Low Cost Dynamic Random Nonce Generator For Web Services

Uma Elangovan, Kannan Arputharaj, and Ramesh Ramadoss

Abstract**—**The random numbers are used in all areas of computer based systems. In general, random numbers are spawned from big seed values that is retrieved from complex formulas. The random number algorithm is proposed to generate the random numbers in low cost and efficient manner. The seed values are collected from parameters of clients and server for example mouse movement values and registry values of the system. The generated random numbers are tested against brute force attack testing tool. Also, it is compared with existing random generating algorithms and it is found the proposed algorithm is very well suitable for system with lowest configuration.

Keywords — Random number, nonce, seed, system parameters.

I. INTRODUCTION

WEB Services are "document oriented" rather than object oriented. This means that Web Services are described by simple XML documents. To address the authentication, WS-Security suggests including a username token in the SOAP header which contains a username, hash of the password, timestamp and nonce. The username will be how the user is registered with the server and sent as digested values. WS-Security requires the password to be hashed using the SHA-1 algorithm. Since SOAP messages are sent over networks in plain text, the password must be hashed with random number and timestamp values as nonce to prevent malicious attackers from discovering passwords and gaining access. A timestamp tells the server when a message was sent, and the message times out if the timestamp is too old. A nonce is a random, unique string generated by the user and the server. The combination of the timestamp and nonce prevent replay attacks by malicious attackers.

To acquire the message encryption in WS-Security the user credentials, an improved authentication scheme is proposed with a dynamic nonce generator. The dynamic nonce generator gets system parameters, user's mouse movements as entropy then generates the nonce value and then handed over to the authentication module.

Uma Elangovan , Department of Information Science and Technology, Anna University, India. (email id: euma@annauniv.edu)

Kannan Arputharaj, Department of Information Science and Technology, Anna University, India. (email id: kannan@annauniv.edu)

Ramesh Ramadoss, Department of Electrical and Electronics Engineering, Anna University, India. (email id: rramesh@annauniv.edu)

II. LITERATURE SURVEY

Recently, researchers are enthusiastically involved in finding and preventing attacks targeted at Web Services by proposing detection and mitigation techniques. Therefore, a number of research works have been published in the literature by various researchers. Gruschka et al (2011) introduced a comprehensive stream-based WS-security processing system that gives a more efficient processing in service computing and improves the robustness against different types of Denial-Of-Service (DoS) attacks. Their engine is capable of processing all standard applications of WS-Security in a streaming approach. Their system was designed to handle, e.g., any arrangement, number, and nesting degree of signature and encryption mechanisms, closing the gap toward more efficient and dependable Web Services. Hoquea et al (2013) studied the behavior and possible impact or severity of damages. Then, the author categorized the attacks into a number of distinct classes. They provided taxonomy of attack tools in a consistent way for the benefit of network security researchers. They presented a wide-ranging and planned survey of existing tools and systems that can support both attackers and network defenders. The authors have provided a discussion on the merits and demerits of such tools and systems for better understanding of their capabilities.

Kim et al (2005) have proposed the use of nonce added with time stamp based scheme. In their model, each scheme has to be combined to with stand time stamp and nonce usage in cryptanalytic attack. Their Nonce based scheme needs time stamp to avoid off-line password guessing attack. The time stamp based scheme needs Nonce values to thwart the attacker from brute force attack.

Wright et al (2010) evaluated the performance of a popular open-source SIP server on multicore architectures. They introduced three analysis-driven optimizations that involve increasing the number of slots in hash tables, an in-memory database for user authentication information and incremental garbage collection for user location information. The lock contention had reduced the search time of wider hash tables and showed significant improvement in multicore scalability.Jensen et al (2011) analyzed the effectiveness of the specific countermeasure of XML Schema validation in terms of fending Signature Wrapping attacks. They investigated the problems of XML Schema validation for Web Services messages, and discussed the approach of Schema Hardening, a technique for strengthening XML Schema declarations. They examined the structure of attack and solved through schema hardening algorithm.

http://dx.doi.org/10.15242/IIE.E0514517

Shahgholi et al (2011) defended main XML threats, especially WSDL attacks in an SOA environment. These Web Services threats are WSDL attacks containing WSDL scanning, parameter tampering, oversize payloads and recursive payloads attacks.

Tsang et al (2011) presented Nymble, a system in which servers can "blacklist" misbehaving users, thereby blocking users without compromising their anonymity. Nymble construction has Blacklistability, Rate-limiting, Nonframability and Anonymity.

Charles Shen et al (2012) studied the performance impact of using Transport Layer Security (TLS) as a transport protocol for Session Initiation Protocol servers. The cost of TLS was evaluated experimentally using a testbed with OpenSIPS, OpenSSL and Linux running on an Intel-based server. Finally, the impact of using TLS as a transport on SIP server performance versus the standard approach of SIP-over-UDP was evaluated and analyzed.

Le et al (2012) presented Double Guard, an Intrusion Detection System that imitates the network behavior of user sessions across both the front-end Webserver and the back-end database. They examined both Web and subsequent database requests; it is able to hunt out attacks that an independent IDS would not be able to identify. Moreover, the limitations of any multitier IDS are quantified in terms of training sessions and functionality coverage.

Chang & Lee (2012) handled single sign-on mechanism for multipurpose of different Web sites. To achieve that they assigned unitary token and time stamp information to the user realms. Similar approach proposed by Xu et al (2012) & McGibneya (2005) for dynamic authentication.

Rosa et al (2013) described an XML injection strategy-based detection system, to mitigate the time gap for 0-day attacks that is previously unknown attack and resulting from an ontology's attack variations. Because many new and unknown attacks are derived from known strategies. The authors presented XID as a hybrid approach that supports knowledge-based detection derived from a signature- based approach using ontology for Web Services.

All existing approaches use extensive level of computing approaches to create random numbers. To overcome the limitations of these existing systems, a self-repetitive method is proposed in this paper.

III. Architecture Of Dynamic Nonce Generator

A.Design of Dynamic Nonce Generator

The nonces are generated from chaotic of computer parameters and from user's activities. Figure 1 provides an architectural model of the proposed Dynamic Nonce Generator (DNG). The DNG includes a source of chaotic input and includes a nonce source. The components of this model are discussed in the following subsections.

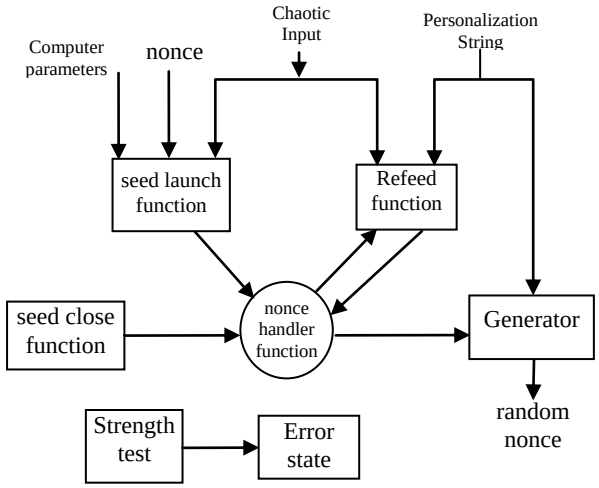

Fig. 1 Architecture of dynamic nonce generator

B. Chaotic input

The chaotic input is provided to a dynamic nonce generator mechanism for the seed. The chaotic input and the seed must be kept secret. The confidentiality of this information provides the basis for the security of the generator. The chaotic input can be defined to be a changeable length, as well as prefixed length.

C. Other inputs

Other data collected by a DNG mechanism as input. This information must kept secret by a consuming application; however, the security of the RBG itself does not rely on the secrecy of this information. The information should be checked for validity when possible; for example, if time stamp is used as an input, the format of the time is checked. During DNG initiation, a nonce is required and it is combined with the chaotic input to create the initial DNG seed. This filter inserts the personalization string during DNG initiation. At that time, the personalization string is combined with the chaotic input bits and a nonce is created for the initial DNG seed. The personalization string should be unique for all initiations of the same DNG mechanism (e.g., Task identifier). Additional input also be provided during refeeding and when pseudo random bits are requested.

D. Nonce handler function

The nonce handler function maintains the status of nonce and stores parameters, variables and other stored values that the DNG mechanism uses.

E. Functions of the Dynamic Nonce Generator Mechanism

The DNG mechanism functions handle the dynamic nonce. The DNG mechanisms have five separate functions:

- The seed launch function acquires chaotic input and may combine it with a nonce and a personalization string to create a seed from which the initial internal state is created.
- The generator function generates pseudorandom bits upon request, using the current nonce and generates a new nonce for the next request.
- The refeed function acquires new chaotic input and combines it with the current nonce and any additional input that is provided to create a new seed and a new nonce.

- The seed close function erases the nonce.
- The strength test function determines that the dynamic nonce mechanism continues to function correctly.

A dynamic nonce generator mechanism requires initiate, uninitiate, generate, and strength testing functions. A DNG mechanism includes a refeed function. A DNG must be initiated prior to the generation of output by the DNG.

F. DNG initiations

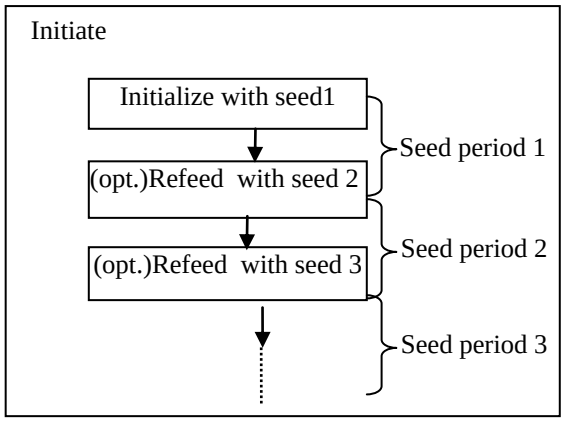

Fig. 2 DNG Initiation

A DNG is initiated using a seed and must be refed. Each seed defines a seed period for the DNG initiation; an initiation consists of one or more seed periods that begin when a new seed is acquired as shown in Figure 2.

G. Nonce handler function

During seed launch, an initial nonce is derived from the seed. The initial nonce for seed launch includes:

a) Working nonce
   - Some values are derived from the seed and become part of the initial nonce. This value is kept secret.
   - A count of the number of requests or blocks produced since the initiation was seeded or refed.

b) Administrative information (e.g., security strength and prediction resistance flag).

The initial nonce is protected when it is requested for the use of the pseudorandom output bits requested by the consuming application. The actual security strength supported by a given initiation depends on the DNG implementation and on the amount of chaotic provided to the initiate function. The generate function checks that the requested security strength does not exceed the security strength for the initiation. Assuming that the request is valid, the requested number of bits is returned. The DNG needs to be initiated for the highest security strength required. Figure 6.4 shows the distributed DNG mechanism.

H. Boundaries of DNG mechanism

A DNG mechanism boundary contains all DNG mechanism functions and status of nonce required for a DNG. Within a DNG mechanism boundary,

- The status of the nonce and the operation impacts the function according to the specification of DNG mechanism.
- The DNG status exists exclusively within the DNG mechanism boundary. The status maintained within the DNG mechanism boundary.
- Information about secret parts of the DNG core state function and intermediate values in computations involving these secret parts will not affect any information that leaves the DNG mechanism boundary.

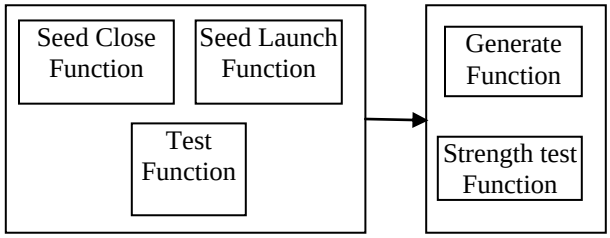

Fig. 3 Distributed DNG mechanism functions

Each DNG mechanism boundary or sub-boundary must pass the robustness test function to test the health of the other DNG mechanism functions within that boundary as shown in Figure 3. In addition, each boundary or sub-boundary contains an uninitiate function in order to perform and/or react to health testing.

H. Seeds

When the DNG is used to generate pseudorandom bits, a seed is acquired prior to the generation of output bits by the nonce generator. The seed is used to initiate the nonce generator and determine the initial internal state. Refeeding is a means of restoring the confidentiality of the output of the nonce generator if a seed or the internal state becomes known.

I. Seed construction for initiation

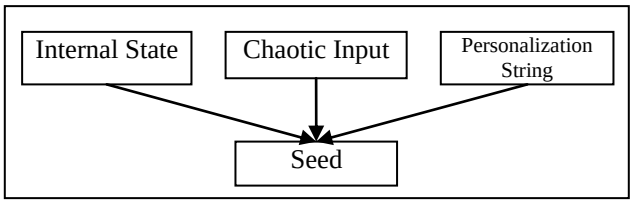

Fig. 4 Seed construction process for initiation

Figure 4 depicts the seed construction process for initiation. The seed material used to determine a seed for initiation consists of chaotic input, a nonce and an optional personalization string. Chaotic input is always be used in the construction of a seed. Depending on the DNG mechanism and the source of the chaotic input, a derivation function is required to derive a seed from the seed material.

Seed Construction for refeeding

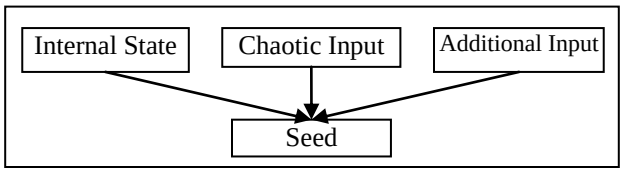

Fig. 5 Seed construction for refeeding

Figure 5 depicts the seed construction process for refeeding an initiation. The seed material for refeeding consists of a value that is carried in the internal state, new chaotic input and optionally, additional input. The internal state value and the chaotic input are required.

Chaotic Requirements for the Construction of Chaotic

The chaotic input acquires input from the chaotic input that is equal to or greater than the security strength of the initiation. Additional chaotic is provided in the nonce or the optional personalization string during initiation, or in the additional input during refeeding and generation, but this is not required.

Source of chaotic input

The source of the chaotic input is either from an approved DNG or an Approved DNG, thus forming a chain of at least two DNGs. The highest level DNG in the chain is seeded by an approved DNG or a chaotic source.

Chaotic input and seed confidentiality

The chaotic input and the resulting seed is handled in a way that is consistent with the security required for the data protected by the consuming application.

Nonce

A nonce is required in the construction of a seed during initiation in order to provide a security to block certain attacks. The nonce can be either: An unpredictable value with at least (1/2 security_strength) bits of chaotic or a value that is expected to repeat no more often than a (1/2 security_strength)-bit random string would be expected to repeat.

Reinitiating

Generating too many outputs from a seed provide sufficient information for successfully predicting future outputs. Periodic regenerating will reduce security risks. Seeds have a finite seed life (i.e., the number of blocks or outputs that are produced during a seed period); the maximum seedlife is dependent on the DNG mechanism used. Reinitiating is accomplished by 1) an explicit reinitiating of the DNG by the consuming application, or 2) by the generate function when prediction resistance is or the limit of the seedlife is reached.

Personalization String

During initiation, a personalization string is used to derive the seed. The intent of a personalization string is to differentiate this DNG initiation from all other created initiations. The personalization string should be set to some bitstring that is as unique as possible and includes secret information. Secret information is not used in the personalization string if it requires a level of protection that is greater than the intended security strength of the DNG initiation. The sources of the personalization string contents include:

- Registry identifier of operating system
- User identification
- Private keys
- PINs and passwords
- Timestamps
- Application identifiers
- User activities like mouse movement
- Random numbers
-

Additional Input

During each request for bits from a DNG and during reinitiating, the insertion of additional input is allowed. This input is optional, and the ability to enter additional input may or may not be included in an implementation. Additional input may be either secret or publicly known; its value is arbitrary, although its length may be restricted, depending on the implementation and the DNG mechanism.

Algorithm for Dynamic Nonce Generation

The Nonce has generated dynamically from the mouse movement of client system. The system calculates the different mouse movements and stores it into database for each session of communication. It collects fresh Nonce for all session. The generator captures mouse movements using an onmousemove event to get the values of window.event.clientX and window.event.clientY and feeds seed values into a random nonce generator, taking the first 256 bits for reasons of keeping a random pool unknown.

During initiation and refeeding, chaotic input is acquired for constructing a seed. To achieve that, a Get_chaotic_input function is used. The chaotic input is not provided by a consuming application as an input parameter in an initiate or refeed request. The Get_chaotic_input function has the following meaning: Get_chaotic_input: A function that is used to obtain chaotic input. The function call is: (status, chaotic_input) = Get_chaotic_input (min_chaotic, min_ length, max_ length), which requests a string of bits (chaotic_input) with at least min_chaotic bits of chaotic. The length for the string is equal to or greater than min_length bits and less than or equal to max_length bits. A status code is also returned from the function.

Initiating the DNG

A DNG is initiated prior to the generation of dynamic random bits. The initiate function . Checks the validity of the input parameters, determines the security strength for the DNG initiation and DNG mechanism specific parameters, obtains chaotic input with chaotic sufficient to support the security strength and the nonce. This also, determines the initial internal state using the initiate algorithm.

The following algorithm is used to initiate a DNG. Initiate_function(requested_initiation_security_strength,prediction_resistance_flag, personalization_string):

- Requested_initiation_security_strength: A requested security strength for the initiation. The maximum security strength is controlled in this parameter.
- Prediction_resistance_flag: Indicates whether or not prediction resistance is required by the consuming
- Personalization_string: An optional input that provides personalization information. The maximum length of the personalization string (max_personalization_string_length) is defined by this function, but less than or equal to the maximum length specified for the given DNG mechanism. If the input of a personalization string is not supported, then the personalization_string input parameter and step 3 of the initiate process are omitted.

Information retained within the DNG mechanism boundary after initiation: The internal state for the DNG, including the working_state and administrative information.

Initiate Process:

// Check the validity of the input parameters.

- If requested_initiation_security_strength > highest_ supported_security_strength, then return an exception.
- If prediction_resistance_flag is set, and prediction resistance is not supported, then return an exception.
- If the length of the personalization_string > max_personalization_string_length, return an exception.
- Set security_strength to the nearest security strength greater than or equal to requested_initiation_security_strength.
- Using security_strength, select appropriate DNG mechanism parameters.

// Obtain the chaotic input.

- (status, chaotic_input) = Get_chaotic_input (security_strength, min_length, max_length).
- If an error is returned in step 6, return a catastrophic error.
- Obtain a nonce.

// This step is included on the acceptability of the nonce. Call the appropriate initiate algorithm to obtain values for the initial working_state.

- initial_working_state = Initiate_algorithm (chaotic_input, nonce, personalization_string).
- Get a state_handle for a currently empty internal state. If an unused internal state cannot be found, return an error.
- Set the internal state indicated by state_handle to the initial values for the internal state and any other values required for the working_state, and set the administrative information to the appropriate values (e.g., the values of security_strength and the prediction_resistance).
- Return success and state_handle.

Refeeding a DNG initiation

The refeeding of an initiation is not required, but is used whenever a consuming application and implementation are able to perform this process. Refeeding will insert additional chaotic into the generation of pseudorandom bits. Refeeding is explicitly requested by a consuming application, performed when prediction resistance is requested by a consuming application, triggered by the generate function when a predetermined number of pseudorandom outputs have been produced or a predetermined number of generate requests have been made (i.e., at the end of the seedlife) or triggered by external events (e.g., whenever sufficient chaotic is available).

The working_state is the working state for the particular DNG initiation, min_length and max_ length is defined for each DNG mechanism and Refeed_algorithm is called to the appropriate refeed algorithm for the DNG mechanism.
The following process is refeed the DNG initiation.

Refeed_function **(**state_handle, additional_input):

- state_handle: A pointer or index that indicates the internal state to be refed.
- additional_input: An optional input. The maximum length of the additional_input(max_additional_input_length) is less than or equal to the maximum value specified for the given DNG mechanism. If the input by a consuming application of additional_input is not supported, then the input parameter and step 2 of the refeed process are omitted and step 5 of the refeed process is modified to remove the additional_input from the parameter list.

Information retained within the DNG mechanism boundary after refeeding:

Replaced internal state values (i.e., the working_state).

Refeed Process

// Get the current internal state and check the input parameters.

- Using state_handle, obtain the current internal state. If state_handle indicates an invalid or unused internal state, return an exception.
- If the length of the additional_input > max_additional_input_length, return an exception.

//Obtain the chaotic input.

- (status, chaotic_input) = Get_chaotic_input (security_strength, min_length, max_length).
- If an error is returned in step 3, return a catastrophic exception.

//Get the new working_state using the appropriate refeed algorithm.

- new_working_state = Refeed_algorithm (working_state, chaotic_input, additional_input).
- Replace the working_state in the internal state indicated by state_handle with the values of new_working_state obtained in step 5.
- Return success.

Generating dynamic nonce using a DNG

This function is used to generate dynamic random bits after initiation or refeeding. The generate function checks the validity of the input parameters. Then it calls the refeed function to obtain sufficient chaotic if the initiation needs additional chaotic because the end of the seedlife has been reached or prediction resistance is required. Afterwards, generates the requested pseudorandom bits using the generate algorithm. Then it updates the working state. Finally, it returns the requested pseudorandom bits to the consuming application.

The Generate Function

The generate function is used to generate dynamic random bits. Generate_function: **(**state_handle, requested_number_of_ bits, requested_ security_strength**,**prediction_resistance_request, additional_input):

- state_handle: A pointer or index that indicates the internal state to be used.
- requested_number_of_bits: The number of dynamic random bits to be returned from the generate function.

- requested_security_strength: The security strength to be associated with the requested pseudorandom bits.
- prediction_resistance_request: Indicates whether or not prediction resistance is to be provided during the request. The function prediction resistance is performed then it calls refeed function. status = Refeed_function (state_handle, additional_input). If status indicates an error, then return status.
- additional_input: An optional input. The maximum length of the additional_input (max_additional_input_length) is less than or equal to the specified maximum length for the selected DNG mechanism. If the input of additional_input is not supported, then the input parameter, generate process steps 4 and the additional_input input.

Output to the consuming application after generation:

- status: the status returned from the generate function. the status will indicate success or an error.
- Dynamicrandom_bits: The dynamic random bits that were requested.

Information retained within the DNG mechanism boundary after generation:

Replaced internal state values (i.e., the new working_state).

Generate Process:

// Get the internal state and check the input parameters.

- Using state_handle, obtain the current internal state for the initiation. If state_handle indicates an invalid or unused internal state, then return an exception.
- If requested_number_of_bits > max_number_ of_bits_ per_ request, then return an exception
- If requested_security_strength > the security_strength indicated in the internal state, then return an exception.
- If the length of the additional_input > max_additional_input_length, then return an exception.
- If prediction_resistance_request is set, and prediction_ resistance_flag is not set, then return an exception.
- Clear the refeed_required_flag.
- If refeed_required_flag is set, or if prediction_ resistance_request is set, then
  - status = Refeed_function (state_handle, additional_ input).
  - If status indicates an ERROR, then return status.
  - Using state_handle, obtain the new internal state.
  - additional_input = the Null string.
  - Clear the refeed_required_flag.

    // Request the generation of pseudorandom_bits using the appropriate generate algorithm.
  - (status, pseudorandom_bits, new_working_state) = Generate_algorithm (working_state, requested_number_of_bits, additional_input).
  - If status indicates that a refeed is required before the requested bits can be generated, then
    - Set the refeed_required_flag.
    - Go to step 7.
  - Replace the old working_state in the internal state indicated by state_handle with the values of new_working_state.
  - Return success and dynamic andom_bits.

Removing a DNG initiation

The internal state for an initiation is need to be released by erasing (i.e., nullifing) the contents of the internal state. The uninitiate function:

- Checks the input parameter for validity.
- Empties the internal state.

The following is used to remove a DNG initiation

Uninitiate_function **(**state_handle**)**

state_handle: A pointer indicates the internal state to be released. If a state handle is not used by an implementation because the implemention does not support multiple simultaneous initiations, a state_handle is not provided as input. In process step 1 is omitted and process step 2 erases the internal state.

Output to a consuming application after uninitiation

status: The status returned from the function. The status will indicate success or exception. The information retained within the DNG mechanism boundary after uninitiation: An empty internal state.

Uninitiate Process

- If state_handle indicates an invalid state, then return an exception.
- Erase the contents of the internal state indicated by state_handle.
- Return success

Then dynamic random nonce is now accessed by the authentication approach

IV. Implementation

The secured system is implemented with C Sharp and WSE 3.0 in .net environment.

```
<wsse:UsernameToken>
<wsse:Username>admin</wsse:Username>
            <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-
1.0#PasswordDigest">RKFQi+jYBQpXRGnHogrRzpcmpVM=</wsse:
Password>

<wsse:Nonce>tdusE7RWcPV19ZrQqWJEJg==</wsse:Nonce>
            <wsu:Created>2012-08-
21T11:45:32Z</wsu:Created>
<wsse:Username>admin</wsse:Username>
            <wsse:Password
Type="wsse:PasswordDigest">
184D2A12DRG8D9F0C6HH82C89B091EG5C8A872F94DC188</wsse:Pa
ssword>
<wsse:Password>
lc/H00ZNNjlPbCHD4jjBH0E2vLM=</wsse:Password>

<wsse:Nonce>xj2b0r4U8maf4bvAexyFiw==</wsse:Nonce>
            <wsu:Created>2012-08-
17T09:42:59Z</wsu:Created>
 </wsse:UsernameToken>
```

Fig. 6 Hashed password and dynamic nonce for authentication schemes

The generated nonce is added in token value of the Web service. The <wsse:UsernameToken> element provides a countermeasure for replay attacks: <wsse:Nonce> and <wsu:Created> is shown in Figure 6. A dynamic nonce is a random value which is created by sender to include in each UsernameToken that it sends. Although using a nonce is an effective countermeasure against replay attacks, it requires a server to maintain a cache of used nonces and consumes the server resources. Combining a nonce with a created timestamp has the advantage of allowing a server to limit the cache of nonces to a "freshness" time period, establishing an upper bound on resource requirements.

A first approach to prevent this could be to specify a timeout value for the token, thus a request with an expired timestamp will not be accepted by the server. If the sender sets a timestamp of 60 seconds and the server receives the message later then 60 seconds after the given <Created>value, it simply rejects the whole request. This is easy to implement, but can also have some problems, like expired messages being accepted due to clock synchronization issues on the server.

```
<wsu:Timestamp wsu:Id="Timestamp-9267154b-9711-409d-80c5-
fb331f541ed8">
            <wsu:Created>2012-08-
17T09:42:59Z</wsu:Created>
            <wsu:Expires>2012-08-
17T09:47:59Z</wsu:Expires>
</wsu:Timestamp>
<wsu:MouseMove>GosOH6PPldyG3VAaaeCCcQ==</wsu:MouseMove>
```

Fig.7 Dynamic random number generation with time stamp and mouse movement

Regarding time synchronization issues, WS-Security provides the <Timestamp> header and for it uses <MouseMove> headers for random number generation. These can be very useful for message creation, receipt and processing. The schema outline for the <Timestamp> and <wsu:MouseMove> element has displayed in Figure 7

V. Conclusion

The proposed algorithm for random number generation produces unpredictable nonce values for the security purpose. It is designed and implemented for low configured devices. Also it is suitable for all types of applications. This approach is tested with various seed values with different chaotic inputs. Further, it is evaluated with various random generating algorithms.

VI. Acknowldegment

The researchers would like to acknowledge the financial support provided by University Grant Commission, Govt of India, New Delhi. (UGC F.No. 41-1363/2012(SR) and Centre for International Affairs , Anna University , Chennai.

VII. References

[1] Gruschka, N, Jensen, M, Lo Iacono, L & Luttenberger, N 2011, 'Server-side Streaming Processing of WS-Security', IEEE Transactions on Services Computing, vol. 4, no. 4, pp. 272-285. http://dx.doi.org/10.1109/TSC.2010.61

[2] Hoque, N, Monowar, H, Bhuyan, Baishya, RC, Bhattacharyya, DK & Kalita, JK 2013, 'Network Attacks: Taxonomy, tools and systems', Journal of Computer and Network Applications (Accepted), Available from: <http://dx.doi.org/10.1016/j.jnca.2013.08.001>.4 October 2013.

[3] Kim, KW, Jeon, JC, & Yoo, KY 2005, 'An Improvement on Yang et al.'s Password Authentication Schemes', Applied Mathematics and Computation, vol. 170, no.1, pp. 207-215. http://dx.doi.org/10.1016/j.amc.2004.11.040

[4] Wright, CP, Nahum, EM, Wood, D, Tracey, JM & Hu, EC 2010, 'SIP Server Performance on Multicore Systems', IBM Journal of Research and Development, vol. 54, no. 1, pp. 71-84. http://dx.doi.org/10.1147/JRD.2009.2036976

[5] Jensen, M, Meyer, C, Somorovsky, J & Schwenk, J 2011, 'On the Effectiveness of XML Schema Validation for Countering XML Signature Wrapping Attacks', IEEE Proceedings of Securing Services On The Cloud (IWSSC), pp 7-13.

[6] Shahgholi, N, Mohsenzadeh, M, Seyyedi, MA & Qorani, SH 2011, 'A New Security Framework Against Web Service's XML Attacks in SOA', Proceedings of the Seventh International Conference on Next Generation Web Services Practices (NWeSP), pp. 314-319.

[7] Tsang, PP, Kapadia, A, Cornelius, C & Smith 2011, 'Nymble: Blocking Misbehaving Users in Anonymizing Networks', IEEE Transactions on Dependable and Secure Computing, vol. 8, no. 2, pp. 256-269.

[8] Charles Shen, Erich Nahum, Henning Schulzrinne & Charles P. Wright 2012, 'The Impact of TLS on SIP Server Performance: Measurement and Modeling', IEEE/ACM Transactions on Networking (TON), vol. 20, no. 4, pp. 1217- 1230.

[9] Le, M., Stavrou, A & Kang, BB 2012, 'DoubleGuard: Detecting Intrusions in Multitier Web Applications', IEEE Transactions on Dependable and Secure Computing, vol. 9, no. 4, pp. 512-525. http://dx.doi.org/10.1109/TDSC.2011.59

[10] Chang, CC & Lee, CY 2012, 'A Secure Single Sign-on Mechanism for Distributed Computer Networks', IEEE Transaction on Industrial Electronics, vol.59, no.1, pp. 629-637. http://dx.doi.org/10.1109/TIE.2011.2130500

[11] Rosa, TM , Santin, AO & Malucelli, A 2013, 'Mitigating XML Injection 0-Day Attacks through Strategy-Based Detection Systems', IEEE Security & Privacy, vol. 11 , no. 4, pp. 46-53. http://dx.doi.org/10.1109/MSP.2012.83