operations required for a given hashing algorithm. Hence, hardware implementations of such algorithms are being developed to improve performance.

As new hash functions emerge, there must also be a way to compare them for authenticity. In such comparisons it is crucial to distinguish between software and hardware implementations of hash functions. Software implementations running on general-purpose processors are used for many authentication protocols as well as for independent computing applications. Hardware implementations created using dedicated circuits are typically used for devices that are impractical or impossible to implement with software, such as embedded systems or high-speed network devices. Runtime and memory usage are two critical metrics to consider when comparing software implementations of hash functions. Thus, the lower the runtime and memory usage, the more advantageous the hash function is. In hardware, runtime and total area are two important metrics to consider. For this type of implementation, runtime can usually be reduced by increasing the total area. This area can be calculated based on the physical size occupied by the final implementation and the number of logic gates used to build it. A hardware implementation can be used to achieve faster execution times than software can provide. Hardware devices such as FPGAs can provide the highly optimized operations needed for hashing algorithms. Therefore, such implementations are usually orders of magnitude better than software implementations.

2. Model analysis of cryptographic algorithms for hash functions compression of critical messages

There is no paper discussing the hardware implementation of the Phash algorithm, but many papers discussing the hardware implementation of SHA-512 have been published and Whirlpool algorithm. The proposed cryptographic models are generally used for the process of obtaining the maximum throughput of critical message compression by hashing functions. These implementations are usually aimed at high throughput or efficient resource utilization [1, 3, 4, 7].

Often, it is impossible to know a priori which design choice for a given component will be the best choice to achieve a particular design goal. Whenever possible, several descriptions and implementations of specific components are provided. After implementing and executing an algorithm using different components, it is possible to comment on the quality of a given implementation, as it is related to achieving high throughput or low total area.

This publication contains information about currently available and existing hardware implementations of SHA-512. Including descriptions and analysis of each implementation [2, 5, 6]. These publications describe the message expansion and compression steps in some detail. Most of the implementations reviewed acknowledge the existence of a key fill phase, but do not include details. Similarly, the hash update phase is partially mentioned, but its implementation is not always correctly described. One of the reasons, in our opinion, for omitting the details of the fill phase and the hash update phase is the hypothesis that the practical implementation of these phases will not have a significant negative impact on the overall performance characteristic of the hash function algorithm.

Simple Hash Function Implementation

Some of the first released hardware implementations of SHA-512 are described in scientific publications [8, 9, 11, 13, 17]. In [19], the implementation of $SHR_{(x)}$ and $ROTR_{(x)}$ operations, used in the extension and compression phase of the algorithm message processing, is discussed in detail. Thus, a block of ROM (Read Only Memory) is used to store the constant variable K_t required for the message compression phase. Several 64-bit adders and simple combinatorial circuits are used to calculate these values, which are necessary for the algorithm. Using the 64-bit adder with five inputs which is involved to calculate the time value T_1 , and the 64-bit adder with two inputs which is involved to calculate the total time value T_2 . On this basis the functions $Ch_{(x, y, z)}$ and $Maj_{(x, y, z)}$ are realized by simple combinatorial circuits.

A more detailed description of the SHA-512 implementation is given in [10, 14]. The construction of the disclosure phase of the message hash function is shown in Figure 1.

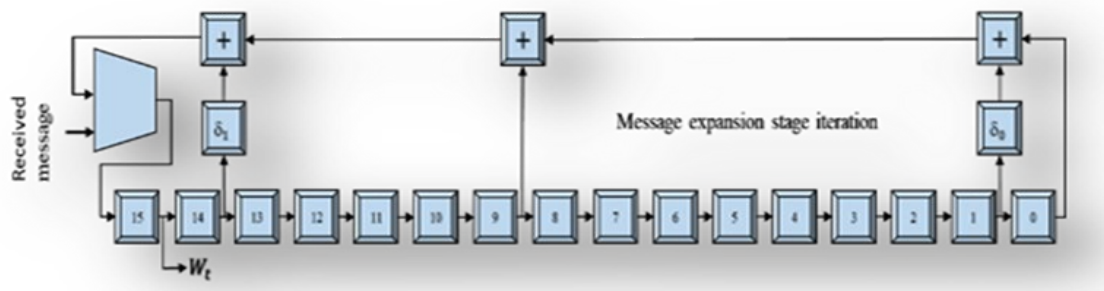
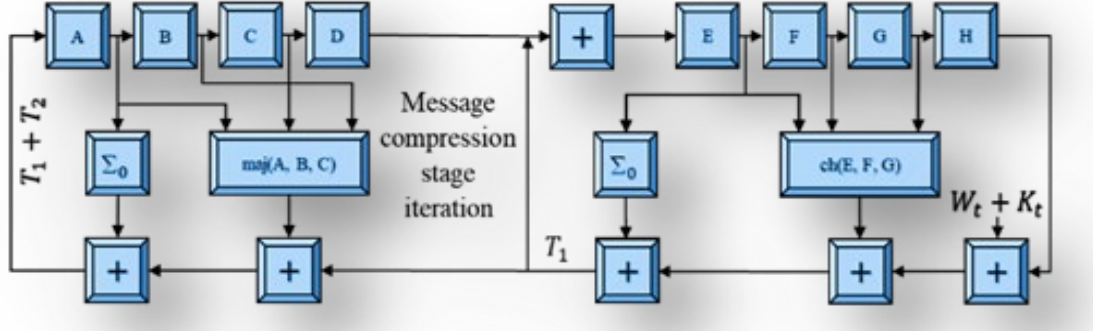


Figure 1: Model design of message expansion stage iteration

So the advantage of the considered implementation is the reuse of memory registers. A simple implementation requires would require eighty 64-bit registers to store all the necessary critical data generated during the message expansion phase, but because of resource sharing only 16 such registers and a multiplexer are needed. From the considered scheme we see that each of the sixteen registers has a width of 64 bits, and the multiplexer is used to determine the time of loading the first block of critical data in the register of the computing system. Thus the data output W_i from the leftmost register of the circuit algorithm is used as input to the compression stage of the generated message. As soon as the first block of critical data is loaded into all registers, the multiplexer immediately returns the result of the specified calculation of the hash function in the leftmost register. The considered design of the hash function algorithm at the message compression stage in [1] is shown in Figure 2. Based on this scheme, the eight 64-bit registers represent the intermediate values of the message hash, which will be used in the update phase after

the 80 iterative cycle of the compression phase [12, 15, 18]. For a more simplified understanding of the proposed algorithm, the details of the implementation of the computational operations $SHR_{(x)}$ and $ROTR_{(x)}$ are omitted. On this basis, we used two two-port Block RAMs to store the K_t constants used in this iterative stage. Block RAMs are on-chip memory components which can be



instantiated and addressed like regular memory modules. The details of the adders used in this implementation are also omitted in this publication.

Figure 2: Model design of message compression stage iteration

A more detailed implementation of the message expansion step diagram is shown in Figure 3. To reduce the total number of additions to be performed on the critical path, it is crucial to use the algorithm to compute the total $Kt + Wt$ in the previous loop [16, 17, 21]. The Rkw register ensures that the quantity is available during the correct iteration. The output of this register is used in the intermediate step of message compression. The combination of CSA and CPA is used to minimize the mathematical calculations and thus the delay that occurs when adding multiple 64-bit numbers to the processing. If you only use cascade CPA , in this case the delay will be even longer. Since the LUT Shift A and LUT Shift B blocks in this algorithm scheme are shift registers. They are designed using the "shift register mode" CLB slices available in the Virtex $FPGA$. Thus they are designed to significantly reduce the overall footprint while not affecting performance.

The design model of the message compression stage structure is shown in Figure 3. In this case, the multiplexer automatically determines whether the current $K_{t+0} - K_{t+7}$ values should be used additionally. For these calculations, the multiplexer selects a value of zero for addition at all stages, except for the last stage of compression of the desired message [20, 22, 24, 26]. The last iteration of the model will additionally use the calculated current $K_{t+0} - K_{t+7}$ value. Thus, the basic operation of these multiplexers serves as a key step to update the hash of the message.

The design model of the necessary message compression process in two iterations is presented in figure 4, and the thickened line shows the critical path of mathematical calculations, which create delays in message compression [23]. On this basis we can notice that when message compression step is extended then working frequency of the algorithm is reduced approximately by half, thus negating the purpose of expansion. On this basis, it should be noted that in order to significantly increase the operating frequency of compression of the necessary message, it is necessary to perform part of the operations in parallel. The study proposes a quasi-conveyor implementation of the SHA-512 mathematical encryption algorithm. For this purpose, a pipeline approach is used in the phase of compression of the necessary message, and since it forms the key delay on the execution of the algorithm [25]. Then the disclosure phase of the main necessary message has been optimized, so as to significantly reduce the delay of a single mathematical calculation step. Thus, in order to significantly reduce the delay in the pre-math phase, it is necessary to use delay balancing technology [27]. Based on this, CSA and CLA must be used to make the necessary critical additions [28]. Analyzing the presented model design scheme figure 3 we can notice that the section τ is influenced by variables Kt and Wt the section τ_{t-1} is indirectly influenced by variables $Kt-1$ and $Wt-1$ through τ .

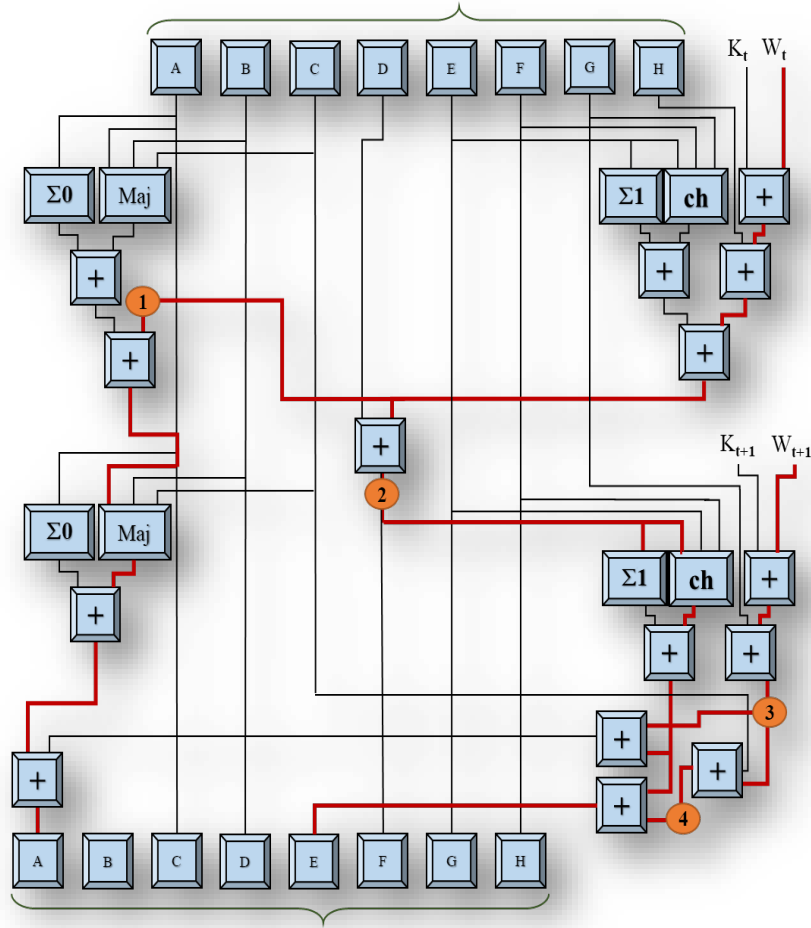


Figure 4: Design two iterations of message compression stage iteration

Based on the operation of the cryptographic algorithm at the stage of reading the last pair of values Kt and Wt it is necessary to carry out three consecutive clock cycles for the final completion of the critical message compression stage. Thus, at the first stage of the cryptographic algorithm for critical message compression registers are used to transmit new values. Which is used in the second step of generating the final values from E to H . And the third step of the execution of the cryptographic algorithm performs the generation of values from A to D . Also note that in the third step of the algorithm the variables from E to H must not be synchronized in any way.

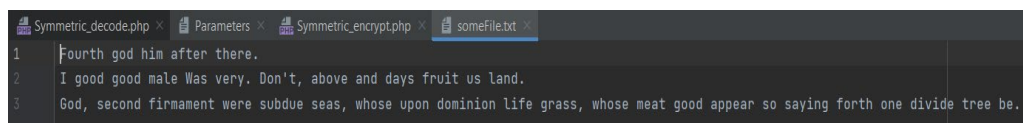
Thus, for effective delay balancing is used to significantly reduce the critical path of mathematical calculations still in the critical message expansion phase, but this method is not quite suitable for its use in combination with the unpacking process, so a modified algorithm was developed to implement the message expansion phase. Based on this effective deployment of this cryptographic model architecture, the values of the variables Wt and $Wt+1$ are required to execute the algorithm in one clock cycle. This usually uses sixteen sixty-four bit registers, and in a modifying algorithm, up to eight registers, but this doubles the width to one hundred and twenty-eight bits. And the first sixty-four-bit word of the new register is used to calculate the variable value Wt , and the second sixty-four-bit word is used to calculate the value $Wt+1$. The combinatorial logic of the message expander was copied to calculate the variable values Wt and $Wt+1$ in one loop of the algorithm. To do this, the multiplexer selection of one hundred and twenty-eight bits instead of sixty-four bits was changed. In this case the outputs of the variable values Wt and $Wt+1$ are fixed, to ensure that they do not affect the critical delay path of the mathematical calculations.

Thus, after the completion of the cryptographic algorithm at the 80th iteration of the critical message compression stage, the created hash code must be updated. In the classical version, this requires eight 64-bit adders, but we consider a hash modification scheme which uses only two adders. Assuming that, most hash variables are only moved to create a new output sequence, and hence, there was definitely an interdependency between the hash variables. Thus, using this interdependence, only two adders are needed to complete the critical message compression cryptographic algorithm to generate six partial results of hash modification iterations, and the other two value variables are calculated differently.

Using these interdependencies, the partial results of the hash modification iterations can also be determined. Based on the above, only two adders can also be used to perform the necessary number of calculations.

3. Practical implementation of the proposed hash function compression algorithm for critical message compression in risk dependent computer systems

In order to analyze how a cryptographic algorithm of hash function compression of critical messages in risk dependent computer systems will behave, a program model was developed, on which the testing of Figure 5-8 was carried out.



```

Symmetric_decode.php x Parameters x Symmetric_encrypt.php x someFile.txt x
1 Fourth god him after there.
2 I good good male Was very. Don't, above and days fruit us land.
3 God, second firmament were subdue seas, whose upon dominion life grass, whose meat good appear so saying forth one divide tree be.

```

Figure 5: File with initial data


```

Fourth god him after there.

I good good male Was very. Don't, above and days fruit us land.

God, second firmament were subdue seas, whose upon dominion life grass, whose meat good appear so saying forth one divide tree be.

```

Figure 6: Data file after SHA-512 hash function

```

Symmetric_decode.php x Parameters x Symmetric_encrypt.php x someFile.txt x Decode_file_symmetric.txt x Encrypt_file_symmetric x
1 vXfgKhqpcIEVMFD2F4/w59ushLZxofLrZ4V4Hkc=
2 sjjyNwGlcIEYDxs+E408452PgLdxo//8bN52Vyk9QM6FQYU8SX3vidxwwvg4AUqcAQxGtoB43B/Bn3D9JART8=
3 vHfxdE6yNYUYOhS+GIui69y1hKoL9e3rZ4V2YDittgJzFRQp/yC+vjF3yFjroQkpPwx41W1xTpbbsxu2Dd8ILvUIJXN2QB1LVn4LJkP3a+cmJNRVQVLgW/6muH3rdc7Y7I3d2oNwT7XCaQfz9bDk

```

Figure 7: File with decoded data

```

<?php
// Створення ключі
$config = array(
    "digest_alg" => "sha512",
    "private_key_bits" => 4096,
    "private_key_type" => OPENSSL_KEYTYPE_RSA,
);
$resource = openssl_pkey_new($config);
openssl_pkey_export($resource, $private_key);
$key_details = openssl_pkey_get_details($resource);
$public_key = $key_details["key"];

$keys = array('private' => $private_key, 'public' => $public_key);
echo $keys['public'] . '<br><br>';
echo $keys['private'] . '<br><br>';

echo '<div style="overflow: scroll;
    overflow-y: hidden">';
$data_file = fopen('someFile.txt', 'r');
$encrypt_file = fopen('PublicKey\Encrypt_file_public.txt', 'w');
while (!feof($data_file)){
    $plaintext = fgets($data_file);
    openssl_public_encrypt($plaintext, $encrypted, $keys['public']);
    $message = base64_encode($encrypted);
    fputs($encrypt_file, $message . "\r\n");
    echo $message . '<br>';
}
echo '</div>';
fclose($data_file);
fclose($encrypt_file);
echo '<br><br>';

$Encrypt_file = fopen('PublicKey\Encrypt_file_public.txt', 'r');
$Deciphered_file = fopen('PublicKey\Deciphered_file_public.txt', 'w');
while (!feof($Encrypt_file)){
    $ciphertext = fgets($Encrypt_file);
    if(!$ciphertext)break;
    $ciphertext = base64_decode($ciphertext);
    openssl_private_decrypt($ciphertext, $decrypted, $keys['private']);
    fputs($Deciphered_file, $decrypted);
    echo $decrypted . '<br>';
}
fclose($Encrypt_file);
fclose($Deciphered_file);

```

Figure 8: Model program hash function compression of critical messages

In this paper, the informational and technological impact of risk protection requirements in relation to critical data for regulation on the software development process has been evaluated. In doing so, the main changes in the hashing of information flows to which this process is defined and how to be prepared for them are described. It is prepared for the division of the critical risk-dependent personal data of the users into ordinary and special categories.

The way and the reasons for their collection, processing and disclosure are explained. In addition, it is necessary to ensure the security of these data, to prevent their leakage outside the organization where they are stored. This can cause damage to the data subject and lead to fines of considerable magnitude.

4. Conclusion

Conducting a critical analysis of the use of cryptographic hash functions to protect risk-dependent critical data, as well as to quickly compress critical messages in risk-dependent computer systems, we can argue that the use of mathematical algorithms has significant promise for information security technologies in modern cryptography. After all, these crypto-resilient functions for protecting risk-dependent data process critical information of limited length while creating a small digital code of fixed size. Along with classic applications of these crypto-resilient hash functions, such as verifying the integrity of the data being processed, and authentication schemes for critical user messages. Research has shown that with a significant increase in the amount of critical information being processed to protect risk-dependent critical data, the hash function's throughput criterion comes to the fore. Therefore, information security research today to protect risk dependent critical data has become an integral part of risky digital operations in cyberspace, and solving such problems with hash functions occupies one of the key places in modern cryptography. Thus, the modification of hash algorithms for protection of risk-dependent critical data, in order to increase the throughput is extremely relevant.

References

- [1] M.McLoone, J.V McCanny, Efficient single-chip implementation of SHA-384 and SHA-512, 2002 IEEE International Conference on Field-Programmable Technology (FPT). Proceedings (Cat. No.02EX603), 2002, pp. 311 – 314
- [2] A.Khas, I.Cicek, SHA-512 based Wireless Authentication Scheme for Smart Battery Management Systems, 2019 8th International Conference on Renewable Energy Research and Applications (ICRERA), Brasov, Romania, 2019, pp. 968-972. doi: 10.1109/ICRERA47325.2019.8996531
- [3] Hłobaž, Statistical Analysis of Enhanced SDEx Encryption Method Based on SHA-512 Hash Function, 2020 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 2020, pp. 1-6. doi: 10.1109/ICCCN49398.2020.9209663
- [4] S.Rathor, M.Rathor, Security of Functionally Obfuscated DSP Core Against Removal Attack Using SHA-512 Based Key Encryption Hardware, in IEEE Access, vol. 7, 2019, pp. 4598-4610. doi: 10.1109/ACCESS.2018.2889224
- [5] U.Rehman, H.Wang, M.M.Ali Shahid, S.Iqbal, Z.Abbas, A.Firdous, A selective cross-substitution technique for encrypting color images using chaos, DNA Rules and SHA-512, in IEEE Access, vol. 7, 2019, pp. 162786-162802. doi: 10.1109/ACCESS.2019.2951749

- [6] Ge et al., Optimized Password Recovery for SHA-512 on GPUs, 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, 2017, pp. 226-229. doi: 10.1109/CSE-EUC.2017.226
- [7] F.Kahri, H.Mestiri, B.Bouallegue, M.Machhout, An efficient fault detection scheme for the secure hash algorithm SHA-512, 2017 International Conference on Green Energy Conversion Systems (GECS), Hammamet, 2017, pp. 1-5. doi: 10.1109/GECS.2017.8066141
- [8] H.N.Bhonge, M.K.Ambat, B.R.Chandavarkar, An Experimental Evaluation of SHA-512 for Different Modes of Operation, 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-6. doi: 10.1109/ICCCNT49239.2020.9225559
- [9] H.Setiawan, K.Rey Citra, Design of Secure Electronic Disposition Applications by Applying Blowfish, SHA-512, and RSA Digital Signature Algorithms to Government Institution, 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia, 2018, pp. 168-173. doi: 10.1109/ISRITI.2018.8864280
- [10] I.Dronyuk, O.Fedevych, N.Kryvinska, High Quality Video Traffic Ateb-Forecasting and Fuzzy Logic Management, 2019 7th International Conference on Future Internet of Things and Cloud (FiCloud), Istanbul, Turkey, 2019, pp. 308-311. doi: 10.1109/FiCloud.2019.00051
- [11] I.Dronyuk, Y.Klishch, S.Chupakhina, Developing Fuzzy Traffic Management for Telecommunication Network Services, 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Polyana, Ukraine, 2019, pp. 1-4. doi: 10.1109/CADSM.2019.8779323
- [12] A.Abuhasel, M.A.Khan, A Secure Industrial Internet of Things (IIoT) Framework for Resource Management in Smart Manufacturing, in IEEE Access, vol. 8, 2020, pp. 117354-117364. doi: 10.1109/ACCESS.2020.3004711
- [13] A.Nugroho, A.Hangga, I.M.Sudana, SHA-2 and SHA-3 based sequence randomization algorithm, 2016 2nd International Conference on Science and Technology-Computer (ICST), Yogyakarta, 2016, pp. 150-154. doi: 10.1109/ICSTC.2016.7877365
- [14] A.Kalyankar, C.Kumar, Aadhaar Enabled Secure Private Cloud with Digital Signature as a Service, 2018 Second International Conference on Electronics, Communication and Aerospace Technology, Coimbatore, 2018, pp. 533-538. doi: 10.1109/ICECA.2018.8474603
- [15] M.Pasyeka, T.Sviridova, I.R Kozak, Mathematical model of adaptive knowledge testing, 2009 5th International Conference on Perspective Technologies and Methods in MEMS Design, Zakarpattya, 2009, pp. 96-97
- [16] M.Medykovskyy, M.Pasyeka, N.Pasyeka, O. Turchyn, Scientific research of life cycle performance of information technology, Paper presented at the Proceedings of the 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT 2017, 1, 2017, pp. 425-428
- [17] M.Nazarkevych, A.Marchuk, L.Vysochan, Ya.Voznyi, H.Nazarkevych, A.Kuza, Ateb-Gabor Filtering Simulation for Biometric Protection Systems, CPITS 2020, 2020, pp. 14-22. doi:10.1109/STC-CSIT.2017.809882
- [18] H.Mykhailyshyn, N.Pasyeka, V.Sheketa, M.Pasyeka, O.Kondur, M.Varvaruk, Designing network computing systems for intensive processing of information flows of data, Lecture Notes on Data Engineering and Communications Technologies, vol.48, 2021, pp. 391- 422. doi:10.1007/978-3-030-43070-2_18
- [19] M.Nazarkevych, N.Lotoshynska, V.Brytkovskyi, S.Dmytruk, V.Dordiak, I.Pikh, Biometric identification system with ateb-gabor filtering, Paper presented at the 2019 11th International

- Scientific and Practical Conference on Electronics and Information Technologies, ELIT 2019 - Proceedings, 2019, pp.15-18. doi:10.1109/ELIT.2019.8892282
- [20] P.Singh, S.K.Saroj, A Secure Data Dynamics and Public Auditing Scheme for Cloud Storage, 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 695-700. doi: 10.1109/ICACCS48705.2020.9074337.
- [21] M.Pasyeka, V.Sheketa, N.Pasieka, S.Chupakhina, I.Dronyuk, System analysis of caching requests on network computing nodes, Paper presented at the 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019 - Proceedings, 2019, pp. 216-222. doi:10.1109/AIACT.2019.8847909
- [22] R.K.Pandey, Y.Zhou, B. U.Kota, V.Govindaraju, Deep Secure Encoding for Face Template Protection, 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Las Vegas, NV, 2016, pp. 77-83. doi: 10.1109/CVPRW.2016.17
- [23] R.Tkachenko, I.Izonin, N.Kryvinska, I.Dronyuk, K.Zub, An approach towards increasing prediction accuracy for the recovery of missing iot data based on the grnn-sgtm ensemble, Sensors (Switzerland), 20(9), 2020. doi:10.3390/s20092625
- [24] R.Tkachenko, I.Izonin, P.Vitynskyi, N.Lotoshynska, O.Pavlyuk, Development of the non-iterative supervised learning predictor based on the ito decomposition and sgtm neural-like structure for managing medical insurance costs, Data, 3(4), 2018. doi:10.3390/data3040046
- [25] V.Sheketa, M.Pasyeka, N.Lysenko, O.Lysenko, N.Pasieka, Yu.Romanyshyn, Neural Networks in Intelligent Analysis Medical Data for Decision Support, IDDM 2020, 2020, pp. 252-264
- [26] M.Pasyeka, V.Sheketa, N.Pasieka, S.Chupakhina, I.Dronyuk, System analysis of caching requests on network computing nodes, 3rd International Conference on Advanced Information and Communications Technologies, AICT2019 - Proceedings, 2019, pp. 216-222. doi:10.1109/AIACT.2019.8847909
- [27] S.Lee, K.Shin, An efficient implementation of SHA processor including three hash algorithms (SHA-512, SHA-512/224, SHA-512/256), 2018 International Conference on Electronics, Information, and Communication (ICEIC), Honolulu, HI, 2018, pp. 1-4. doi: 10.23919/ELINFOCOM.2018.8330578
- [28] R.S.Romaniuk, A.Smolarz, W.Wójcik, Photonics applications and web engineering: WILGA 2021, Proc. SPIE 12040, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2021, 120400Y (29 November 2021). <https://doi.org/10.1117/12.2603845>