Journal of Engineering and Applied Sciences 14 (13): 4435-4441, 2019
ISSN: 1816-949X

Integrity Checking of Several Program Codes

Abdullah Th. Abdalsatir and Ali J. Abboud
Department of Computer Engineering, College of Engineering, University of Diyala, Baqubah, Iraq

Abstract: Integrity checking of software programs are becoming the most serious concern for industry and academic institutions. The reason of this concern is the high interest of adversaries in altering or modifying software programs according to their interest. The best to counterfeit these harmful attitude toward tampering software programs is to develop new integrity and authentication algorithms. Based on the earlier, we have proposed and suggested a method to check several programs simultaneously. This method divides the original software into several program codes and then compute integrity value of each program using one of selected integrity algorithms in this research. After that these integrity values of program codes are combined together to obtain single integrity value for all these program codes. Finally, the program codes and the combined integrity value are encrypted using RSA public key cryptographic algorithm. The results analysis of this proposed method proves its capability to provide robust integrity checking for several programs simultaneously.

Key words: Integrity checking, multiple program codes, MAC and cryptographic hash functions, academic institutions, authentication algorithms, cryptographic algorithm

INTRODUCTION

Software systems become the main controller of current mobile and ubiquitous computing devices in the current era of internet revolution. The software is embedded inside these apparatuses in the form of operating systems, applications or other controlling tools (Chen *et al*., 2002). Also, anyone of these software systems are composed of several program codes that executes different kind of tasks related to specific application. The integrity checking of these program codes is of utmost importance to the security and privacy of the devices that embed these programs (Asokan *et al*., 2018). The intruders work hard to break the software inside these devices in aim to modify it to their benefit for example, to bypass security checks, monitor these devices, steal credit card information, recording videos and voices. Hence, modifying or tampering software programs can result in devastating consequences on the performance of embedded software inside device (Cappaert and Preneel, 2010). Smart cards, mobile phones, routers, television receivers, Point of Sale (POS), Automatic Teller Machines (ATM) are few examples of these devices that can be breached by attackers (Cappaert and Preneel, 2010).

Viruses, worms, Trojan horses are examples of malicious software that try to tamper original device program codes in some either injecting themselves inside device software by modifying it static program code or changing the execution path of the software (Kirovski *et al*., 2002). These malicious programs penetrate through the weakest point in the software systems with aim finally, to control on all resources of the device. To prevent such adversary activities, we need to develop authentication, integrity and confidentiality checking measures to deviate these threats away from our device entirely (Chen *et al*., 2002). In this study, we will focus on the integrity of several program codes only and other 2 measures will be explored in other research studies. The integrity of several program codes that constitute the whole software of the computing device is a challenging task, since, sometimes these codes are distributed at different locations and integrated on the device to obtain final software (Spinellis, 2000). Now a days, the mobile phone software applications are available in the Apple store and Google play and the user can download any application easily and installed on his/her device. However, the integrity of these applications is a major concern for developers and users, since, attackers can tamper these application at program code or execution level (Graunke and Rozas, 2000). Hence, there is urgent need to check the integrity of applications program codes before install them on mobile phone devices. The topic of integrity verification is considered currently one of important issues of mobile phone security.

Data integrity checking algorithms: The main the task of data integrity algorithms is checking whether the

Corresponding Author: Abdullah Th. Abdalsatir, Department of Computer Engineering, College of Engineering, University of Diyala, Baqubah, Iraq

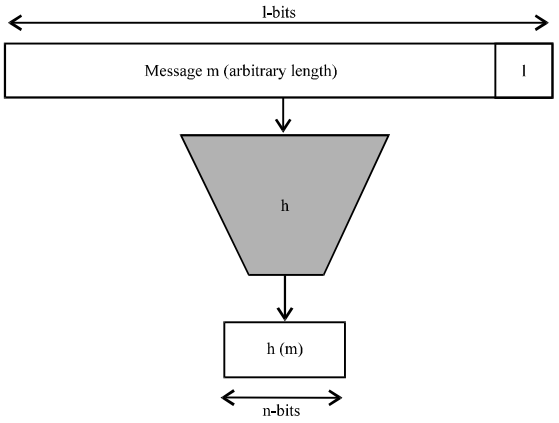

Fig. 1: Hash function

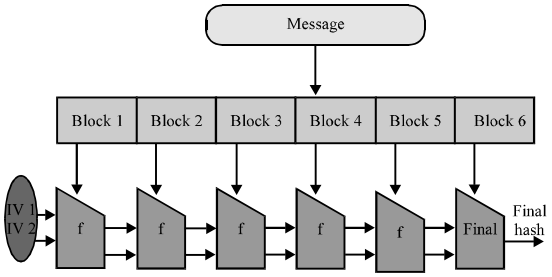

Fig. 2: Merkle-damgard construction of hash function (Lindell and Katz, 2014)

alteration happened on the data in the computing device. Also, they verify the integrity of data transmitted through computer networked communication channels and the internet. There are 3 types of data integrity algorithms as explained briefly below:

Hash functions: Hash functions are the mathematical algorithms used to produce fixed length output for arbitrary length software program codes as shown in the Fig. 1. The characteristics of these algorithms (Stallings, 2006) are producing fixed short hash digest for arbitrarily length message it is easy to compute integrity value it is very difficult to find original data if the attacker have hash value must be resistant to different kinds of attacks. Wide pipe, sponge function and the Hash Iterated Framework (HAIFA) are the 3 main categories of hash function constructions. One example of these hash function constructions is Merkle-Damgard construction as shown in the Fig. 2 as Lindell and Katz (2014).

The hash functions should be at least resistant to the preimage, second preimage and collision attacks. The resistance to the preimage attack is achieved by making the task of attacker very difficult to find (M) given h (M) while the resistance to the second preimage attack is

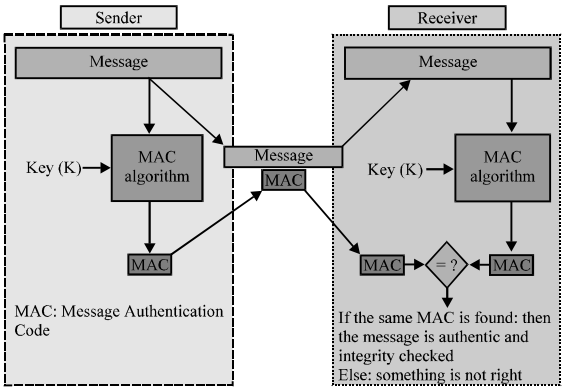

Fig. 3: Message Authentication Code (MAC) (Lindell and Katz, 2014)

accomplished by making very hard to attacker to find h (M2) = h (M1) if he has M2 and h (M2). Finally, the resistance to the collision attack is achieved by making hard to attacker to find 2 messages with the same hash value (i.e., h (M1) = h (M2)).

Message Authentication Code (MAC): Verifying the origin and integrity of messages (program codes) is the responsibility of message authentication (MAC) algorithms attacks (Lindell and Katz, 2014). The origin of the message means the source that generates the message, for example, the producer of the software package while the integrity means that data does not change and remained without any modifications. Hash functions, block ciphers are the most popular ways to obtain MAC values but they are need an encryption key to do their process attacks (Stallings, 2006). Hence, sometimes the MACs are called keyed-hashes because there is key used in producing the final MAC as shown in the Fig. 3 below attacks (Lindell and Katz, 2014).

The MAC is called commonly a tag and mostly used in money transfer from financial institutions. It is also attached to the end of message sent by user to the bank to check the integrity and origin of this message and comes from intended user without any change on it through communication channel (Paar and Pelzl, 2009). We can use also MAC for checking origin and integrity of software program codes and HMAC is an example of most used MAC algorithms for verifying software integrity and origin as shown below in the Fig. 4 (Paar and Pelzl, 2009).

Digital signatures: Digital signatures are the best advancement in the cryptography and information security. It can provide several security services, simultaneously. Such of these services are authentication integrity and non-repudiation as shown in the Fig. 5. It is also very useful and candidate

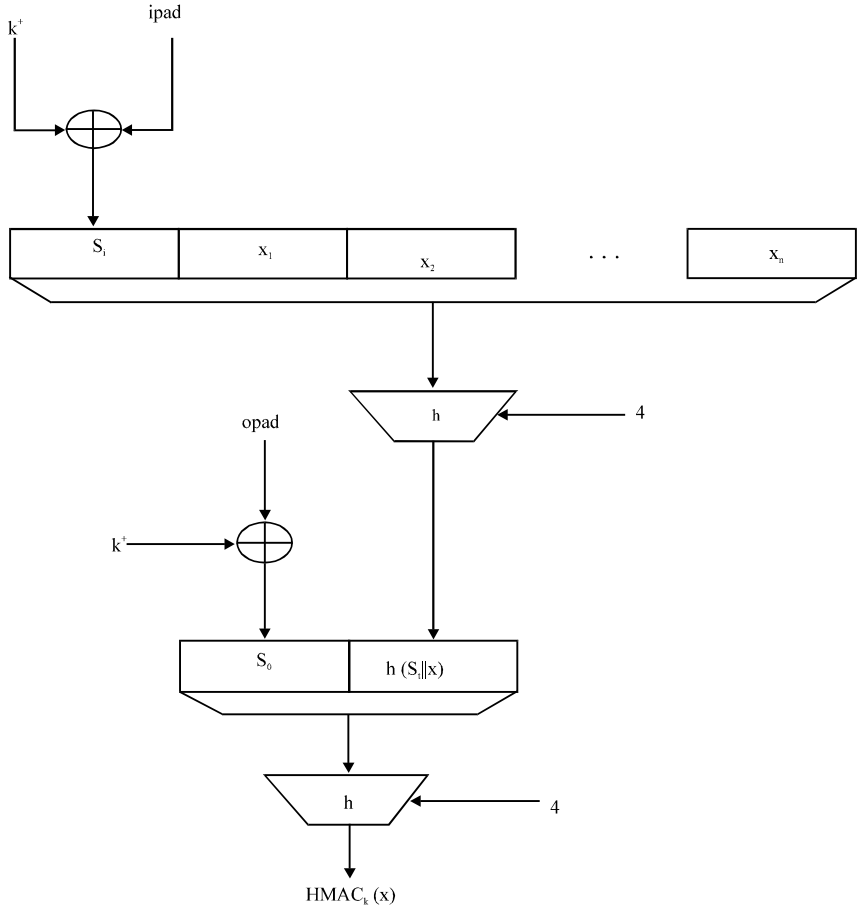

Fig. 4: MAC based on hash function (HMAC) (Paar and Pelzl, 2009)

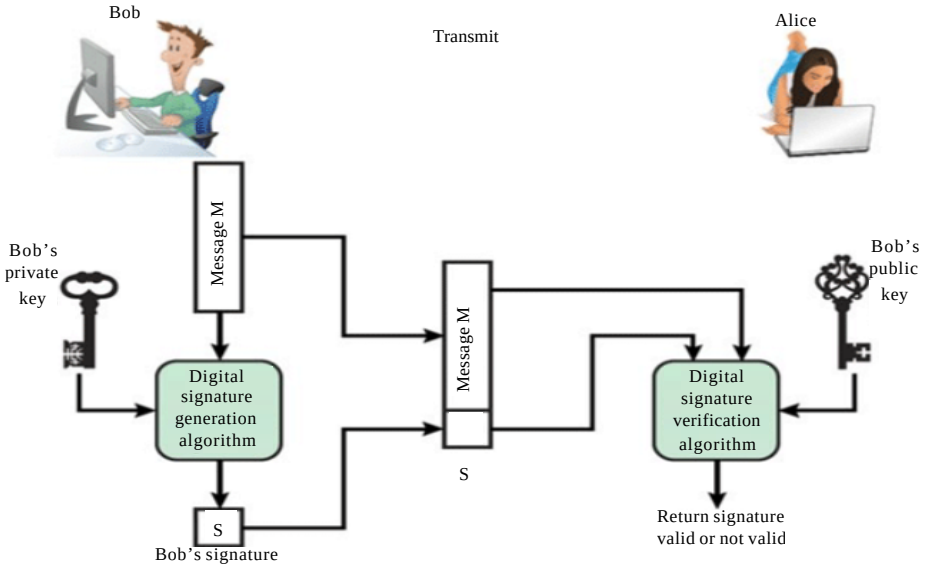

Fig. 5: Digital signature process (Stallings, 2006)

mechanism that can be used to check the integrity of several program codes in the application software (Stallings, 2006).

Digital signatures use public key cryptography concepts to provide their security services to the users. In public cryptography, public and private keys are the 2 main keys to encrypt and decrypt data. A practical example of signing and verifying electronic file is illustrated in Fig. 6 (Lindell and Katz, 2014).

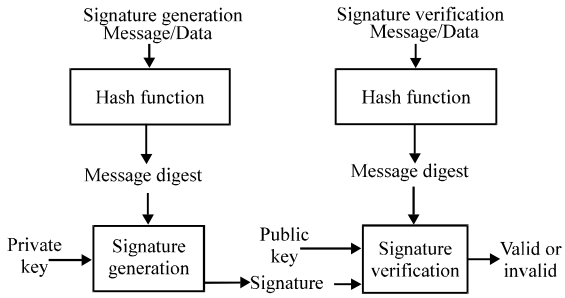

Fig. 6: An example of signing and verify processes in the digital signatures (Lindell and Katz, 2014)

MATERIALS AND METHODS

Proposed integrity checking method: We have proposed in this research a method to check the integrity of software that consists of several program codes. In this method, we have used the data integrity checking algorithms mentioned in the last section. Also, public key cryptography algorithms are used in our proposed method to provide confidentiality to the hash values and the architecture of this proposed method is shown in the Fig. 7.

The above framework to check the integrity of several program codes is designed based on the using RSA public key cryptographic algorithm. We have noticed in this method the original software is divided into several program codes numbered sequentially (1-N). In the integrity values generation phase, the integrity value of each program code is calculated using integrity function (either hash function, MAC or digital signature). Then all these integrity values are XORed together to obtain total integrity value. RSA cryptographic algorithm is used to encrypt the total integrity value (i.e., integrity value of all program codes) by using public key. However, in the

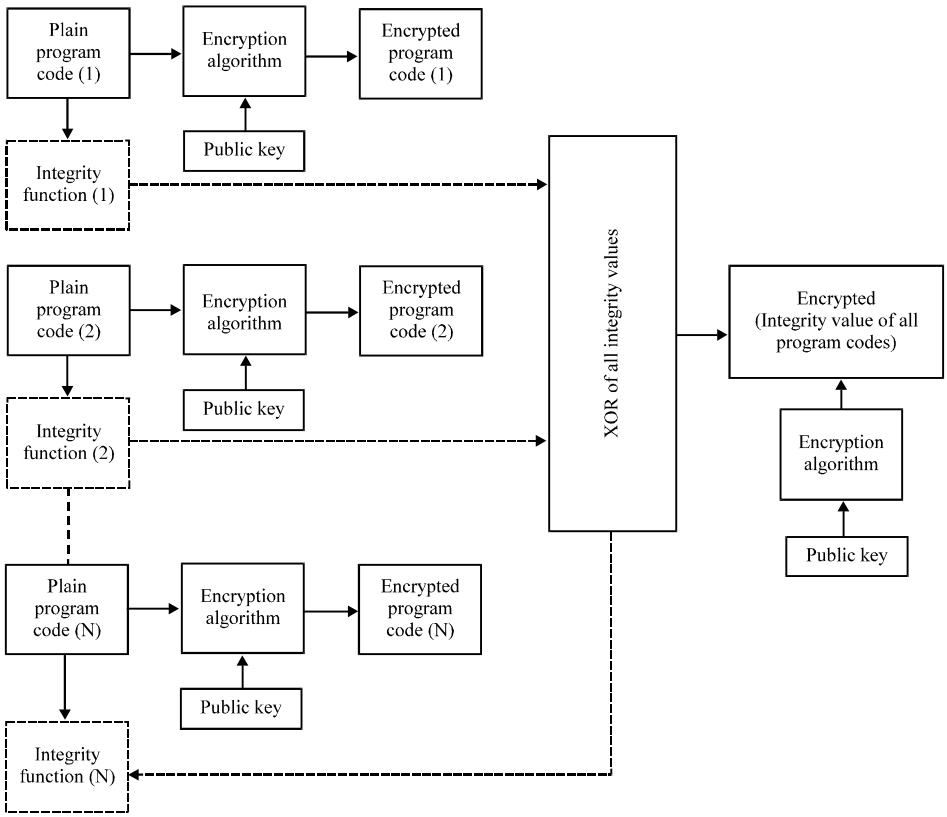

Fig. 7: Integrity checking of several program codes (Integrity values generation phase)

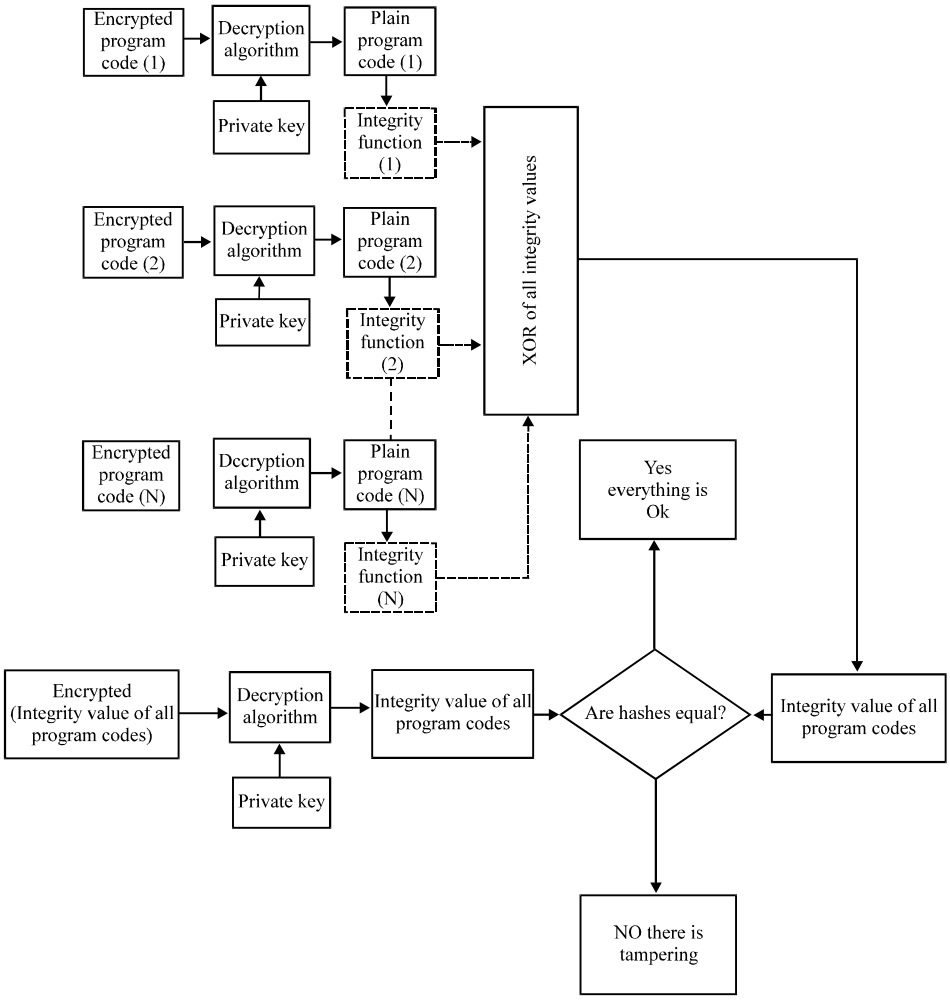

Fig. 8: Integrity checking of several program codes (Verification Phase)

integrity checking phase as shown in the Fig. 8, RSA cryptographic algorithm is used to decrypt the encrypted program codes by utilizing private key. Then, the integrity of each program code is calculated again and all program codes integrity values XORed together to obtain new total integrity value. This new value is compared with deciphered old total integrity value. If there is match between 2 values, then we can assure that there is no tampering or alteration happened on the program codes. Otherwise, if mismatch is occurred between them, we conclude that modification or alteration happened on the program codes.

RESULTS AND DISCUSSION

We have C++ program that multiply 2 matrices as a testing software. It is divided into 3 program codes as shown below with aim to check their integrity. We have done several tests on this software by using several integrity checking algorithms.

Algorithm 1; Program code 1:

```
#include <iostream>
using namespace std
int main ()
{
    int a [10][10], b [10][10], mult [10][10], r1, c1, r2, c2, I, j, k
    count << "Enter rows and columns for first matrix
    cin>>r1>>c1
    count << "Enter rows and columns for first matrix"
    cin>>r2>>c2
    //If columns of first matrix in not equal to row of second matrix
    //Ask the user to enter the size of matrix again
    while (c1! =r2)
    {
    Count<< "Error! Column of first matrix not equal to row of second"

    Count<< "Enter rows and columns for first matrix:"
    cin >>r1>> c1
```

```
      Count<< "Enter rows and columns for second matrix:" cin >>r2>>
c2
}

//Storing elemnts of first matrix
Count<<endl<< "Enter elements of matrix 1:" <<endl
for (i = 0; i<r1; ++i)
      for (j = 0; j<c1; ++j)
      {
         Count<< "Enter element a" << I+1<<j+1<< ":"
         Cin>>a [i] [j]
      }
```

Algorithm 2; Program code 2:

```
//Storing elements of second matrix.
 count <<endl>> "Enter elements of m,atrix 2:" <<endl
 for (i = 0; i<r2; ++I)
   for (j = 0; j<c2; ++j)
   {
      count<< "Enter element b" <<I+1<<j+1<< ":"
      cin>>b [i] [j]
   }

//Initializing elements of matrix mult to 0.
 for (i = 0; i<r2; ++i )
      for (j = 0; j<c2; ++j)
      {
        Mult [I][j] = 0
      }
//Multiplying matrix a and b and storing in array mult
 for (i = 0; i<r1; ++I)
 for (j = 0; j<c2; ++j)
      for (k = 0; k<c1; ++k)
      {
         Mult [I][j]+ = a[i][k]*b[k][j]
      }
```

Algorithm 3; Program code 3:

```
//Displaying the multiplication of two matrix.
      Count<<endl<< "Output matrix:" <<endl
      for (i = 0; i<r1; ++I)
      for (j = 0; j<c2; ++j)
      {
        count<< ""<<mult [I][j]
        if (j = = c2-1)
          count<<endl
      }

 return ()
      }
```

Table 1-3 are presented to show the hash values of several integrity algorithms for program codes 1-3, respectively and the hash checksums in these tables are in hexadecimal format. The lengths of these hashes are {160, 256, 384, 512, 128, 128} bits for SHA-1, SHA-256, SHA-384, SHA-512, MD2, MD5 algorithms, respectively. However, Table 4 is used to presented the total integrity value that resulted from XOR of all program codes integrity values.

Table 1: Integrity values of the program code 1 using several integrity algorithms

Integrity algorithm	Program code 1
SHA-1	A7FEBCC93ED9C2C5F1563F7AB200B1B76B1F684B
SHA-256	27E279F055D460D9F14F08906C893264019151828072727FA053DDA378B5E109
SHA-384	C86A28E4ED2BD175982F3B39F10CA73EE7BF9BA81BB01515EAD 93471E03366D50802771F15808AEA3FE7F5F9EA412691
SHA-512	F4527FFE804CD7F17A53FB43FC23D92D5664A54B4DDB38BF7C63679E5F70CA696E 845A4237AAF6130D3D9AC9AC4AE395ACBA5970BC7121B7C7081CE303ADA27F
MD2	61469E75E980ECAE1266BABDACC4C787
MD5	1DCFE5B4DB9D632FED190070FC31433E

Table 2: Integrity values of the program code 2 using several integrity algorithms

Integrity algorithm	Program code 2
SHA-1	6434839F302A7010087C13EB59DFD4F4484B51DE
SHA-256	5B3D31CE72CD79D3A57F159C25EE48789DC0ACA341DC34EDC3697B33A94AAAFF
SHA-384	76D3C1E65BAF9F062BEBB336625AB44ECF20A4CAA5811739C 96F5EAAC5988AFA3B9D7195DF2F07E9939006607E903E30
SHA-512	47ED1646C3AC9455B0F0FCE9D10FF55CC9C9E70A2C1C1DA 5654036761A9F57C63738F4BE02CEA12E949EFC54E12BFBB 10712A34ED765F3D6FEECFF211331F375
MD2	BDD466051D4075230134A9C01F03AC3F
MD5	51886366DEE7ECD37FC9F01C48116BCA

Table 3: Integrity values of the program code 2 using several integrity algorithms

Integrity agorithm	Program code 3
SHA-1	443A75918B2B50395553E5329ECEF1E61DE584BE
SHA-256	A7F960A99773E6BFAC107BF09ED27EBDDAC895E70F9AB4131787DCB31E6833A3
SHA-384	DDEBF2EE90DCDF913AC37F648FFE3BDBED0872C69C3760B4FB9E1712DBBC88EE085A 84F8442DF18CB0442E6D89AC6AD0
SHA-512	5CD266F5EFF379F62D32FE2EACD293D77E9BE99433301494934793AC3BF2CFD61A76C 599EAA77410C14F0DB5A0741153583FDB01CC235A8464C8C174F49FE5F1
MD2	B117DFDB36C364B8401A82EDB3AB80DC
MD5	1A6E0A2C085E6201CB768748E218A011

Table 4: Total integrity values of all program codes using several integrity algorithms

Integrity algorithm	Total integrity value for 3 program codes
SHA-1	8F4C8DEEA7CA719A3BB270A7582CC9935145E1DB
SHA-256	D229B6FBF26FDB0A496CC3F87B72C975B6870AF5806C75416986E4214DA3F785
SHA-384	651E259E80F61A2AC94A2069D27CF16C3C87887813DF177732BC6812977BC88B57 D4262888D9E741B522E2CFC3D4DD21
SHA-512	E604A135E9F88FBAE3AD5F3886C471574C66DC225E62E107F923A38E522BE0B FFDFDC3A271941EF616B5271EA424ED293AB5F33D8CB8D597371F7785DC26434B
MD2	682AC0F3549906E6D57B23D538100CB4
MD5	528F02EF5A72538E69CED4DD96746885

CONCLUSION

Integrity checking of software programs is important topic for industry and academic. In this research study, we proposed a method to check integrity of several program codes simultaneously with aim to prevent modification of these programs. We have used several integrity algorithms in our tests including SHA-1, SHA-256, SHA-384, SHA-512, MD2 and MD5. Also, RSA public key cryptographic algorithm is used to provide confidentiality to the program codes and the total integrity value. The main conclusion from this study is that one integrity algorithm not sufficient to provide robust integrity checking for the software programs and should be several algorithms with different characteristics are used together to secure program codes.

ACKNOWLEDGEMENT

We would like to thank the College of Engineering in the University of Diyala for supporting our research by various types of help. Also, We thank our colleagues in the department of computer engineering for providing all necessary research equipment's and tools.

REFERENCES

Asokan, N., J. Mantyla and R. Serafat, 2018. Method and device for verifying the integrity of platform software of an electronic device. USPTO. Washington, DC., USA. https:// patents. google.com /pat ent/U S988 1150B 2/en

Cappaert, J. and B. Preneel, 2010. A general model for hiding control flow. Proceedings of the ACM 10th Annual International Workshop on Digital Rights Management (DRM '10), October 4, 2010, ACM, New York, USA., ISBN:978-1-4503-0091-9, pp: 35-42.

Chen, Y., R. Venkatesan, M. Cary, R. Pang and S. Sinha *et al*., 2002. Oblivious hashing: A stealthy software integrity verification primitive. Proceedings of the 5th International Workshop on Information Hiding (IH'02), October 7-9, 2002, Springer, Noordwijkerhout, The Netherlands, ISBN:978-3-540-00421-9, pp: 400-414.

Graunke, G.L. and C .V. Rozas, 2000. Method and apparatus for integrity verification, authentication and secure linkage of software modules. Patent and Trademark Office, Washington, DC., USA. https: //patents .goog le.com/ patent /US610 5137A/en

Kirovski, D., M. Drinic and M. Potkonjak, 2002. Enabling trusted software integrity. ACM. Sigplan Not., 37: 108-120.

Lindell, Y. and J. Katz, 2014. Introduction to Modern Cryptography. 2nd Edn., CRC Press, Boca Raton, Boca Raton, USA., ISBN:9781466570269, Pages: 583.

Paar, C. and J. Pelzl, 2009. Understanding Cryptography: A Textbook for Students and Practitioners. Springer, New York, USA., ISBN:978-3-642-44649-8, Pages: 367.

Spinellis, D., 2000. Reflection as a mechanism for software integrity verification. ACM. Trans. Inf. Syst. Sec., 3: 51-62.

Stallings, W., 2006. Cryptography and Network Security. 4th Edn., Prentice Hall, New Jersey, USA., ISBN-13: 9780131873162, Pages: 680.