CHAPTER 6

A DYNAMIC NONCE BASED AUTHENTICATION SCHEME TO DEFEND INTERMEDIARY ATTACKS ON WEB SERVICES

6.1 INTRODUCTION

Authentication is an act of ensuring the truth of an attribute of a datum or entity (Xu et al 2012). This could involve confirming the identity of a person or software program. This is the process of identifying an individual using the credentials of that individual. For example, a bank teller is required to authenticate the person by examining his social security number. Authentication typically occurs immediately after identification.

This enforces the verification and the validation of the identities and credentials exchanged between the Web Services provider and the consumer. It ensures that the consumer accessing the information is the correct person or not. The initiating service requester must be authenticated to prove its identity with reliable credentials.

Web Services are thought of to be a means to provide easily accessible services over a network. They can be simply usable regardless of the underlying network structure, configuration, operating system, communication mechanism or implementing language. While there are different possibilities to communicate with a Web Services, SOAP is regarded to be the de-facto standard. SOAP messages are being sent to service endpoints identified by URIs, commonly with the endpoint's service
processing some action and sending a SOAP response containing results or error codes.

To be successful in business concerns, Web Services are suitable for secure communication. So far the original SOAP specification contains no solutions to solve the security problem. Other techniques as Secure Socket Layer (SSL) or Internet Protocol Security (IPSec) provide standard transport level security. The problem is the SOAP connection from one endpoint to another can be seen as a logical connection, abstracting from the physical infrastructure beyond. Logically being an end-to-end connection, the physical layer can comprise of diverse intermediaries forwarding SOAP messages. During this process of receiving and forwarding messages, security information defined on transport level, the way i.e., SSL works, can easily get lost. Thus any recipient had to rely on the security handling of his physical connection point predecessor, as well as its handling of the data integrity and confidentiality. To overcome this problem the transferred data must specify security information on message level.

Web Services are invoked by sending Simple Object Access Protocol (SOAP) messages over the internet. SOAP messages are XML crafted documents. The client sends a SOAP request to the server and then the Web Service performs some task and sends a SOAP response back to the client. A SOAP message consists of a header and a body. The header holds metadata about the message. The body contains the actual message, in a request sent to the server, the body specifies a Web Service to be invoked and parameters and return variables required to call a Web Service, in the response from the server the SOAP body includes the variables returned by the Web Service.
Web Service Description Language (WSDL) is an extension of XML that is used to define a Web Service. Every function, parameter and return type made accessible by the Web Service is defined in the WSDL file. Using WSDL allows Web Services to be defined while still being platform independent. WSDL are a key element in the UDDI Registry. A business can publish the WSDL file for their Web Services on the UDDI Registry and then a user can discover their Web Services and create client application to access the Web Services. Since the Web Services are described by platform independent WSDL files, the client application can be written in any language and communicate with the Web Service flawlessly. The ease of discovery and platform flexibility make Web Services a very powerful tool, and have led to broad usage in all types of applications where two modules written in different languages need to communicate over the internet to accomplish a task.

The XML Web Services require client authentication to prevent attacks. The system needs to share data of user information such as username, password in encrypted format which was implemented using WS-Security framework to provide message level security. WS-Security protects message contents, while transport service intermediaries and gives authentication and authorization control, which protects service provider from malicious requesters (Christian 2005). WS-Security does not define any authentication ticket mechanism; instead, it defines how to use plain user name/password, Kerberos and X.509 tickets within the context of a SOAP header.

WS-Trust, WS-SecureConversation and WS-Federation define the protocols that help establish agreements between requester and provider about the kinds of security in use. WS-SecurityPolicy is used to declare a provider’s requirements for security like strong authentication (Cremonini 2003).
WS-Security header can be added to SOAP messages before sending to that service provider. The header should include authentication, authorization, encryption and signature. The provider can validate the credentials of the requester before executing the service. Invalid credentials, typically result in the return of an error message to the requester. Invalid credentials, will be returned to the client by an error message.

A WS-Security Username Token enables the end-user identity to be passed over multiple hops before reaching the destination Web Service. The user identity is inserted into the message and is available for processing at each hop on its path. The client user name and password are encapsulated in a WS-Security <wsse:UsernameToken>. When the Enterprise Gateway receives this token, it performs the following tasks, depending on the requirements as shown in Figure 6.1.

- Ensure that the timestamp on the token is still valid
- Authenticate the user name against a repository
- Authenticate the user name and password against a repository

```xml
<wsse:UsernameToken>
   <wsse:Username>...</wsse:Username>
   <wsse:Password>....</wsse:Password>
   <wsse:Nonce>.... wsse:Nonce>
   <wsu:Created>...</wsu:Created>
</wsse:UsernameToken>
<wsu:Timestamp wsu:Id="Timestamp-9267154b-9711-409d-80c5-fb331f541ed8">
   <wsu:Created>...</wsu:Created>
   <wsu:Expires>...</wsu:Expires>
</wsu:Timestamp>
```

Figure 6.1 WS-security specification for authentication
The UsernameToken profile specifies how the UsernameToken can be used as a means to identify a requester by username. A password, or some sort of a shared secret constituting a password equivalent, may also be included. Passwords may be included in their original form or as a SHA-1 digest. In order to prevent replay attacks, it is also recommended that a nonce (i.e., a random value created by the sender) and a timestamp are included. By combining nonces with timestamps, nonces are not required to be cached beyond their validity period. The SHA-1 password digest is to be calculated over the nonce, timestamp and password, thus, both the sender and the receiver need to know the plaintext password or password equivalent. Notice though, that if the password equivalent is the digest of the password, the receiver is not required to store the plaintext password.

The UsernameToken profile also defines a way to derive a shared cryptographic key from the password associated with a given username. Key derivation is achieved by specifying a salt (i.e., a random value) and a number of iterations. By hashing the password and nonce, and iterating the number of times specified on the result, the shared key can be obtained. This way, only those who know the passwords are able to derive the shared key (given the nonce and the number of iterations). The maximum size of a derived key is 160 bits, although the chaotic of keys generated from typical passwords in this way is likely to be much lower. For instance, there are only \((26+26+10)^8\), or about 248, different eight character passwords consisting of lower- and uppercase letters as well as numbers. Thus, a key derived from such a password would be comparable in strength with a 48-bit key, which is not secure for general use. Additionally, such a scheme may be vulnerable to dictionary attacks unless the passwords have been randomly generated. The specification does not provide measures to prevent a UsernameToken from being replayed to a different receiver. Thus, if the same usernames/passwords are valid with multiple receivers, measures against such replay attacks must
be provided by implementers. One potential solution is to require the identity of the receiver to be included in the password digest. Nevertheless, such custom solutions may be vulnerable to cause interoperability problems.

The WS-Security specification, describes how to secure Web Services at the message level, rather than at the transport protocol level. Existing transport-level solutions such as SSL/TLS provide solid point-to-point data encryption and authentication but have limitations if a message needs to be processed or examined by an intermediate service.

To overcome said limitations, the proposed system has been implemented to provide random values for any type of authentication approach.

6.2 ARCHITECTURE OF DYNAMIC NONCE GENERATOR

A Web Service is a set of protocols to allow communication between two applications. A client and a server can pass messages back and forth to exchange data over the internet similar to how two processes communicate on a single computer. There are several ways to pass these messages, but the most common is HTTP. 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.

WS-Security requires a signature in SOAP messages to prove
message integrity. A user is required to use the XML signature specification
provided by W3C organization to send a signature for each message sent to a
Web Service. The signature is defined using a <Signature> tag in the SOAP
message and included in parts of the security header. The signature should
not be wrapped around the entire header, as required in the XML Signature
Specification. Instead, WS-Security suggests signing certain sections of the
message. This way a message can be signed in different sections with
multiple signatures, allowing one message to flow through different stages in
a distributed application. If a Web Service receives a request with an invalid
signature, the message is rejected because it could have been tampered with
as it passed over the network.

WS-Security also specifies a method for providing confidentiality
in a Web Service. WS-Security suggests the XML Encryption specification
provided by W3C for encrypting confidential information sent in SOAP
messages. The encrypted key should either be a shared key by the sender and
receiver, or a key that is included in the message itself. Encryption is used
only in portions of a message, similar to how signatures are treated.

The ideas presented by WS-Security are very solid and helpful for
creating a Web Service application that is robust and secure. Web Services
need secure authentication to prevent unauthorized access, and to identify
users for its session information.
To acquire the message encryption in WS-Security the user credentials, an improved authentication scheme is proposed with two modules. First module is a dynamic nonce generator and second module is an authentication scheme combined with dynamic nonce. 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.

### 6.2.1 Design of Dynamic Nonce Generator

The nonces are generated from chaotic of computer parameters and from user’s activities. Figure 6.2 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.

![Figure 6.2 Architecture of dynamic nonce generator](image-url)
6.2.1.1 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. Preferably, the chaotic input will have full chaotic; however, the dynamic nonce mechanisms have been specified to allow for some bias in the chaotic input by allowing the length of the chaotic input to be longer than the required amount of chaotic. The chaotic input can be defined to be a changeable length (within specified boundary), as well as preset length. In all cases, the dynamic nonce mechanism expects that when chaotic input is requested, the returned bitstring will contain at least the requested amount of chaotic. Additional chaotic beyond the amount requested is not required.

6.2.1.2 Other inputs

Other information may be obtained by a DNG mechanism as input. This information must be 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.
6.2.1.3 Core state function

The core state function maintains the status of nonce and stores parameters, variables and other stored values that the DNG mechanism uses. It contains both the administrative data (e.g., the security strength) and data that is acted upon and/or modified during the generation of pseudorandom bits.

6.2.2 Functions of the Dynamic Nonce Mechanism

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

1. The initiate 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.

2. The generate function generates pseudorandom bits upon request, using the current nonce and generates a new nonce for the next request.

3. 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.

4. The uninitiate function nullifies (i.e., erases) the nonce.

5. The robustness test function determines that the dynamic nonce mechanism continues to function correctly.

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

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 6.3.

6.2.2.2 Core state function

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

1. Working nonce

   - One or more values that are derived from the seed and become part of the initial nonce; these values must usually kept secret
   - A count of the number of requests or blocks produced since the initiation was seeded or refed.
2. 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. A DNG mechanism implementation is designed to handle multiple initiations. Each DNG initiation has its own initial nonce. The internal nonce for one DNG initiation will not be used as the initial nonce for a different pseudo random number creation. A DNG changes initial nonce when the generator is requested to provide new pseudorandom bits.

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. That the security strength actually supported by a particular initiation must be less than the maximum security strength DNG implementation. For example, a DNG that is designed to support a maximum security strength of 256 bits (can be set as 112, 128, 192 and 256). Following initiation, requests can activate the generate function for pseudorandom bits. For each generate request, a security strength is provided for the bits is requested.

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.
6.2.2.3 Boundaries of DNG mechanism

A DNG mechanism boundary contains all DNG mechanism functions and status of nonce required for a DNG. Data enters a DNG mechanism boundary via the DNG’s public interfaces, which are made available to consuming applications.

Within a DNG mechanism boundary,

1. The status of the nonce and the operation impacts the function according to the specification of DNG mechanism.

2. The DNG status exists exclusively within the DNG mechanism boundary. The status maintained within the DNG mechanism boundary.

3. 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.

![Figure 6.4 Distributed DNG mechanism functions](image)

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 6.4. In addition, each
boundary or sub-boundary contains an uninitiate function in order to perform and/or react to health testing.

When DNG mechanism functions are distributed, appropriate mechanisms is used to protect the confidentiality and integrity of the internal state or parts of the internal state that are transferred between the distributed DNG mechanism sub-boundaries. The confidentiality and integrity mechanisms and security strength is reliable with the data to be protected by the DNG’s consuming application.

6.2.3 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. Periodic Refeeding is addressing the threat of either the nonce generator seed, chaotic input or working state being compromised over time.

6.2.3.1 Seed construction for initiation

Figure 6.5 Seed construction process for initiation

Figure 6.5 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.

6.2.3.2 Seed Construction for refeeding

![Diagram of Seed Construction](image)

**Figure 6.6 Seed construction for refeeding**

Figure 6.6 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.

6.2.4 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. The use of more chaotic than the minimum value offers rich level of security. This is useful if the assessment of the chaotic provided in the chaotic input is incorrect. Having more chaotic than the assessed amount is acceptable; having less chaotic than the assessed amount could be critical to security. The presence of more chaotic than is required, especially during the initiation, will provide a higher level of assurance than the minimum required chaotic.
6.2.4.1 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.

6.2.4.2 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.

6.2.4.3 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 \( \frac{1}{2} \) bits of chaotic or a value that is expected to repeat no more often than a \( \frac{1}{2} \)-bit random string would be expected to repeat.

For a case, the nonce is acquired from the same source and at the same time as the chaotic input. In this case, the seed is considered to be constructed from an “extrastrong” chaotic input and the optional personalization string, where the chaotic for the chaotic input is equal to or greater than \( \frac{3}{2} \) bits. The nonce ensures that the DNG provides \( \text{security_strength} \) bits of security to the consuming application.

6.2.5 ReInitiating

Generating too many outputs from a seed provide sufficient information for successfully predicting future outputs. Periodic reinitializing
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. Reinitializing is accomplished by 1) an explicit reinitializing 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.

Reinitializing of the DNG is performed in accordance with the specification for the given DNG mechanism. The DNG refeed specifications within this recommendation is designed to produce a new seed that is determined by both the old seed and newly obtained chaotic input that will support the desired security strength.

An alternative to reinitializing is to create an entirely new initiation. Though, reinitializing is preferred over creating a new initiation. If a DNG initiation was initially seeded with sufficient chaotic and the source of chaotic input subsequently fails without being detected, then a new initiation using the same (failed) source of chaotic input would not have sufficient chaotic to operate securely. Though, if there is an undetected failure in the source of chaotic input of an already properly seeded DNG initiation, the DNG initiation will still retain any previous chaotic when the refeed operation fails to introduce new chaotic.

6.2.6 Other Inputs to the DNG Mechanism

Other input is provided during DNG initiation, dynamic random bit generation and reinitializing. This input contains chaotic, but this is not required. During initiation, a personalization string is provided and combined with chaotic input and a nonce to derive a seed. When dynamic random bits are requested and when reinitializing is performed, additional input is provided.
6.2.7 **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:

- Device serial numbers,
- Public keys,
- User identification,
- Private keys,
- PINs and passwords,
- Timestamps,
- Special secret key values for this specific DNG initiation,
- Application identifiers,
- User activities like mouse movement
- Random numbers
- Nonces
6.2.8 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. The use of additional input may be a means of providing more chaotic for the DNG internal state that will increase assurance that the chaotic requirements are met. If the additional input is kept secret and has sufficient chaotic, the input can provide more assurance when recovering from the compromise of the chaotic input, the seed or one or more DNG internal states.

6.3 ARCHITECTURE OF DYNAMIC NONCE BASED AUTHENTICATION APPROACH

![Diagram of Dynamic Nonce Authentication Scheme for Web Services]

Figure 6.7 Architecture of dynamic nonce authentication scheme for Web Services
As illustrated in Figure 6.7, the following components are involved in authentication process.

6.3.1 Components of Service Consumer

The service consumer receives user interface from the service provider which is intended to user credentials at that time, it receives a concatenated result from dynamic nonce generator.

6.3.1.1 Chaotic input

The input is collected from the parameters of the system that is collected at the time of processing the instructions. This is collected from user’s mouse movement, time stamps, and application identifiers.

6.3.1.2 Dynamic nonce generator (DNG) and hash function

The nonces are generated from chaotic of computer parameters and from user’s activities. The DNG includes a source of chaotic input and includes a nonce source. Then it forwards the random number to hash generator. Afterwards, it is concatenated with user credentials, private keys and public key of the authentication.

6.3.1.3 Account database

A set of claims used to prove the identity of a client. They contain an identifier for the client and a proof of the client’s identity, such as a hashed password. This also includes information of user name and password which is concatenated with dynamic random nonce generator, to indicate that the issuer service identifies the claims in the credential.
6.3.1.4 Service consumer

The service consumer accesses the Web Service. The client provides credentials for authentication during the request to the Web Service.

6.3.2 Components of Service Provider

The following components are the significant part of service provider to maintain the secrecy of the authentication.

6.3.2.1 Chaotic input

The input is collected from the parameters of the system that is collected at the time of processing the instructions. This is collected from memory references, time stamps and application identifiers.

6.3.2.2 Dynamic nonce generator

The nonces are generated from chaotic of computer parameters and from activities of the server. The DNG includes a source of chaotic input and includes a nonce source. Then it forwards the random number to hash generator. Afterwards, it is concatenated with user credentials, private keys and public keys of the authentication.

6.3.2.3 Key generator and hash function

After receiving all information from the DNG the values are concatenated with public key, private key, user credentials and nonces. Then this result is passed in to hash function. The hashed result is forwarded into service provider.
6.3.2.4 Account database

Data passed among the consumer and provider contain information of service consumer account information, that must be protected. Consumer accounts are stored in a database service. The hashed random nonce generator is concatenated with the authentication.

6.3.2.5 Service provider

The service provider receives the authentication request from the requester as the hashed and concatenated result. Then it checks with the account database and starts required authentication steps and produces dynamic nonce. Again this information is forwarded to the service client. After completion of credential confirmation in third interaction the provider allows the client to access the services.

6.4 Algorithm for Proposed Authentication Approach

6.4.1 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: \((\text{status, chaotic_input}) = \text{Get_chaotic_input}(\text{min_chaotic, min_length, max_length})\)

Which requests a string of bits (chaotic_input) with at least \(\text{min_chaotic}\) bits of chaotic. The length for the string is equal to or greater than \(\text{min_length}\) bits and less than or equal to \(\text{max_length}\) bits. A status code is also returned from the function.

### 6.4.1.1 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.

\[
\text{Initiate}\_\text{function}(\text{requested_initiation_security_strength, prediction_resistance_flag, personalization_string})
\]

1. **Requested_initiation_security_strength**: A requested security strength for the initiation. The maximum security strength is controlled in this parameter.

2. **Prediction_resistance_flag**: Indicates whether or not prediction resistance is required by the consuming
3. 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.

1. If requested_initiation_security_strength > highest_supported_security_strength, then return an ERROR_FLAG.

2. If prediction_resistance_flag is set, and prediction resistance is not supported, then return an ERROR_FLAG.

3. If the length of the personalization_string > max_personalization_string_length, return an ERROR_FLAG.

4. Set security_strength to the nearest security strength greater than or equal to requested_initiation_security_strength.

5. Using security_strength, select appropriate DNG mechanism parameters.
// Obtain the chaotic input.

6. \((\text{status, chaotic\_input}) = \text{Get\_chaotic\_input}(\text{security\_strength, min\_length, max\_length})\).

7. If an ERROR is returned in step 6, return a CATASTROPHIC\_ERROR\_FLAG.

8. 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.

9. \(\text{initial\_working\_state} = \text{Initiate\_algorithm}(\text{chaotic\_input, nonce, personalization\_string})\).

10. Get a state\_handle for a currently empty internal state. If an unused internal state cannot be found, return an ERROR\_FLAG.

11. 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\_flag).

12. Return SUCCESS and state\_handle.

6.4.1.2 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).

If a refeed capability is not supported, a new DNG initiation is created. The refeed function checks the validity of the input parameters, obtains chaotic input with sufficient chaotic to support the security strength and combines the current working state with the new chaotic input and any additional input to determine the new working state.

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):

1) state_handle: A pointer or index that indicates the internal state to be refeeded. If a state handle is not used by an implementation because the implementation does not support multiple simultaneous initiations, a state_handle is not provided as input.

2) 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. Output the consuming application after refeeding is the status returned from the function. The status will indicate SUCCESS or an ERROR.

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.

1) Using state_handle, obtain the current internal state. If state_handle indicates an invalid or unused internal state, return an ERROR_FLAG.

2) If the length of the additional_input > max_additional_input_length, return an ERROR_FLAG.

//Obtain the chaotic input.

3) (status, chaotic_input) = Get_chaotic_input (security_strength, min_length, max_length).

4) If an ERROR is returned in step 3, return a CATASTROPHIC_ERROR_FLAG.

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

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

7) Return SUCCESS.

6.4.1.3 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):

1. state_handle: A pointer or index that indicates the internal state to be used.

2. requested_number_of_bits: The number of dynamic random bits to be returned from the generate function.

3. requested_security_strength: The security strength to be associated with the requested pseudorandom bits.
4. **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. \[\text{status} = \text{Refeed\_function} (\text{state\_handle, additional\_input})\]. If status indicates an error, then return status.

5. **additional\_input**: An optional input. The maximum length of the **additional\_input** (\text{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:

1. **status**: The status returned from the generate function. The status will indicate SUCCESS or an ERROR.

2. **Dynamic\_random\_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.

1. Using **state\_handle**, obtain the current internal state for the initiation. If **state\_handle** indicates an invalid or unused internal state, then return an ERROR\_FLAG.
2. If requested_number_of_bits > max_number_of_bits_per_request, then return an ERROR_FLAG.

3. If requested_security_strength > the security_strength indicated in the internal state, then return an ERROR_FLAG.

4. If the length of the additional_input > max_additional_input_length, then return an ERROR_FLAG.

5. If prediction_resistance_request is set, and prediction_resistance_flag is not set, then return an ERROR_FLAG.

6. Clear the refeed_required_flag.

7. If refeed_required_flag is set, or if prediction_resistance_request is set, then

   7.1 status = Refeed_function (state_handle, additional_input).

   7.2 If status indicates an ERROR, then return status.

   7.3 Using state_handle, obtain the new internal state.

   7.4 additional_input = the Null string.

   7.5 Clear the refeed_required_flag.

// Request the generation of pseudorandom_bits using the appropriate generate algorithm.

8. (status, pseudorandom_bits, new_working_state) = Generate_algorithm (working_state, requested_number_of_bits, additional_input).

9. If status indicates that a refeed is required before the requested bits can be generated, then
9.1 Set the refeed_required_flag.

9.2 Go to step 7.

10. Replace the old working_state in the internal state indicated by state_handle with the values of new_working_state.

11. Return SUCCESS and dynamic random_bits.

6.4.1.4 Removing a DNG initiation

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

1. Checks the input parameter for validity.

2. 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 ERROR_FLAG. The information retained within the DNG mechanism boundary after uninitiation: An empty internal state.
Uninitiate Process

1. If state_handle indicates an invalid state, then return an ERROR_FLAG.
2. Erase the contents of the internal state indicated by state_handle.
3. Return SUCCESS.

Then dynamic random nonce is accessed by the authentication approach of our proposed system.

6.4.2 Algorithm for the Dynamic Nonce Based Authentication Scheme

The existing systems provide the static Nonce for user identification. The static Nonce can be easily hackable by brute force attack. The proposed system has been implemented with dynamic Nonce with the use of client’s mouse movements.

The proposed system has been developed with one way hash function with Nonce and Time stamp. This scheme is an improved scheme from Yang et al (2005) schemes. The proposed system provides secure authentication to prevent unauthorized access and to identify users for its session information, but the information returned by existing Web Services does not signed and encrypted. This system encodes a string that includes the password and a timestamp using the SHA-1 hashing algorithm. Including a timestamp to the password before sending the message will prevent replay attack. This authentication is based on Diffie-Hellman Key Exchange and improves the security of the original Web Service authentication scheme. The important symbols are listed in Table 6.1.
Table 6.1 Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>F(.)</td>
<td>One way hash function</td>
</tr>
<tr>
<td>PW</td>
<td>Password</td>
</tr>
<tr>
<td>p</td>
<td>Prime number</td>
</tr>
<tr>
<td>g</td>
<td>G&lt;p and g is a primitive root of p</td>
</tr>
<tr>
<td>C, P</td>
<td>The user and the service provider</td>
</tr>
<tr>
<td>ID&lt;sub&gt;x&lt;/sub&gt;</td>
<td>The identity of the entity X</td>
</tr>
<tr>
<td>ΔT</td>
<td>Expected time interval</td>
</tr>
<tr>
<td>T&lt;sub&gt;c&lt;/sub&gt;, T&lt;sub&gt;s&lt;/sub&gt;</td>
<td>Time stamp of client and service provider</td>
</tr>
<tr>
<td>r&lt;sub&gt;x&lt;/sub&gt;</td>
<td>Private key r&lt;sub&gt;x&lt;/sub&gt;&lt;p for entity x</td>
</tr>
<tr>
<td>T&lt;sub&gt;x&lt;/sub&gt;</td>
<td>Public key for entity x</td>
</tr>
<tr>
<td>K&lt;sub&gt;x&lt;/sub&gt;</td>
<td>Secret key of client and provider</td>
</tr>
<tr>
<td>⊕</td>
<td>The XOR operator Nonce value of client and provider</td>
</tr>
</tbody>
</table>

6.4.3  **Registration and Login Phase**

The new user has to register the username and password to become a legitimate user of remote server. The user name and password are stored in the database of remote server. The registration and login phase of the proposed scheme is shown in Figures 6.8 and 6.9.
6.4.3.1 Registration phase

A new Client C sends the ID<sub>C</sub>, a hashed password F(PW) and T<sub>c</sub> to the service provider via secure channel. Then the service provider sends g and p to the respective client. Thus the C<sub>i</sub> client registers user Id and password with the service provider.

6.4.3.2 Login phase

Figure 6.8 Registration phase of the proposed scheme

Figure 6.9 Proposed scheme for an user authentication
If the Client C wants to get the resources of service provider P, P must authenticate the user U. To accomplish this C and P must perform the following steps.

The step by step procedure for user authentication is given below.

**Algorithm for authentication**

**Input:** user name, password  
**Output:** accept/reject

1. C selects private random integer $r_1 < p$ and calculates $t_1 = g^{r_1} \mod p$. The value $t_1$ is public.
2. C sends the service request with a dynamic Nonce $N_1$, username, $t_1 \oplus F(pw)$ and $T_c$.
3. Upon receiving message from C, P checks timestamp $\Delta T > T_s - T_c$ is true and stores $N_1$ into Nonce Table 1 and gets the value $t_1$ value by xoring. Provider P changes its id for every client from client’s id and Nonce value of client.
4. P selects private random integer $r_2 < p$ and computes $t_2 = g^{r_2} \mod p$. The value $t_2$ is public. Then Provider P calculates $K_1 = t_1^{r_2} \mod p$.
5. The provider P sends challenge message to C which holds the data of $\text{Challenge}(\text{realm}, t_2 \oplus F(pw), F(t_1, K_1), P, N_2, T_s)$.
6. After receiving message 2 from provider, the C verifies $\Delta T, N_1$ of incoming message and stores $N_2$. Then it calculates $M$ by XOR the value of $N_1, N_2$ and $K_2 = t_2^{r_1} \mod p$.
7. Client C finds the $t_2$ value from challenge message by XOR $F(pw)$ with $(t_2 \oplus F(pw))$. Then the client verifies $F(t_1, K)$ to authenticate the provider.
8. After authenticating the service provider P, it sends the response message to Provider P $\text{Msg3}: \text{Response (username, realm, F(username, realm, K_2), (M, T_c))}$.
9. Service provider receives the response message 3 from Client C. It verifies $\Delta T$ and $N_2$ by XOR operation with $M$ to get $N_1$ to authenticate the client.

**Figure 6.10 Algorithm for the proposed authentication approach**

Now, Client gets the available services from service provider.
6.5 IMPLEMENTATION

The secured system is implemented with C Sharp and WSE 3.0 in .net environment to provide role based security. The proposed system supports role-based authorization of SOAP messages by constructing a security token within the SOAP message. The server and client find the incoming message and compare the password digest to a stored digest of the correct password. The timestamp must be recent otherwise the server will reject the user’s login. The proposed system use public-key signatures to sign their messages so the server can be sure the contents of the message have not been viewed using diffie-helman key exchange algorithm and dynamic nonce.

Figure 6.11 Web Service configuration

The implementation of username token is used to implement authentication at the message layer. The client passes the credentials to the Web Service as part of a secure message exchange. A password is sent in the message as encrypted and hashed. The Web Service decrypts the
message, validates the credentials, verifies the message signature, and then sends an encrypted response back to the client.

Figure 6.12 Trace of WS-security of communication flow between client/server

The encryption method has been implemented in username token which is implemented in authenticate token. Then, it is accessed through token checker libraries. The TokenCheckerLibrary (dll file) is registered with the Web Service. Through by adding TokenCheckerLibrary and UserNameToken in add tag which is under SecurityTokenManager tag of Web.Config file in Figures 6.11 and 6.12. The add tag consists the details about UserNameToken like type, namespace and local name. The TokenChecker method has been called by service provider to retrieve the password of token Username to get a password from database for the given username. The service provider checks the password returned by TokenChecker and matches with the password in the SOAP header. If the
password in SOAP header does not match with server’s data, then server sends an exception will to the client.

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.

Figure 6.13 Hashed password and dynamic nonce for authentication schemes

<wsse:UsernameToken>
  <wsse:Username>admin</wsse:Username>
  <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-usernames-token-profile-1.0#PasswordDigest">RKFQ1+jYBqXRGeHqGrscRspompVM</wsse:Password>
  <wsse:Nonce>tdue7mEpYI95ZqMfEcg==</wsse:Nonce>
  <wsse:Created>2012-06-30T14:32Z</wsse:Created>
</wsse:UsernameToken>

<wsse:UsernameToken>
  <wsse:Username>admin</wsse:Username>
  <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-usernames-token-profile-1.0#PasswordDigest">185DA21B9aSo0f0C5892c99e091e25c8A072f94bc188</wsse:Password>
  <wsse:Password>
    lo/HO02Nj1f0cCH41jjH90E2vLM=<</wsse:Password>
  
  <wsse:Password>
    lo/HO02Nj1f0cCH41jjH90E2vLM=<</wsse:Password>
  </wsse:Password>
</wsse:UsernameToken>

<wsse:UsernameToken>
  <wsse:Username>admin</wsse:Username>
  <wsse:Password>
    lo/HO02Nj1f0cCH41jjH90E2vLM=<</wsse:Password>
  </wsse:Password>
</wsse:UsernameToken>

Figure 6.13 Hashed password and dynamic nonce for authentication schemes

<wsse:UsernameToken> element provides a countermeasure for replay attacks: <wsse:Nonce> and <wsu:Created> is shown in Figure 6.13. 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.
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 6.14.

6.6 RESULTS

![Figure 6.15 Service for dynamic nonce generation](image1)

![Figure 6.16 login service](image2)
The service for dynamic nonce is published to access it through service provider for authentication is shown in Figure 6.15. This service calls the DNG function to generate the random number. The service provider for authentication publishes the server page to consume the service from the client side is shown in Figure 6.16. The server maintains all incoming and outgoing message through Web.config using WS-Security.

### 6.7 SECURITY ANALYSIS

The security robustness of proposed scheme has been analyzed. The comparison with the related reviewed schemes on security properties of proposed scheme is summarized in Table 6.2 and its performance comparison has been listed in Tables 6.3 and 6.4.

**Table 6.2 Comparison of security on resisting attacks**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Replay attack</th>
<th>Offline password attack</th>
<th>Server spoofing guessing attack</th>
<th>Man-in-the-middle attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yang et al. (2005)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Wu and Weaver (2007)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Lee et al. (2005)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Shi and Yoo (2006)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Proposed scheme</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Table 6.3 Performance comparison**

<table>
<thead>
<tr>
<th>Authentication schemes</th>
<th>Dynamic nonce</th>
<th>Time stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yang et al (2005)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Shi &amp; Yoo (2006)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Proposed Scheme</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Table 6.4 Comparison based on methodologies of various schemes

<table>
<thead>
<tr>
<th></th>
<th>Encryption</th>
<th>Verification table Use</th>
<th>MACaddress</th>
<th>Mutual authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lee et al (2005)</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Shi &amp; Yoo (2006)</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Proposed scheme</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

6.7.1 Replay Attack

The proposed system is a timestamp-based password authentication scheme, the replay attack is prevented by checking the freshness of the message. The expected time interval can be set by service provider. The service provider and client maintain the Nonce table to check the freshness of the random values. If the random value exceeds the expected time interval, that is $T_x (N2') > \Delta T$, then the message of the attacker will be treated as old. Then the server or client discards the message.

6.7.2 Password Guessing Attack

An attacker tries to give different passwords by brute force or dictionary method from the legitimate user name in the Msg1: Request (username, t$_1'^\oplus$F(pw)', N$_1'$, T$_c'$). The server will not respond the attacker’s message. It is because attacker needs that exact value of t$_1$. 
6.7.3 Server Spoofing Attack

In this scheme, the client and service provider pre-shares the id and password. The message passed between the client and service provider needs id of sender for authentication. Even though the attacker knows the server id, the client rejects the message, because, the client analyses all incoming message for secret values. The attacker sends the message to client Msg2: Challenge (realm’, t_2’⊙F(pw)’, F(t_1,K_1)’, P’, N_2’, T_s’) the received message has checked for expected validity time, freshness of N_1, F(pw) ⊕(t_2’⊙F(pw)’) will return value t_2’ and K_2 = t_2’^{r_1} mod p’ is checked with value of K_1 is found false. The client will interpret that message sent by the server is a spoofed server.

6.7.4 Man-in-the Middle Attack

An attacker cannot get the password or key value of client by using the Msg2: Challenge (realm’, t_2’ ⊕F(pw)’, F(t_1,K_1)’, P’, N_2’, T_s’). Because, the client checks the id of the service provider for each incoming message. Also, the client checks the validity of P’, N_2’, T_s’.

6.8 PERFORMANCE EVALUATION

The proposed authentication system has been analyzed and identified that the system is capable to perform 100 request per second for 41 minutes and 10 seconds (00:41:10) as shown in Figure 6.17. During this period, it performs 30,814 requests.
Figure 6.17 Load testing setup of proposed dynamic authentication approach in Load UI 2.6.1

The LoadUI 2.6.1 testing tool is used perform load testing on proposed authentication approach. To do performance calculation, first the new project is opened in LoadUI environment. Then the Web page runner in testing tool is created and the URL path of the WSDL for authentication is given. Finally, the testing tool is started to record the performance. In this period, it completes 30,814 requests and failed to perform 4 requests at the time of ending. There are no discarded requests identified during load performance. This proves this approach capable of performing the requests efficiently.

Figure 6.18 The performance result of authentication approach against total request sent, response size, failures, TPS and BPS.
The number of requests sent to the proposed system is gradually increased from 0 to 30,814 are shown in Figure 6.18. The throughput of the system is equal and distributed evenly in the graph without any fluctuation. The failure rate of the system is clearly seen that there is no failures during the load testing.

Figure 6.19  The performance result of authentication approach against total request sent and completed

The graph in Figure 6.19, is catched at the time of completion of 6,000 requests. Here the request sent and completed requests are increased gradually without any fluctuation. It is clearly identified and recorded that the proposed system is capable performing the request without any failure or incompleted requests.

Figure 6.20 Performance comparison of authentication schemes
With increase in size of data, that the performance difference between the various algorithms has increased. At 5 concurrent users, MD5 is around 33% faster than SHA1. The performance of SHA512 is around 55% slower than SHA1. At final, the performance of proposed system is degraded with more data. It is around 50% slower than SHA512. That longer cycles of random number generation provides greater security at the cost of performance as shown Figure 6.20.

6.9 SUMMARY

In a Web Service context, a dynamic business process may involve many applications and services from different organizations and security realms which are combined at runtime and collaborate a peer-to-peer way. Web Services are a hurriedly growing technology, especially in the service oriented enterprise. Web Services have a lot to offer when it comes to creating enterprise-based applications for selling things over the internet. A new authentication scheme for Web Services applications is proposed in this thesis using a low cost effective method. It is well suitable for all types of systems with lower configurations. The proposed scheme has been analyzed for various types of attacks in Web Services environment.