IJCCS (Indonesian Journal of Computing and Cybernetics Systems)
Vol.13, No.3, July 2019, pp. 273~282
ISSN (print): 1978-1520, ISSN (online): 2460-7258
DOI: https://doi.org/10.22146/ijccs.47267

Data Integrity and Security using Keccak and Digital Signature Algorithm (DSA)

Muhammad Asghar Nazal*[1], Reza Pulungan[2], Mardhani Riasetiawan[3]
[1]Master Program of Computer Science and Electronics, FMIPA UGM, Yogyakarta, Indonesia
[2,3]Department of Computer Science and Electronics, FMIPA UGM, Yogyakarta, Indonesia
e-mail: ***[1]m.asghar@mail.ugm.ac.id**, [2]mardhani@ugm.ac.id, [3]pulungan@ugm.ac.id

Abstrak

Data security menjadi sangat penting ketika menggunakan cloud computing, salah satu penelitian yang sedang berjalan dan menggunakan teknologi cloud sebagai sarana penyimpanan adalah G-Connect. Salah satu pengembangan yang dilakukan projek G-Connect adalah mengenai keamanan pada data, terutama dalam masalah verifikasi data yang dikirim. Pada penelitian sebelumnya, algoritma Keccak dan RSA diimplementasikan untuk kebutuhan verifikasi data. Namun setelah dilakukan studi literatur mengenai algoritma lainnya yang dapat membuat tanda tangan digital, ditemukan bahwa terdapat algoritma yang lebih cepat dari RSA yaitu, Digital Signature Algorithm (DSA).

DSA merupakan algoritma kunci yang digunakan untuk tanda tangan digital, namun karena DSA masih menggunakan Secure Hash Algorithm (SHA-1) sebagai algoritma untuk hash, maka DSA sudah jarang digunakan untuk keperluan keamanan data, sehingga dipilih dan digunakan algoritma Keccak sebagai pengganti algoritma hash pada DSA. Sekarang ini algoritma Keccak telah dijadikan standar untuk algoritma fungsi hash SHA-3 yang baru. Karena permasalahan di atas maka fokus penelitian ini adalah pada keamanan data cloud dengan permasalahan verifikasi data menggunakan algoritma Keccak dan DSA. Hasil dari penelitian ialah terbukti bahwa algoritma Keccak dapat berjalan pada sistem kerja DSA, serta diperoleh perbandingan waktu eksekusi proses signing dan verifying antara DSA dan RSA di mana keduanya menggunakan algoritma Keccak.

***Kata kunci**—Algoritma Keccak, algoritma RSA, DSS, DSA*

Abstract

Data security is a very important compilation using cloud computing; one of the research that is running and using cloud technology as a means of storage is G-Connect. One of the developments made by the G-Connect project is about data security; most of the problems verification of the data sent. In previous studies, Keccak and RSA algorithms have implemented for data verification needs. But after a literature study of other algorithms that can make digital signatures, we found what is meant by an algorithm that is better than RSA in rectangular speeds, namely Digital Signature Algorithm (DSA).

DSA is one of the key algorithms used for digital signatures, but because DSA still uses Secure Hash Algorithm (SHA-1) as an algorithm for hashes, DSA rarely used for data security purposes, so Keccak is used instead of the hash algorithm on DSA. Now, Keccak become the standard for the new SHA-3 hash function algorithm. Because of the above problems, the focus of this research is about data verification using Keccak and DSA. The results of the research are proven that Keccak can run on DSA work system, obtained a comparison of execution time process between DSA and RSA where both use Keccak.

***Keywords—**Keccak algorithm, RSA algorithms, DSS, DSA*

Received July 3rd,2019; Revised July 17th, 2019; Accepted July 23th, 2019

1. INTRODUCTION

Cloud computing is a technology that utilizes services using a central server that is provided by a provider and is virtual and can provide services to the use of software, data storage, networks, and data computing using. Therefore, data security is very important when using cloud computing at all levels: infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS), including data-in-transit, data-at-rest, data processing, data flow, and data origin [1].

One study that is running and using cloud technology as a means of storage is G-Connect. G-Connect is a research that focuses on developing applications in the field of Internet of Things (IoT) and cloud technology with the characteristics of research locations in 3T areas that are minimal with their internet network. Some developments are in the active device IoT model, data management on IoT devices, management of data transmission from IoT devices to the cloud, and security for data transfer and data storage on the cloud.

For development carried out on G-Connect using cloud technology as a means of storing data, several aspects need to be considered in its development, that is data security aspects. In the security aspect of data, one of the problems is about verifying the data sent, whether it is from the node (address) that is correct or not. One method that can be used to verify data is the digital signature method. Digital signatures are a method used to authenticate message content and provide the ability to verify the owner of the message and the time of signature. One algorithm that can be used for digital signature needs is the Digital Signature Algorithm (DSA) [2].

DSA designed by the National Institute of Standards Technology (NIST) and the National Security Agency (NSA) in the early 90s and then published in 1991. DSA is a public key technique, which is only a scheme of digital signatures with the results of signs his hand is 320 signature bits [3]. DSA uses Secure Hash Algorithm (SHA) as its hash algorithm. SHA is the most used hash function, made by NIST and published in 1993. Until now, there are four types of SHA, i.e. SHA (SHA-0), SHA-1, SHA-2, and SHA-3 [4]. Although SHA-1 cannot be solved yet, because of the structure and operation similar to MD-5 and SHA-0, it is considered unsafe. SHA-2 is safer than SHA-1, but because of the same mathematical structure and operation as SHA-1, it might be unsafe. Therefore NIST built a new standard and created a competition generation new hash function created by NIST, which is now called SHA-3 [5].

The Keccak algorithm is one of the hash function algorithms designed by Guido Bertoni, Joan Daemen, Michael Peeteres, and Gilles Van Assche. Keccak is Keccak, the winner of the SHA-3 Cryptographic Hash Algorithm Competition, organized by NIST and has become the standard for the new Secure Hash Algorithm (SHA-3) hash function algorithm. Keccak is different from other SHA-3 finalists in terms of using sponge construction. If other designs depending on the compression function, Keccak uses a non-compression function to absorb and then squeezing the digestion [6].

In its application, the Keccak algorithm can be combined with public-key techniques. In the previous study [7], which discussed the use of the Keccak algorithm on RSA for data verification needs. But after a literature study of the RSA and DSA algorithms in the use of digital signatures, it was found that DSA is a better algorithm than RSA in terms of speed.

Based on the previous description, the focus of this research is on how the application of Keccak algorithm in Digital Signature Algorithm (DSA) to verify data, and comparison of execution time between Keccak algorithm on DSA and Keccak algorithm in RSA using data sourced on IoT devices.

2. METHODS

2.1 System Analysis

This research is part of G-Connect, where the research is one of the research projects within the Department of Computer and Electrical Sciences that implement IoT devices and Cloud technology to help disaster-prone areas. The scope of the G-Connect Project is divided into seven parts including device communication between Arduino and Raspberry Pi, compression and transmission of data on the Raspberry Pi, operating system and scheduling on Raspberry Pi, cloud scheduling, the transmission of data extraction, correct and data validation in the cloud while the main focus of this research is about data validation (verification) in the cloud. For illustrations data validation of G-Connect shown in **Figure 1**.

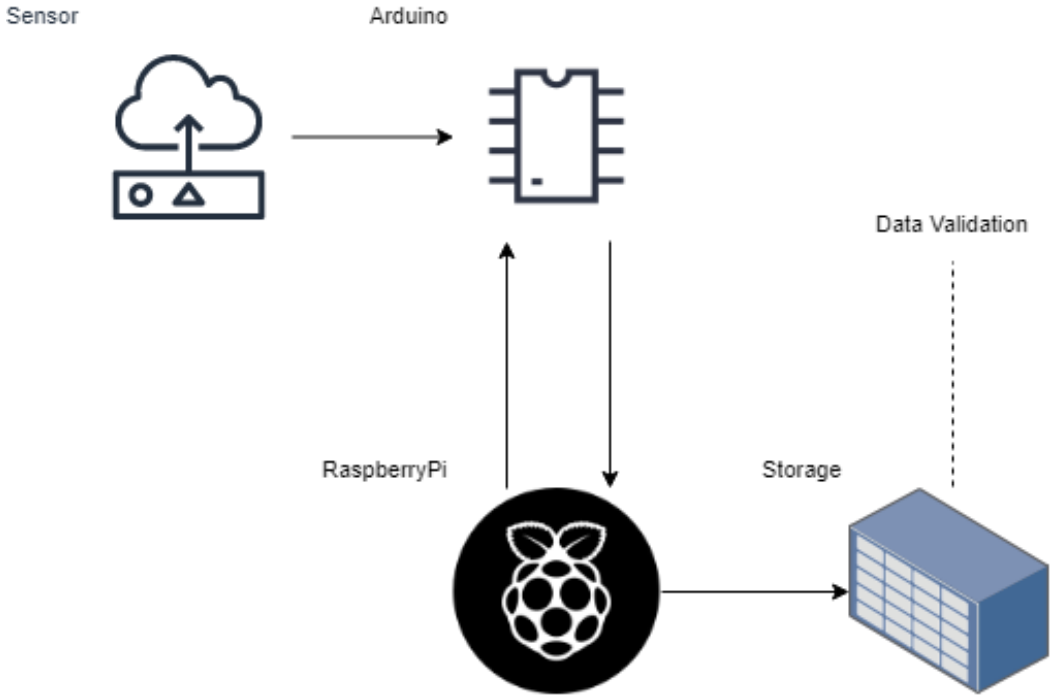

Figure 1 Illustrations Data Validation of G-Connect

Regarding data verification, this research used the Keccak algorithm on DSA to verify data received in the cloud. DSA is used to create digital signatures, while the Keccak algorithm is used to do the hashing process on the DSA. DSA creates 320-bit digital signatures and is an algorithm used for digital signature processes, where digital signatures are a method to authenticate message content and provide the ability to verify the owner of the message and the time the signature for the DSA summary is shown in **Table 1.**

Table 1 Digital Signature Algorithm

Public Key Global Values	
p	The prime numbers $2^{L-1} \le p \le 2^L$ for $512 \le L \le 1024$ and L are multiples of 64; that is bit lengths between 512 and 1024 bits with addition of 64 bits
q	The dividing prime number $(p-1)$, where $2^{159} < q < 2^{160}$ which the bit length is 160 bits
g	$g = h^{(p-1)/q} \bmod p$, where *h* any integer with $1 < h < (p-1)$ so that $h^{(p-1)/q} \bmod p > 1$
User Private Key	
x	Random or pseudorandom integer with $0 < x < q$
User Public Key	
y	$y = g^x \bmod p$
User Per-Message Secret Numbers	
k	Random or pseudorandom integer with $0 < k < q$
Signing	
r	$r = (g^k \bmod p) \bmod p$
s	$s = [k^{-1}(H(M) + xr)] \bmod q$
Signature = (r, s)	
Verifying	
w	$w = (s')^{-1} \bmod q$
u1	$u1 = [H(M')w] \bmod q$
u2	$u2 = (r')w \bmod q$
v	$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$
TEST: *v = r'*	
M	Message to sign
H(M)	*hash* fromM use SHA-1
M', r', s'	version received from *M, r, s*

For the Keccak algorithm, it is chosen to replace the hash function that exists on DSA, because the hash function on DSA still uses SHA-1, so Keccak is chosen to replace the hash that is on DSA, because Keccak is an SHA-3 hash function that has become the standard hash function new, proven security [8].

2. 2 Keccak Algorithm

Keccak is a one-way hash function algorithm based on sponge construction using the *f-keccak* permutation function with a permutation length range *b* size of each lane. The condition of *b* is indicated by **equation (1)** and **equation (2).**

$$b = 25 \times 2^l \quad (1)$$

where

$$0 \le l \le 6 \quad (2)$$

The Keccak algorithm has the same principle as the cipher block algorithm, where the process is carried out on blocks, each process result depends on the input and results of the previous process, and each process is imposed on the main function consisting of several round functions which are titrated several times. But there is a difference between the Keccak one-way hash algorithm, with the cipher block algorithm, as follows:

1. Keccak does not have a key schedule
2. Use round constants that are fixed rather than round keys.

Keccak uses the inner state during the hashing process. And the function of the sponge used consists of padding, absorbing, and squeezing. Each state has a length according to the length of the permutation i.e. *b*.

The Keccak algorithm accepts three input parameters, i.e. bitrate (*r*), capacity (*c*), and diversity (*d*). In general the process of this Keccak is:

1. Preparation of input messages (*P*), which is applying padding to the input message. The length of the message input padding result must be a multiple of *r*, with *r* = bitrate.
2. Enumeration of the input message becomes $P_0, P_1, P_2, \cdots, P_i$, where *i* = number of multiples of the length of the bitrate for the length of the input message.
3. *Absorbing* all fractions of the input message.
4. *Squeezing* a number of *j*, where *j* = multiple output lengths *r/w* to fill the desired output length, *r* = *bitrate* and *w* = lane length of state. Where:

$$w = 2^l \quad (3)$$

5. Output is a concomitation of squeezing output in a certain bitrate range.

State on Keccak is a series of bits that are seen as a three-dimensional array of these bits. Each axis in the array is represented by *x*, *y*, and *z*. *x x y* is the slice of the state, and z is the axis of the lane state. The process carried out on the Keccak state is based on each slice state. The number of bits for each slice in the state is fixed, i.e. 5 x 5 or 25-bit while the size of each lane for a state is 1, 2, 4, 8, 16, 32 or 64.

The Sponge function on Keccak is based on the sponge work scheme. The sponge work process scheme is a simple iterative process scheme for constructing a sponge function with variable length inputs and variable output lengths depending on the fixed length of transformation (or permutation) *f* operating in a fixed number *b* in bits [9].

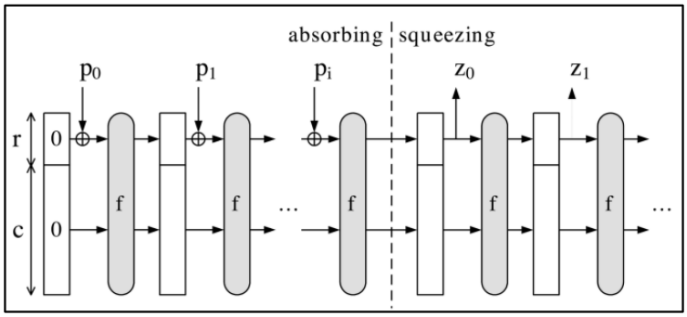

Figure 2 Keccak Sponge Scheme

The general equation for the *keccak-f* function is the *keccak-f* [b] or the *keccak-f* [r + c] where b corresponds to **equation (1**). Because the value of l has a range between 0 and 6, the possible values are 25. 50, 100, 200, 400, 800 and 1600. In Figure 1, there are two phases in the sponge construction, i.e. the absorbing phase and the squeezing phase.

1. *Absorbing phase*, is a phase where the process is carried out on all fractions of the input input $(P_0, P_1, P_2, \cdots, P_i)$ xor with the bitrate part of the state then passed into the *f Absorbing* function iteratively according to the number of fractions obtained..
2. Fase *squeezing*, is a phase to get the output. In this phase, several specific bits of *f* function is *confirmed* so that the number of concomitant bits is the same as the desired number of concomitant bits.

The function of the Keccak sponge is the application of sponge construction by first carrying out the initialization process. In general, the initialization process is divided into two stages, as follows:

1. Sets each bit in a state with zero for the initial state.
2. Applying the padding to the message so that the length of the input message is a multiple of the length of the initial bitrate specified. This process is done by adding 1 and several 0 as little as possible until the length of the message meets the multiples of the specified bitrate state length. **Equation (4)** and **equation (5)** are applied in this process.

$$P = pad(M,8) \,||\, enc(d,8) \,||\, enc(r/8,8) \quad (4)$$

$$P = pad(P,r) \quad (5)$$

$pad(M,n)$ function where message *M* plus 1 is then added 0 such that the number *M* is the smallest multiple of *n*.

$enc(x,n)$ function that produces a string with *n*-bit length taken from *Least Significant Bit* (LSB) kto *Most Significant Bit* (MSB), on *x*.

The *keccak-f* permutation function is the main function in Keccak. This function takes the state as input and performs several permutation operations consisting of five operating stages [6], i.e. diffusion (theta), inter-slice dispersion (rho), disturbing horizontal/vertical alignment (pi), non-linearity (chi) and break symmetry (iota).

1. Diffusion operation/ θ (theta) Diffusion operations are linear.
   This operation only checks 11 bits into one. Therefore, each bit affects the other eleven bits. In this process, 50 XOR and five rotations occur.
2. Inter-slice dispersion operation/ ρ (rho)
   Inter-slice dispersion operations consist of translation operations in the lane. Without this operation, the diffusion between slices will be very slow. This operation is also linear, with the inverse in the form of a reshuffle which is contrary to the previous shift.
3. Disturbing horizontal/vertical alignment operation/ π (pi)
   The disturbing operation horizontal/vertical alignment is a transposition operation against a lane that provides dispersion and aims to obtain long-term diffusion. The essence of this operation is to multiply each bit in slice with a matrix [[0,1], [2,3]].
4. Non-Linearity operation/ χ (chi)
   The non-linearity operation is the only non-linear mapping operation in the *keccak-f*. Without this operation, the round Keccak function will be linear. This operation can be seen as a 5w S-Box operation application for 5-bit lines. This operation itself is invertible; the inverse of χ itself is different.
5. Break symmetry operation/ ι (iota)
   The break symmetry operation consists of adding round constants which aim to disperse symmetry. The number of active bit positions in the round constant is $l + 1$. If l increases, the round constant will add more asymmetry.

This permutation operation in the *keccak-f* function is often also called the Round. At each *keccak-f* function, several rounds are carried out. The number of rounds recommended can be calculated using **equation (6)**, with l as in **equation (2)**. So for *keccak-f* [1600] the recommended number of rounds is 24.

$$N_r = 12 + 2l \quad (6)$$

2. 3 DSA System Architecture

DSA has a property in the form of several parameters as follows.

1. *p,* is a prime number with length *L* bits, where $512 \le L \le 1024$ and *L* must be multiples 64. Parameter *p* public and can be shared by people in the group.

2. *q*, is prime 160 bit, is a factor of $p-1$, so that $(p-1) \bmod q = 0$. Parameter *q* is a public key.
3. $g = h^{(p-1)/q} \bmod p$, where $h < p-1$ so that $h^{(p-1)/q} \bmod p > 1$. Parameter *g* is a public key.
4. *x*, is an integer and $x < q$. *x* is a private key.
5. *y*, $y = g^x \bmod p$, is a public key.
6. *m*, message will be signed.

DSA has three main processes; i .e. Key Pair Generation, Digital Signature Generation, and Digital Signature verification. The Key Pair Generation and Digital Signature Generation processes are shown in **Figure 3**, while the Digital Signature verification process is shown in **Figure 4.**

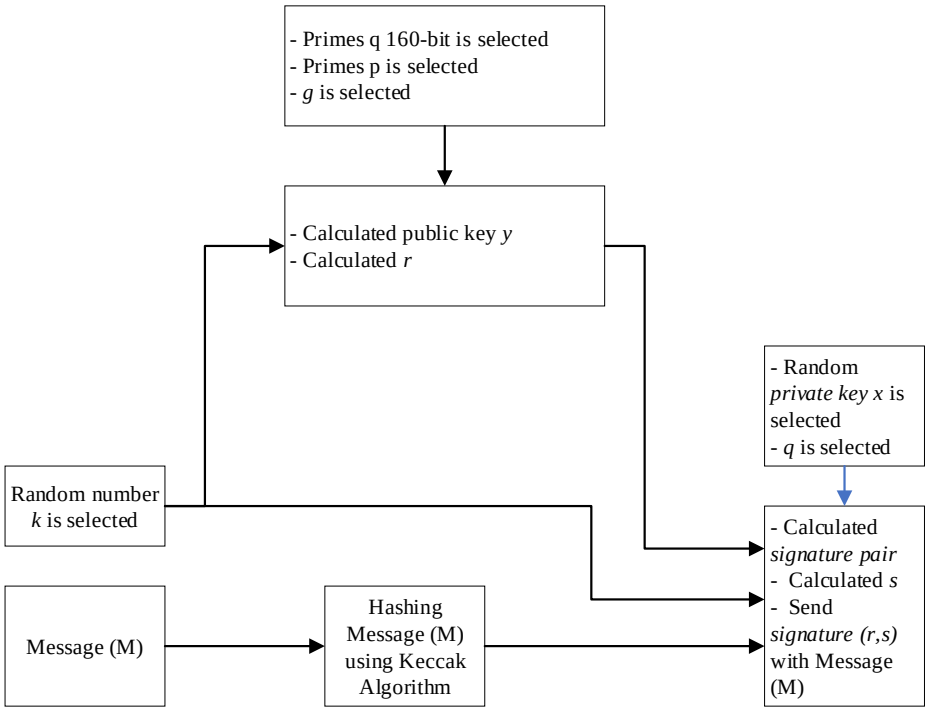

Figure 3 Illustration of the signing process

The procedure for generating a key pair is as follows.

1. Primes *p* and *q* are selected, where $(p-1) \bmod q = 0$
2. Calculated $g = h^{(p-1)/q} \bmod p$, where $h < p-1$ and $h^{(p-1)/q} \bmod p > 1$
3. Random private key *x* is selected, where $x < q$
4. Calculated public key $y = g^x \bmod p$

Next is a signature generation procedure (signing), as follows.

1. Message *m* is converted to message digest with Keccak Algorithm *H(m)*.
2. Random number *k* is selected, where $k < q$
3. The signature of message *m* is number *r* and *s*. *r* and *s* are calculated as follows.

$$r = (g^k \bmod p) \bmod q$$

$$s = [k^{-1}(H(m) + xr)] \bmod q$$

4. Message *m* sent with the signature *r* and *s*.

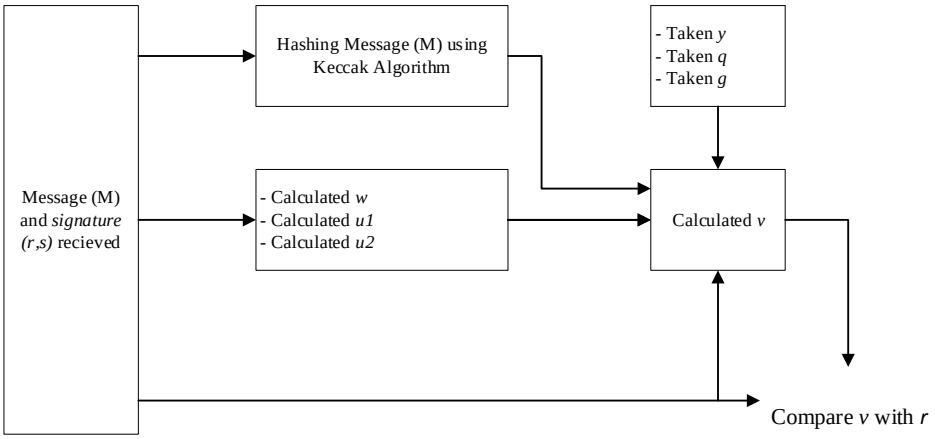

Figure 4 Illustration of the verifying process

The procedure for verifying the validity of the signature is as follows.

1. Calculated

$$w = s^{-1} \bmod q$$
$$u1 = (H(m)w) \bmod q$$
$$u2 = (rw) \bmod q$$
$$v = ((g^{u1} y^{u2}) \bmod q) \bmod q$$

2. If $v = r$, then the signature is valid; this means that the message is still original and sent by the correct sender.

3. RESULTS AND DISCUSSION

Testing is done using the black box testing method to ensure that the program runs as expected. Testing is done by calculating the execution time for the process of making a signature and the verification process. Testing is done by executing the program ten times for each algorithm and condition. The division of execution time is divided into two, i.e. the signing process and verifying time, with two conditions that valid data and invalid data (has been changed when the data was sent) The results of the execution of the signature processing time using the Keccak algorithm on DSA and the Keccak algorithm on RSA are shown in **Table 2** and **Table 3**.

Table 2 Keccak Process Execution Results on DSA and RSA for Valid Data

Experiment	**Time (second)**			
	Signature		**Verification**	
	DSA	**RSA**	**DSA**	**RSA**
1	0.599	5.820	0.078	0.030
2	0.710	5.519	0.080	0.017
3	0.199	5.746	0.078	0.013
4	0.151	6.052	0.094	0.014
5	0.198	5.726	0.079	0.013
6	0.406	5.569	0.078	0.015
7	0.348	5.367	0.091	0.016
8	0.321	5.501	0.092	0.010
9	0.419	5.436	0.078	0.015
10	0.278	5.337	0.081	0.014
Average	**0.363**	**5.607**	**0.083**	**0.016**

Table 2 is a table of results of the execution of the verification process on DSA and RSA, each of which uses the Keccak algorithm for its hashing function. From **Table 2** it can be found that the difference between the DSA and RSA for the signature processing time is 5.224 seconds faster when the DSA signature process, the difference in processing time is very far because the signature produced by DSA is 320 bits, while RSA produces a key 1024 —2048 bits [10]. For the verification process time, RSA is faster with a time difference of about 0.067 seconds, which is a difference that is not so far away.

Table 3 Keccak Process Execution Results on DSA and RSA for Non Valid Data

Experiment	Time (second)			
	Signature		Verification	
	DSA	RSA	DSA	RSA
1	0.262	5.437	0.093	0.010
2	0.167	7.071	0.076	0.014
3	0.561	5.539	0.079	0.011
4	0.242	5.935	0.077	0.011
5	0.326	5.677	0.076	0.012
6	0.317	5.569	0.080	0.015
7	0.421	5.366	0.079	0.014
8	0.493	5.388	0.084	0.016
9	0.450	5.450	0.099	0.011
10	0.221	5.578	0.087	0.012
Rata-rata	**0.346**	**5.701**	**0.083**	**0.013**

In **Table 3** the difference in the time of signature processing between DSA and RSA is 5.355 seconds superior to DSA, while for the verification process, the difference in time obtained is 0.070 seconds superior to RSA. From the results of **Table 2** and **Table 3**, it was found that the results of the time of the data verification process are valid and invalid data require execution times that are not much different.

For testing its integrity, the message *M* is modified to be another message (*M*') and then with a digital signature, the original message *M* is sent back to the recipient. If something like that happens, then with digital signatures, it can be known whether the message sent is still intact or has been modified. If the original message is modified even if only one character is then verified, then the verification results will show the message has been modified. This happens because the digital signature was invalid for the modified message. That way, the digital signature on the DSA can also check the integrity of the original message from the sender.

Furthermore, authentication testing is used to check the validity of the sender of the message. On the side of the recipient, signatures are verified to prove their authenticity using; first, the digital *signature* (*r, s*) is decrypted using the message sender's public key, generating the original message digest. Second, the receiver then converts the *M* message into a message digest using the Keccak algorithm, which is the same as the hash function used by the sender. Third, if $v = r$, means the signature is received authentic and comes from the correct sender. If the sender does not recognize that the sender sent the message, the recipient can prove it with the digital signature of the message.

4. CONCLUSIONS

The Keccak algorithm can be applied to the Digital Signature Algorithm (DSA) and runs according to the design of the system created. When compared with previous studies, where the Keccak algorithm is used in RSA digital signature work systems, the DSA signature work system that uses the Keccak algorithm is better in terms of the signing process execution time with a time difference of 5,224 seconds for valid data and 5,355 seconds for invalid data. But for verifying comparisons, the RSA execution process is faster than DSA with a time difference of 0.067 seconds for valid data and 0.070 seconds for invalid data. The Keccak algorithm on DSA can fulfill three information security services, i.e. the authenticity of data authentication, data integrity, and non-denial.

REFERENCES

[1] T. Mather, S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy*. United States of America: O'Reilly Media, 2009.

[2] W. Stallings, *Network Security Essentials: Applications and Standards*, Fourth. Prentice Hall: Pearson, 2013.

[3] C. F. Kerry and P. D. Gallagher, "Digital Signature Standard (DSS)," *Fed. Inf. Process. Stand. Publ.*, vol. 186–4, no. July, 2013 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[4] F. Kurniawan, A. Kusyanti, and H. Nurwarsito, "Analisis dan Implementasi Algoritma SHA-1 dan SHA-3 pada Sistem Autentikasi Garuda Training Cost," vol. 1, no. 9, pp. 803–812, 2017 [Online]. Available: http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/download/247/110/

[5] M. J. Dworkin, "FIPS PUB 202 SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions," *NIST PUB Ser.*, vol. 202, no. August 04, 2015 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[6] M. A. Patil and P. T. Karule, "Design and implementation of keccak hash function for cryptography," in *2015 International Conference on Communications and Signal Processing (ICCSP)*, 2015, pp. 0875–0878 [Online]. Available: http://ieeexplore.ieee.org/document/7322620/

[7] R. A. Azdy, "Tanda tangan Digital Menggunakan Algoritme Keccak dan RSA," *Jnteti*, vol. 5, no. 3, pp. 184–191, 2016 [Online]. Available: http://ejnteti.jteti.ugm.ac.id/index.php/JNTETI/article/download/255/190

[8] H. Gross, D. Schaffenrath, and S. Mangard, "Higher-Order Side-Channel Protected Implementations of KECCAK," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 205–212 [Online]. Available: http://ieeexplore.ieee.org/document/8049787/

[9] H. Mestiri, F. Kahri, B. Bouallegue, M. Marzougui, and M. Machhout, "Efficient countermeasure for reliable KECCAK architecture against fault attacks," in *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*, 2017, pp. 55–59 [Online]. Available: http://ieeexplore.ieee.org/document/7905263/

[10] A. R. Anggoro, "Studi Mengenai Fully Homomorphic Encryption dan Perkembangannya dari RSA sebagai Enkripsi Homomorfis Populer," 2009 [Online]. Available: http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2009-2010/Makalah1/Makalah1_IF3058_2010_069.pdf