Secure Data Flow Messaging on Web Socket using Rivest Code 6

Ika Oktavia Suzanti[1], Yoga Dwitya Pramudita[1] , Husnul Atho' Mubarok[1] and Dwi Kuswanto[1]
[1]Departement of Informatics Engineering, University of Trunojoyo Madura, Bangkalan, Indonesia

Keywords: Realtime System, Data Security, Cryptography.

Abstract: Websocket is a two-way real-time system communication protocol for web and mobile applications. By utilizing full-duplex communication, WebSocket is very effective in applications that require fast response times such as instant messaging, real-time gaming and multimedia data communication. Data security issues are important especially when it comes to financial, health to personal data of many people, but WebSocket technology does not provide security mechanisms such as connection authentication or message encryption, making it very vulnerable to passive attacks such as packet monitoring or sniffing by unauthorized parties. To solve these problems, can be applied an encryption method in data flow on communication lines used. Rivest Code 6 (RC6) is a cryptographic algorithm with a good avalanche effect of plaintext and keys, and a fast time in encryption and decryption process. This research aims to apply RC6 encryption method on websocket communication channel to better maintain confidentiality and authenticity of sent messages. Speed performance test results show that Rivest Code 6 (RC6) has an avalanche effect of 28.29% changes in plaintext and 52.28% changes in keys.

1 INTRODUCTION

Websocket is a real-time protocol communication that are often considered as an alternative method because it has advantages over request-response communication as common as Hyper Text Transfer Protocol (HTTP). These advantages include using a full duplex transmission model that makes communication between client and server can be done simultaneously, reducing delay rate in sending messages from sender to recipient (Latency) and easy to use API (Park et al., 2014) (Pimentel and Nickerson, 2012) (Tsai et al., 2019). Websocket can be implemented on a web browser, web server or on Android instant messaging application (Erkkilä, 2012).

On HTTP protocol communication between web browser (client) and web server (server) will only occur when there is a request from client, but when using websocket, it is allows the server to push latest information to client without prior requests (Tsai et al., 2019). By utilizing full-duplex communication, WebSocket is very effective in applications that require fast response times such as instant messaging, realtime gaming and multimedia data communication (Joshi, 2012). Some of websocket implementations are e-learning (Arora et al., 2016) portofolio response (Tsai et al., 2019) smarthome application (Soewitoa et al., 2019), medical application (Kubov et al., 2019) email address verifier (Weizhen et al., 2012) and real-time sharing screen (Darmawan et al., 2019).

Data security issues are important especially when it comes to financial, health to personal data of many people (Namasudra and Gandomi, 2020). Some public facilities often require personal data for authentication even though they also cannot guarantee the security of data that has been given (Dinesha and Kb, 2016). In its development, the challenge that must be faced when using websocket in an application is a security problem because focus of this technology is only on client server connectivity (Enghardt et al., 2019). WebSocket technology does not provide security mechanisms such as connection authentication or message encryption, so it is very vulnerable to passive attacks such as packet monitoring or sniffing by unauthorized parties. To solve these problems, can be applied an encryption method in a data stream on communications line used (Mishra and J, 2013) such as sent message from an instant messaging application.

Rivest Code 6 (RC6) is a cryptographic algorithm with a good avalanche effect of plaintext and key (Liu et al., 2017) and fast time in the

Suzanti, I., Pramudita, Y., Mubarok, H. and Kuswanto, D.
Secure Data Flow Messaging on Web Socket using Rivest Code 6.
DOI: 10.5220/0010305400003051
In *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies (CESIT 2020)*, pages 151-157
ISBN: 978-989-758-501-2

encryption and decryption process (Aggarwal, 2015). RC6 has been used for encryption in wireless communications (Mahidhar and Raut, 2016) image protection (Naeem et al., 2016) or data transfer in cloud computing (Bhardwaja et al., 2016). The RC6 algorithm adopts three primitive operations used in RC5, also using a 32-bit multiplication operation that can already be implemented on today's modern processors. This primitive operation is very effective to produce a "diffusion" effect or a more efficient distribution effect, operation results in RC6 algorithm being safer than RC5, multiplication operation is used to count number of bits rotated so that concept of data-dependent rotations can be implemented with more perfect by RC6 algorithm (Rivest et al., 1998) This research is aim to implementing RC6 encryption method, it is expected that data flow on websocket communication path used will be more secure in confidentiality and authenticity of messages sent.

2 METHOD

2.1 Encrypted Communication Design

In this research, encrypted communication was developed by implementing RC6 method as an algorithm for encoding messages carried out by exploiting frames in websocket packages. To be able to communicate with encrypted communication, all parties involved in communication must have the same type of encryption from both client and websocket server. Before the message is sent, encryption process will first be carried out. There are two main parts of encrypted communication system design on a websocket, namely:

1. Client A will be party sending the message and client B who will be the recipient of encrypted message.
2. Server applications that play a role in routing message delivery from client to client.

Users who act as senders can send information in the form of files or text, while users who act as recipients will receive the information sent. Message sending is done during websocket communication. The whole system works are:

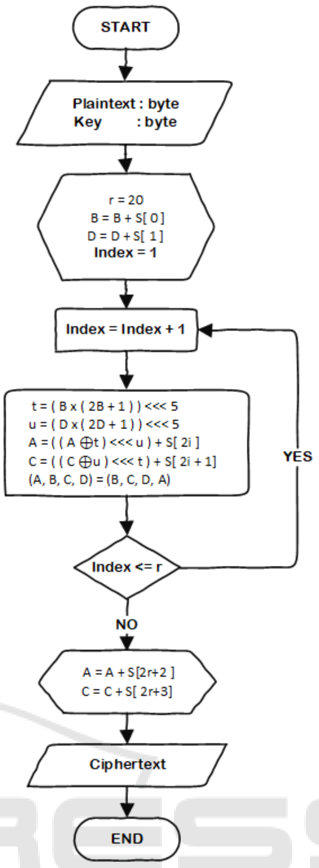
START
Plaintext : byte
Key : byte
r = 20
B = B + S[0]
D = D + S[1]
Index = 1
Index = Index + 1
t = (B x (2B + 1)) <<< 5
u = (D x (2D + 1)) <<< 5
A = ((A ⊕ t) <<< u) + S[2i]
C = ((C ⊕ u) <<< t) + S[2i + 1]
(A, B, C, D) = (B, C, D, A)
YES
Index <= r
NO
A = A + S[2r+2]
C = C + S[2r+3]
Ciphertext
END

Figure 1. RC6 Algorithm Encryption Process

1. Encrypted communication process starts from client sending encrypted messages. All types of secret messages both files and text are first converted into bytes. After message is successfully converted to byte format, message encryption process is carried out.
2. Encryption key is determined as soon as websocket implementation built both from client server and port number of each connection is added. After key is added with port number, it changed to a byte. In experiments conducted by, client and server are on the same local network so that port numbers used are also the same. If client and server are on different networks, then a static key should be used so that it can be encrypted by all connecting devices from sender to receiver.
3. The encryption process RC6 algorithm asks for two inputs of a plaintext and a byte type key. Details secret message encoding process is shown in Figure 1. After plaintext and key received, encryption process is then performed by :

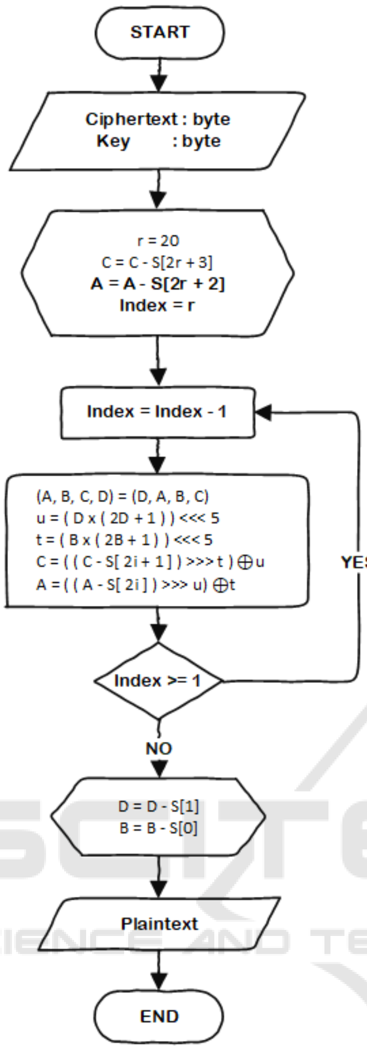
START
Ciphertext : byte
Key : byte
r = 20
C = C - S[2r + 3]
A = A - S[2r + 2]
Index = r
Index = Index - 1
(A, B, C, D) = (D, A, B, C)
u = (D x (2D + 1)) <<< 5
t = (B x (2B + 1)) <<< 5
C = ((C - S[2i + 1]) >>> t) ⊕ u
A = ((A - S[2i]) >>> u) ⊕ t
YES
Index >= 1
NO
D = D - S[1]
B = B - S[0]
Plaintext
END

Figure 2. RC6 Algorithm Decryption Process

1. Plaintext input and encryption key are in byte.
2. Variable initialization
a. Initialization variable r with a value of 20. Variable r is number of repetitions during encryption process.
b. Initialize variable B by summing value of variable B with internal key S index 0. Variable B is part of four register blocks A, B, C, and D.
c. Initializing variable D by adding up the value of variable D with the internal key S index 1.
d. Initialize index variable with a value of 1 as an iteration indicator.
3. The number of r repetitions, with r value is 20 turns.
4. The encryption process for RC6 algorithm. Each repetition in this algorithm follows the following rules, value of B is inserted into function f, which is defined as follows f (X) = X (2X + 1), then rotated left as far as 2log(w) or 5 bits. The results of this process are assumed to be t. The value of t is then XOR with value of A and result will be value A. The value of t is also used as a reference for C to rotate value to the left. Likewise for value of u is also used as a reference for value of A to do the leftist spinning process.
5. Check the number of repetitions that have been done.
6. After the looping process is finished, then value of variable A is carried out with key value S to 2r + 2 and value of C with key S to 2r + 3.
7. Ciphertext encryption results.
4. Furthermore, after encryption process at sending client is completed, encrypted message (ciphertext) will be sent to server.
5. When server receives an encrypted message (ciphertext) then message will be decrypted to find out information from message. Information needed by server is about message recipient. Details decryption process is shown in Figure 2. In message decryption process two inputs are required, which are encrypted messages (ciphertext) and encryption key then each of it will converted to a byte. At decryption process that occurs are :
1. Input ciphertext and encryption key in byte.
2. Variable initialization
a. Initialization variable r with a value of 20. Variable r is number of repetitions during encryption process.
b. Initialize variable C by summing value of variable C with internal key S index 0. Variable C is part of four register blocks A, B, C, and D.
c. Initializing variable A by adding up the value of variable A with the internal key S index 1.
d. Initialize index variable with a value of 1 as an iteration indicator.
3. The number of r repetitions, with r value is 20 turns.
4. The decryption process for RC6 algorithm. Each repetition in this algorithm follows the following rules, value of D is inserted into function f, which is defined as follows f (x) = x (2x + 1), then rotated left as far as 2log(w) or 5 bits. The results of this process are assumed to be t. The value of t is then XOR with value of D and result will be value D. The value of t is also used as a reference for C

to rotate value to the right. Likewise for value of u is also used as a reference for value of A to do right playback process.

5. Check the number of repetitions that have been done.
6. After the looping process is finished, then value of variable D is carried out with key value S to 2r + 2 and value of B with key S to 2r + 3.
7. Plaintext encryption process results.
6. After recipient's information is known by server, message will be back to encryption process and then encrypted message will be sent to receiving client.
7. Client receives an encrypted message from server then message will be decrypted to find out information sent by message sender.

2.2 Testing Scenario

2.2.1 Speed Performance Test

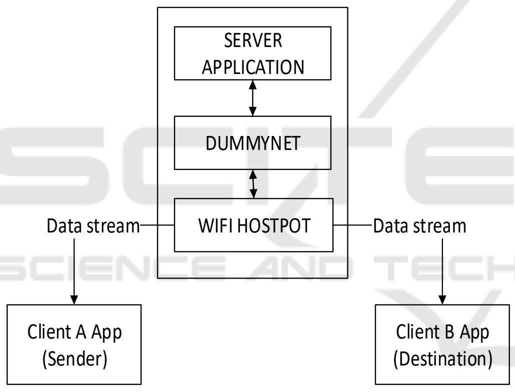

Figure 3. Speed performance test scenario.

Speed performance test is done by comparing time usage difference in message sending process, using Transport Layer Security (TLS) with certificate format “PKCS12” and Rivest Code 6 (RC6) encryption. Testing process is carried out using text and binary data with different sizes so that average value of overall test results will be obtained. In the speed performance test, Dummynet is used as a network simulator to simulate packet loss. Test scenario can be seen in Figure 3.

Message sending speed performance tests are carried out from sender to destination applications. Speed performance test has 5 packet loss scenarios, namely 0%, 3%, 6%, 9% and 12%. As well as 2 types of data that are 5 text and 5 binary data which has a different size. Table 1 shows types and sizes of data used in the process.

Table 1. Types and size of data test

No	Data types	size (*byte*)
1	*Text*	141
2	*Text*	325
3	*Text*	379
4	*Text*	612
5	*Text*	1320
6	*Binary*	732808
7	*Binary*	972524
8	*Binary*	1034296
9	*Binary*	1051521
10	*Binary*	2958132

2.2.2 Security Test

Security testing was carried out on RC6 and TLS algorithm by calculating avalanche effect of ciphertext. This test is done by changing for example one bit randomly from plaintext and making changes to one bit in encryption key. In this test, avalanche effect formula will be used to calculate complexity of encryption algorithm. The formula for calculating avalanche effect can be seen in equation 1

$$AE = \frac{N_c}{N_t} x100\% \quad (1)$$

AE is the Avalanche Effect value. If encryption is performed on plaintext or key, Nc is number of bits of ciphertext change and Nt is total number of bits of ciphertext. In this test encryption algorithm is said to be good if value of avalanche effect is around 45% - 60% or about half of total ciphertext.

3 RESULT AND DISCUSSION

3.1 Websocket Library Modifications

The web source source code library was obtained from Github site shared by Nathan Rajlich . After source code for websocket library is obtained, websocket library is modified. A modification was made to existing websocket program code to add RC6 algorithm program code as a data encryption method. There are two main programs that will be modified in WebsocketImpl.java and DefaultExtension.java files. Modifications made to source code of WebsocketImpl.java file are to add program code to generate keys based on initial key that was determined at the time of websocket's implementation . The key then added to port number of each client connection, for example initial key is "AripasolA" and connected port number is "13579" so generated key will be "AripasolA13579". Adding port number to key is done so that when there is one message being sent to different client connections it will produce a different ciphertext. The following is a source code added to modified WebsocketImpl.java file.

```
  private String getEncryptionKey(){
  String key = "AripasolA-";
  if (this.role == Role.CLIENT) {
  key += getLocalSocketAddress().getPort();
  return key;
  }else{
  key+= getRemoteSocketAddress().getPort();
  return key;
  }}
```

Code Program 1. WebsocketImpl Code

Next is to modify the DefaultExtension.java file. Modification is done on this files by adding encryption and decryption on definition decode and encode. Encryption process is added to encode method while decryption process is added to decode method. The encode method is used when websocket will send a message and decode method is used when websocket receives message from both server and client side. In implementing RC6 algorithm, it will be divided into functions based on their respective uses.

Encrypt function is used to arrange plaintext in array of bytes which has length n into plaintext blocks with array length 16. And if it turns out that plaintext length is not multiple of 16, then it will be padded so that it will eventually become a multiple of 16. Here is source code encrypt function:

```
  public static byte[] encrypt(byte[] data, byte[] key){
  if (data.length < 1 || key.length < 16) return new byte[0];
      byte[] bloc = new byte[16];
      S = generteSubkeys(key);
      int lenght = 16 - data.length % 16;
      byte[] padding = new byte[lenght];
      for (int i = 0; i < lenght; i++)
          padding[i] = 0;
        int count = 0;
      byte[]tmp= new byte[data.length+lenght];
          int i;
      for(i = 0; i < data.length+lenght; i++){
          if (i > 0 && i % 16 == 0) {
          bloc = encryptBloc(bloc);
     System.arraycopy(bloc, 0, tmp, i-16, bloc.length);}
          if (i < data.length) {
              bloc[i % 16] = data[i];}else{
              bloc[i % 16] = padding[count];
              count++;
              if(count > lenght + 1) count = 1;}}
      bloc = encryptBloc(bloc);
      System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);
      return tmp;}
```

Code Program 2. Encrypt Code

Decrypt function is used to compile ciphertext in array of bytes which has length n into ciphertext blocks with array length 16. At the end of the decrypt function process will call deletePadding function to delete padding that was previously done during encryption process. Here is the source code function decript.

```
  public static byte[] decrypt(byte[] data, byte[] key){
      if (data.length < 1 || key.length < 16) return new byte[0];
      byte[] tmp = new byte[data.length];
      byte[] bloc = new byte[16];
      S = generteSubkeys(key);
      int i;
      for(i = 0; i < data.length; i++){
          if (i > 0 && i % 16 == 0) {
              bloc = decryptBloc(bloc);
System.arraycopy(bloc, 0, tmp, i - 16, bloc.length);}
          if(i < data.length)
              bloc[i % 16] = data[i];}
      bloc = decryptBloc(bloc);
      System.arraycopy(bloc,     0,
tmp, i - 16, bloc.length);
      tmp = deletePadding(tmp);
      return tmp;}
```

Code Program 3. Decrypt Code

3.2 Testing Result

3.2.1 Speed Performance Test Result

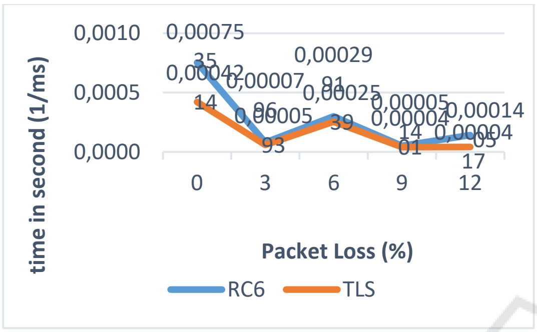

Figure 4. Speed performance test result

The results of time comparison test that have been obtained then analysed for each additional number of packet loss to time required in each message sending process both RC6 and TLS algorithm. From figure 4 shows that testing with a packet loss of 3% has decreased due to addition of packet loss which is bigger than the previous 0% so that packet delivery process becomes slower, because it requires repetition process of sending lost packages. Meanwhile, the 6% packet loss experienced an increase due to device condition which experienced an increase in speed performance and end of processes running on testing device or network traffic level which was not too dense at the test time. In testing with packet loss of 9% and 12% with the device and network conditions returning to normal, there was a decrease in performance due to the large number of packets lost.

The test results comparing of average messages delivery time (1/x) taken from client A (sender) to client B (destination) between RC6 and TLS algorithm displayed in graphical form as seen in Figure 4. The graph shows test results using 0%, 3%, 6%, 9% and 12% packet loss of each experiment so that comparison of message delivery times can be seen. From each experiment, average values (x) were 1327, 12569, 3343, 19441 and 7118 for RC6 and 2373, 16855, 3939, 24926 and 23961 for TLS. It can be concluded that RC6 algorithm has a better performance compared to TLS.

3.2.2 Security Test Result

The result of avalanche effect test on RC6 and TLS algorithms have been obtained by two test scenarios, based on plaintext and based on encryption key. In table 2 shows that avalanche effect test which is carried out by changing plaintext for RC6 algorithm, the average value is 28.29% and for TLS average value is amounted to 42.50%. Furthermore, the results of avalanche effect test with changes based on key, for RC6 algorithm obtained an average value of 52.28% and for TLS an average value was 53.19%. From these results it can be concluded that TLS is safer than RC6 algorithm.

Table 2. Analysis results calculation of avalanche effect RC6 and TLS algorithm

No	Changes	Algorithm	Testing			Avg
			1	2	3	
1	Plain text	RC6	25,20 %	28,61 %	31,95 %	28,29 %
2		TLS	37,57 %	38,59 %	51,36 %	42,50 %
3	Key	RC6	47,17 %	46,97 %	62,70 %	52,28 %
4		TLS	49,46 %	53,26 %	56,86 %	53,19 %

4 CONCLUSIONS

From the implementation and result that has been done, several conclusions can be drawn as follows:

1. The speed performance test results of sending messages from client A (sender) to client B (destination) is about average 8757 milliseconds for RC6 and 14411 milliseconds for TLS. From these results it can be concluded that RC6 algorithm has better performance compared to TLS.
2. The avalanche effect calculation results for RC6 algorithm obtained an average value of 28.29% based on plaintext and 52.28% based on keys. Meanwhile for TLS, average value was 42.50% based on plaintext and 53.19% based on keys. From these results it can be concluded that TLS is safer than RC6 algorithm.

REFERENCES

AGGARWAL, K. Comparison of RC6, Modified RC6 & Enhancement of RC6. 2015 International Conference on Advances in Computer Engineering and Applications,, 2015 Ghaziabad, India.

ARORA, S., MAINI, J., MALLICK, P., GOEL, P. & RASTOGI, R. Efficient E-learning management system through web socket. 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016 New Delhi, India.

BHARDWAJA, A., SUBRAHMANYAM, G., AVASTHI, V. & SASTRY, H. Security Algorithms for Cloud Computing. International Conference on Computational Modeling and Security (CMS 2016), 2016 Bengaluru, India.

DARMAWAN, I., RAHMATULLOH, A. & GUNAWAN, R. Real-time Screen Sharing Using Web Socket for Presenting Without Projector. 2019 7th International Conference on Information and Communication Technology (ICoICT), 2019 Kuala Lumpur, Malaysia, Malaysia.

DINESHA, A. & KB, E. B. Computations on Cipher Speech for Secure Biometrics. 6th International Conference On Advances In Computing & Communications, 2016 Cochin, India.

ENGHARDT, T., TIESEL, P. S., ZINNER, T. & A. FELDMANN. Informed Access Network Selection: The Benefitsof Socket Intents for Web Performance. 15th International Conference on Network and Service Management (CNSM), 2019 Halifax, NS, Canada, Canada, .

ERKKILÄ, J.-P. WebSocket Security Analysis. Seminar on Network Securi, 2012 Aalto University T-110.5291

JOSHI, B. 2012. HTML5 Programming for ASP.NET Developers. New York :apress.

KUBOV, V., DYMYTROV, Y. & KUBOVA, R. Wireless Devices HTML-interface for Medical Applications. 8th Mediterranean Conference on Embedded Computing (MECO), 2019 Budva, Montenegro, Monten.

LIU, N., CAI, J., ZENG, X., LIN, G. & CHEN, J. Cryptographic Performance for Rijndael and RC6Block Ciphers. 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID), 2017 Xiamen, China.

MAHIDHAR, R. & RAUT, A. A Survey On Scheduling Schemes With Security In Wireless Sensor Networks,. International Conference on Information Security & Privacy (ICISP2015), 2016 Nagpur, INDIA.

MISHRA, P. & J, J. 2013. Application Research and Penetration Testing on WebSocket Technology. *International Journal of Science and Research (IJSR),* , 4.

NAEEM, E. A., ELNABY, M. M. A., EL-SAYED, H., EL-SAMIE, F. E. A. & FARAGALLAH, O. S. 2016. Wavelet fusion foren crypting image swith a few details. *Computers and Electrical Engineering,* 54**,** 450-470.

NAMASUDRA, S. & GANDOMI, G. C. D. P. J. M. H. A. H. 2020. The Revolution of Blockchain: State of the Art and Research Challenges. *Archives of Computational Methods in Engineering*.

PARK, J.-T., H.-S. HWANG, YUN, J.-S. & MOON, I.-Y. 2014. Study of HTML5 WebSocket for a Multimedia Communication. *International Journal of Multimedia and Ubiquitous Engineering,* 9**,** 61-72.

PIMENTEL, V. & NICKERSON, B. G. 2012. Communicating and Displaying Real-Time Data with WebSocket. *IEEE Computer Society,* 16**,** 340-361.

RIVEST, R. L., M.J.B. ROBSHAW, R.SIDNEY & YIN, Y. L. 1998. The RC6 Block Cipher.

SOEWITOA, B., CHRISTIANA, GUNAWAN, F. E., DIANAA & KUSUMA, I. G. P. Websocket to Support Real Time Smart Home Applications. 4th International Conference on Computer Science and Computational Intelligence 2019(ICCSCI), 2019 Yogyakarta, Indonesia.

TSAI, H., CHANG, C., HOU, X., YONG, Y., CHIOU, K. & YU, P. 2019. Interactive student response system with iBeacon and web socket for flipped classroom learning. *Journal of Computing in Higher Education,* 31**,** 3430-361.

WEIZHEN, S., YUTONG, F. & LIMING, L. 2012. A Hybrid Design and Implementation of Socket Directly and Web Browser Online Email Address Verifier. *Affective Computing and Intelligent Interaction,,* 137**,** 11-19.