ARUS: Pseudo-Asymmetric Encryption Algorithm

Lazuardi Firdaus
School of *Electrical Engineering and Informatics*
Bandung Institute of Technology
Bandung, Indonesia
lazuardifirdaus369@yahoo.com

Abstract— **ARUS is an encryption algorithm made using AES, RSA, and SHA2. The aim of ARUS is to provide the functionality of RSA while having the encryption time of AES. Even though in theory such goal was to be achieved, the testing however has shown that ARUS need further development, implementation, and testing.**

Keywords—encryption, aim, functionality, time

I. Introduction

The introduction of asymmetric encryption algorithm in the 20th century opens up whole new possibilities in the cryptography world. For the first time, it is possible to encrypt messages while distributing the key for the whole world to use. This is because encryption and decryption use a set of 2 different keys. Allowing other people to encrypt a message but preventing them from decrypting it (or vice versa). Thus, eliminating the problem that persist on symmetric encryption algorithm which is the transportation of keys between sender and receiver.

Asymmetric encryption algorithm, however, is not without its cons. Many asymmetric encryption algorithm such as RSA and ECC requires more time to encrypt messages than its symmetric counterpart. Thus limiting its usability when dealing with a large size of messages.

A solution to this problem is to somehow combine the two approach. Acquiring the encryption time of symmetric encryption while pertaining the characteristics of asymmetric encryption of using different keys for encryption and decryption. ARUS is an encryption algorithm made exactly just for that.

ARUS, which stands for AES RSA Union SHA, combines the encryption speed of AES and the sets of keys used by RSA. ARUS can optionally uses SHA to help maintain the integrity of the sent message.

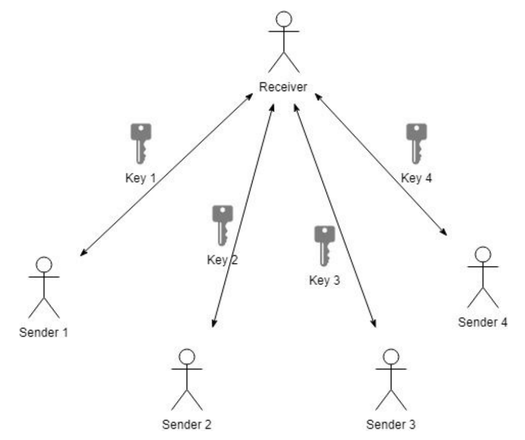
II. Basic Theory

A. Symmetric Encryption

The more traditional of the two, Symmetric Encryption Algorithm dates back way back to the rise of the Roman Empire. Caesar cipher, one of the oldest cryptographic algorithm, was used by Julius Caesar to protect messages of military importance.

The term 'Symmetric' comes from the fact that the key used for encryption are also used for decryption. Whoever has access to the key are able to both encrypt and decrypt a message. This means that the confidentiality of the key is crucial.

To decrypt a message one needs to know the key used to encrypt the message. The sender and the receiver of the message must agree on the said key beforehand. The problem lies on how to communicate the key between the sender and the receiver. The most straightforward and perhaps the safest method is to meet physically. This however is simply not possible in many case. Just simply sending the key through a mediator raises the chance of it being intercepted thus rendering the encryption unsafe.



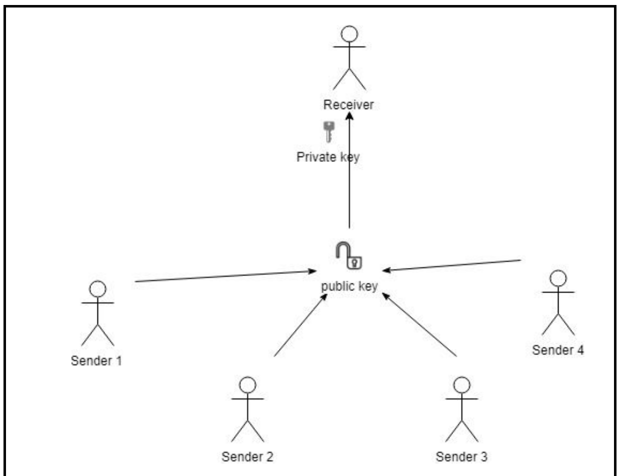
When using symmetric encryption to communicate, a key is needed on every pair of sender receiver

B. Asymmetric Encryption

'Asymmetric' comes from the fact that the key used to encrypt a message is different from the key used the decrypt it. One of the two key can be made open to public (named public key), while the other be kept in secret (named private key).

Asymmetric encryption brings up many utility. When the encryption key is made public and decryption key is made private, multiple sender can send message to the same receiver without fear of anyone but the receiver can read the message.

When decryption key is the one made public and encryption key made private, encrypted message can be a way of authenticating the sender because none but the holder of the private key can encrypt the message in a way that can be decrypted using the decryption key.



When communicating using asymmetric encryption, a receiver only needs a pair of keys

C. Hashing and Messages Integrity

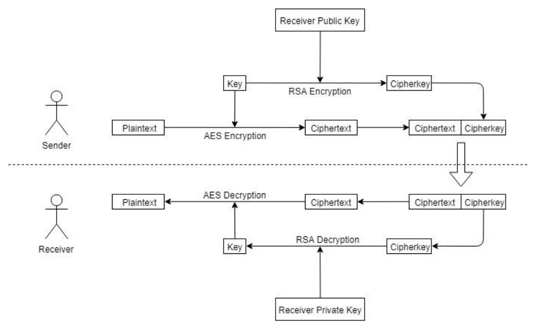
Hashing function is a function that receives any amount of string and produce a fixed amount of digest. The length of the digest is varied from one hashing algorithm to the other. Unlike encryption, hashing function accept no key and the process is irreversible. We cannot deduct for sure the input of the function based on the digest alone.

The most common use of hashing function is to ensure the integrity of messages. This can be done because a small amount of changes on the input can produce an entirely different digest. Hashing digest of the messages can be added to the messages itself. When the receiver receives the message he/she can try to reproduce the same digest using the original message. If the digest is the same as the one included, very most likely the original message is unchanged thus proving the integrity of the message.

III. ARUS Pseudo-Asymmetric Encryption

A. Basic Idea

The goal of ARUS is to provide the function or characteristics of asymmetric encryption while having an almost similar encryption time to symmetric encryption, especially on large messages. Such things can be done by actually encrypting the original message using symmetric encryption using a generated key. The key can be generated randomly or by using a digest of a hash function, using the original message as the input. The key is then encrypted using asymmetric encryption and is added to the encrypted message.



ARUS flowchart in general

Theoretically, the time needed to encrypt a message using ARUS would be similar to the encryption time of AES especially on longer messages. The RSA encryption only need to encrypt the key used to encrypt using AES which is a fixed length. If the key was generated using a hash function, the time needed to calculate the hash was also added to the calculation.

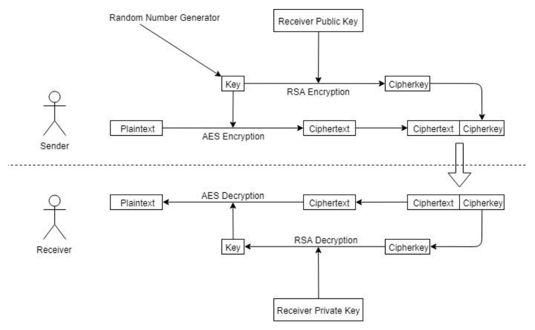
ARUS uses AES for its symmetric encryption component, RSA for its asymmetric encryption component, and SHA2 for its hash function component. Hence the name ARUS (AES RSA Union SHA).

B. ARUS Modes

ARUS can encrypt messages on 2 modes: random mode and hash mode. Regardless of the modes, ARUS can decrypt encrypted messages the same way. Hash mode, however, provides an additional feature of message integrity checking. Adding more security to the message.

1) *Random Mode*

When encrypting in random mode, the key for AES encryption is generated randomly. This is a more straightforward solution than Hash mode. Random mode also can encrypt faster than Hash mode because the key is generated randomly instead of calculated from a hash function. The use of random mode can also confuse third party because the same message can have a whole different form when encrypted because of the random number.

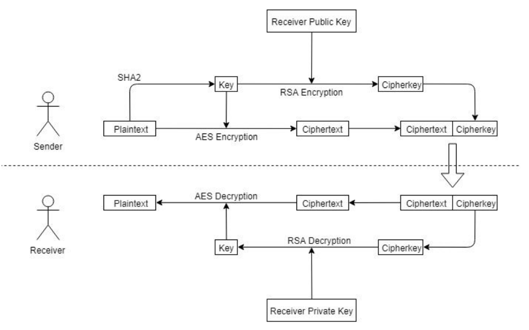


ARUS flowchart in random mode

The downside of Random mode is the possibility that the random number generator can be guessed. If a third party can somehow guess the number calculated by the random number generator used to generate the AES key, that third party can have access to the original unencrypted message. Risking the confidentiality and integrity of the message.

2) *Hash Mode*

In hash mode, the key for the AES encryption is generated from the hash of the unencrypted message. Using hash digest as key means that the receiver can check message integrity just by comparing the hash digest of the decrypted message with the key itself. Hash mode eliminates the weakness of Random mode which is the possibility that the random number generator can be guessed.



ARUS flowchart in hash mode

The downside of using hash mode is that it needs more time to encrypt a message than Random mode. The algorithm needs additional time to calculate the hash digest of the message before encrypting it with AES.

IV. Simulation

A. Implementation

An implementation for ARUS, both Random mode and Hash mode, was made using Ruby. AES, RSA, and SHA2 implementation was also made as part for ARUS implementation. The source code can be accessed on the following page https://github.com/LazuliSound/PseudoAsymmetric

B. Testing: encryption and decryption example

Testings for the implementation of ARUS was done to ensure the success of the encryption and the decryption process. 3 messages of varying length was to be encrypted using the implementation of ARUS in random mode and hash mode. The 3 messages, both unencrypted, encrypted, and decrypted as well as the public and private key used are as follow.

1) *Random mode*

Public Key: 13822674922986224941,27246841126251727704183438597338711409713822674922986224941,272468411262517277041834385973387114097		
Plaintext: This is a text	Ciphertext: 22911238842440881 02866593144425205 61572257088922547 03623918895510662 86496675840000000 00000000000000000 000000000000000▒^ C▒{▒E▒	Decrypted: This is a text
Public Key: 13888891667212343329,232243904596404327712266002337569603687 Private Key: 216738412961773588578140407735232067649,232243904596404327712266002337569603687		
Plaintext: Out of the night that covers me	Ciphertext: 16164104642445505 39333203452095474 10403165233222874 15964916246334278 71651418620000000 00000000000000000 000000000000000▒! ▒ws© ▒bK▒WF[(▒[~ ▒	Decrypted: Out of the night that covers me
Public Key: 17832816390864117679,273148042566786783456309798902533403453 Private Key: 87447784404735358708839532063233318187,273148042566786783456309798902533403453		
Plaintext: Out of the night that covers me Black as the pit from pole to pole I thank whatever gods may be For my unconquerable soul	Ciphertext: 07775218539535169 97189995639190606 78621009299496183 71810721860835774 67587609(▒,▒*Ū▒`y `▒a▒*▒ow9o▒L ▒b1B▒d▒rqC ▒,mq▒"A▒`▒I N▒^▒%z▒4 6▒Xnή▒2	Decrypted: Out of the night that covers me Black as the pit from pole to pole I thank whatever gods may be For my unconquerable soul

2) *Hash mode*

Public Key: 13577236990676810111,219528860184670269831763672654387382741 Private Key: 7698439096039124922374455302297696203,219528860184670269831763672654387382741		
Plaintext: This is a text	Ciphertext: 09018994489276925 56973072403531877 02353168081746730 52288276268199918 92760258530000000 00000000000000000 000000000000000▒ ▒L ▒J▒	Decrypted: This is a text
Public Key: 14042930197723137587,321771280091682056852496023443332778697		

Private Key: 3194424348401856597136873244714384771O3,321771280091682056852496023443332778697		
Plaintext: Out of the night that covers me	Ciphertext: 20562471795849631 71135984136460087 79469162939427533 44509969948164549 89086996310000000 00000000000000000 000000000000000▒ ▒`Z▒▒b▒.▒ ▒b▒P▒_3▒.▒	Decrypted: Out of the night that covers me
Public Key: 146105790329882500031,267404012391213913974126345131042028967 Private Key: 2524363186105084726464794584768988967 1,267404012391213913974126345131042028967		
Plaintext: Out of the night that covers me Black as the pit from pole to pole I thank whatever gods may be For my unconquerable soul	Ciphertext: 08240100258238092 53128921887365593 66688020948775689 65137116833610622 63599404iN`▒A3▒ .a|▒vQ▒}▒y▒ 00000000000000000 04▒□▒ja▒  `I▒g▒ϰ▒D▒zC▒ϴ(▒-▒|▒fW▒g▒ ▒Mz6▒:▒\▒ ▒DL▒響 wcX▒R/뮟▒	Decrypted: Out of the night that covers me Black as the pit from pole to pole I thank whatever gods may be For my unconquerable soul

C. Testing: encryption time

Testings for the implementation of ARUS was carried. The testing was done on a computer with specification:

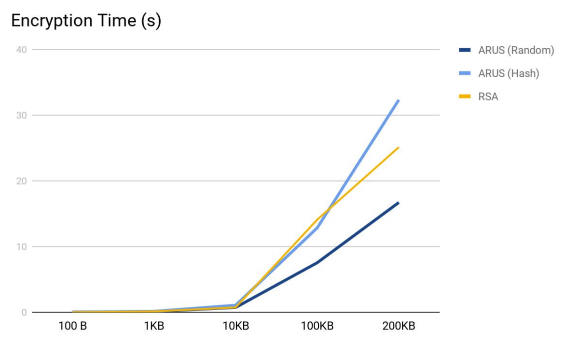
- Lenovo g40
- Windows 10
- Intel Core i5-4210u (4 CPUs ~2.4Ghz)
- 6GB RAM
- HDD

The testing was done by encrypting and decrypting 5 messages of varying size using ARUS random mode, ARUS hash mode, and RSA. Each file size uses a different set of randomly generated private and public key. The key was shared between ARUS random, Arus hash, and RSA encryption to ensure fair comparison. Each key was 1024 bit long.

The time needed to encrypt and decrypt each message size and each encryption method will be calculated. The decrypted message will also be checked to ensure that the encryption and decryption was a success

Such was the result of the test.

Message Size: 100 Byte		
ARUS Encrypt Time (Random Mode): 0.018844 s	ARUS Encrypt Time (Hash Mode): 0.037255 s	RSA Encrypt Time: 0.018281 s
ARUS Decrypt Time (Random Mode): 0.052994 s	ARUS Decrypt Time (Hash Mode): 0.093011 s	RSA Decrypt Time: 0.028192 s
Message Size: 1 Kilo Byte		
ARUS Encrypt Time (Random Mode): 0.119642 s	ARUS Encrypt Time (Hash Mode): 0.152562 s	RSA Encrypt Time: 0.136195 s
ARUS Decrypt Time (Random Mode): 0.384682 s	ARUS Decrypt Time (Hash Mode): 0.282001 s	RSA Decrypt Time: 0.128893 s
Message Size: 10 KiloByte		
ARUS Encrypt Time (Random Mode): 0.741534 s	ARUS Encrypt Time (Hash Mode): 1.082473 s	RSA Encrypt Time: 0.741873 s
ARUS Decrypt Time (Random Mode): 2.571902 s	ARUS Decrypt Time (Hash Mode): 2.421922 s	RSA Decrypt Time: 1.044429 s
Message Size: 100 KiloByte		
ARUS Encrypt Time (Random Mode): 7.548439 s	ARUS Encrypt Time (Hash Mode): 12.850284 s	RSA Encrypt Time: 14.10012 s
ARUS Decrypt Time (Random Mode): 26.042851 s	ARUS Decrypt Time (Hash Mode): 25.411035 s	RSA Decrypt Time: 13.124358 s
Message Size: 200 KiloByte		
ARUS Encrypt Time (Random Mode): 16.713517 s	ARUS Encrypt Time (Hash Mode): 32.335428 s	RSA Encrypt Time: 25.130664 s
ARUS Decrypt Time (Random Mode): 49.857806 s	ARUS Decrypt Time (Hash Mode): 51.981146 s	RSA Decrypt Time: 26.316504 s



Comparison of ARUS(random), ARUS(hash), and RSA encryption time on various message size

V. Security

AES, RSA, and SHA2 each can be considered secure by today's standard when using a large enough keyspace. The integration of those 3 in the form of ARUS implementation ,however, may raise several security concern:

– When using random mode to generate AES key, there is a risk that the key can be compromised if the random number generator used to generate the key can be guessed. This can be overcome by using hash mode.

Nevertheless, when those weakness can be overcome, ARUS can be considered a secure alternative to RSA. This is as long as AES, RSA, and SHA2 can be considered secure.

VI. Conclusion

Theoretically, the implementation of ARUS will provide the use case of RSA with the speed of AES especially on larger messages. The test indicates that this may be true although on smaller messages ARUS took a similar time or worse to encrypt a message than RSA.

The addition of SHA2 to improve the security of ARUS however present a negative impact on the performance of ARUS when compared to traditional RSA according to test. This however may indicate that the implementation of SHA2 used in the experiment is just isn't as efficient as it should be. In theory the time needed to produce a digest of a message using hash function should be considerably faster than encrypting the same message. This proves that the development, implementation, and testing of ARUS can be improved even more.