# Preventing Session Hijacking using Encrypted One-Time-Cookies

Renascence Tarafder Prapty [1], Shuhana Azmin [1] Md. Shohrab Hossain [1], Husnu S. Narman[2]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh
[2]Weisberg Division of Computer Science, Marshall University, Huntington, WV, USA
Email: prapty101@gmail.com,shuhana.azmin@gmail.com, mshohrabhossain@cse.buet.ac.bd, narman@marshall.edu

***Abstract*—Hypertext Transfer Protocol (HTTP) cookies are pieces of information shared between HTTP server and client to remember stateful information for the stateless HTTP protocol or to record a user's browsing activity. Cookies are often used in web applications to identify a user and corresponding authenticated session. Thus, stealing a cookie can lead to hijacking an authenticated user's session. To prevent this type of attack, a cookie protection mechanism is required. In this paper, we have proposed a secure and efficient cookie protection system. We have used one time cookies to prevent attacker from performing cookie injection. To ensure cookie integrity and confidentiality, we have encrypted sensitive information in the cookie. We have verified that our proposed system can ensure confidentiality, authenticity and integrity through security analysis. Our proposed system can efficiently prevent session hijacking performed through replay attack and cookie poisoning attack.**

***keywords*—Reverse Proxy Server, One Time Cookie, Public-Key Cryptography, Symmetric Encryption, Session Hijacking.**

## I. Introduction

HTTP is a stateless protocol which does not require remembering each user's status across multiple requests. So web applications use HTTP session cookies to make stateless HTTP protocol stateful. An HTTP cookie is a small piece of information that is sent by the HTTP server and stored in the web browser. Parameters of a cookie are name, value, expiration, path, domain, and secure. Value is used to store a user's personal information including their user identification (ID), session ID, and email address. Cookies are used for session management, customizing web pages, saving shopping cart, etc.

Cookies create various security risks when used as session identification tokens. Cookies are stored on a browser and transferred over the Internet without any security mechanism. Thus, the user information stored in value can be easily acquired by an adversary using various cookie theft technologies. Various web applications use Hypertext Transfer Protocol Secure (HTTPS) to protect their communication. However, it is not possible to be "always-on HTTPS" due to performance and cost. Moreover, an attacker can still gain access to cookies by performing an attack against HTTPS and the web browser. Hence, using HTTPS is not a complete solution.

Several studies have been performed to address cookie-related security problems. Several schemes [1]–[4] use cryptography-based techniques to achieve cookie confidentiality and integrity. However, each mechanism lacks in one aspect or another. The scheme described in [1] does not achieve cookie confidentiality. In [3], the web server is required to store one-time keys that lead to key management problems. In [2] and [4], a one-time key is encrypted by the web server's public key. As public key encryption is computationally expensive, these schemes are not efficient. [5] proposes the use of cache cookies as an alternative to cookies for authentication of users. This scheme does not ensure cookie confidentiality and integrity. [6] proposes a mechanism using one-time cookies which ties a unique cookie to every HTTP request. However, this scheme relies on HTTPS to protect the credentials of a cookie. In [7], a secure and efficient cookie protection scheme is proposed based on self-verification. But this scheme uses static cookies. So it is still vulnerable to replay attack.

Our *objective* is to propose an efficient HTTP cookie protection scheme to simultaneously ensure cookie confidentiality, authenticity and integrity and to prevent session hijacking performed through replay attack and cookie poisoning attack.

The *contributions* of this work are: (i) Creating a reverse proxy server (RPS) that generates session management cookies, (ii) Creating a cryptography module based on self-verification to protect sensitive information in cookies.

Our proposed reverse proxy server uses the concept of a one-time cookie where each HTTP request is tied to a unique session token. Our proposed cryptography module ensures confidentiality, authenticity and integrity of cookies.

Our proposed scheme *differs* from existing solutions in protection against cookie replay attack, cookie poisoning attack and in requirement of no additional state by the Web server. We use a unique cookie to authenticate each request. This cookie comprises of encrypted sensitive information, plain text nonsensitive information and a digital signature created by using both sensitive and nonsensitive information. Usage of unique cookie per request makes it difficult to reuse cookies to get unauthorized access. As we have used the self-verification method, no additional state is generated in the Web server. During decryption, the web server retrieves the symmetric key from the digital signature. Nonsensitive information in the cookie is used to derive symmetric key used for encryption and decryption. If nonsensitive information of cookie is modified in the client side, decryption of sensitive information in RPS fails. Thus, the proposed encryption has twofold benefits: it preserves the confidentiality of sensitive information and also the integrity of non-sensitive information.

The rest of the paper is organized as follows. Related works on session hijacking and cookie protection are briefly discussed in Section II. In Section IV, our proposed architecture is explained along with its implementation details. Section V describes how confidentiality, authenticity, the integrity of cookie and prevention against replay attack is ensured in our proposed model. Timing analysis of cryptography operations are given in short in Section VI Finally, we conclude the paper in Section VII.

## II. Related Works

Several works have been done with the goal to prevent cookie hijacking. Part et al. [2] proposed secure cookies based on public-key cryptography to provide cookie confidentiality and integrity and authenticity. However, a public-key cryptosystem is relatively slow compared to symmetric key cryptosystem. Fu et al. [1] proposed a simple web client authentication scheme based on unforgeable cookies with explicit expiration time. However, this scheme is vulnerable to replay attacks. Liu et al. [8] proposed a secure cookie protocol based on symmetric cryptosystem. Blundo et al. [3] proposed a scheme using one-time-keys for browsers. These keys are stored in memory which leads to key management problems. Juels et al. [5] proposed the use of cache cookies to provide user authentication. However, it does not ensure the confidentiality and integrity of a cookie. Alabrah et al. [9] proposed a mechanism which utilized hybrid two dimensional one way hash chains and sparse caching to authenticate session of social media users. However, effectiveness of this mechanism to prevent session hijacking is not explored. Yang et al. [4] suggested a scheme that provides better confidentiality, integrity and authenticity based on the Public Key Infrastructure (PKI). Calzavara et al. [10] investigated prevailing session hijacking techniques, reviewed and assessed existing prevention mechanisms and listed few guidelines commonly followed by designers to mitigate session hijacking.

Another system is proposed in [11] which uses OTC to prevent Session Hijacking. This system includes a Reverse Proxy Server to issue and validate OTC. Implemented RPS checks IP address, session ID, OTC and browser fingerprint. Lee et al. [7] proposed a secure and efficient cookie protection scheme, having three phases: cryptography key generation, cookie issue and Login. In [12] the idea of n Sub Sessions is explained. Cookie Scope Fragmentation is identified as root cause for Sub Session Hijacking.

Some works use One Time Cookies to prevent session hijacking and some other works use Encryption in Cookies to protect sensitive information. But unlike our proposed method, no system uses these two mechanisms together for prevention of session hijacking. Our proposed system is unique in this approach.

## III. Terminologies

In this section, we present a brief description of cookie and cookie hijacking attack.

### A. HTTP Cookie

An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to a user's web browser. The browser may store it and send it back with the next request to the same server. Typically, it is used to check whether two requests have come from the same browser keeping a user logged-in, for example. It remembers stateful information for the stateless HTTP protocol. Cookies are mainly used for three purposes: session management, personalization and tracking. A server generates cookies and sends them to the browser in responses as HTTP headers as follows:

$$Set-Cookie :< cookie-name >=< cookie-value >$$

the server can set other parameters such as domain and path (cookies scope), expiration (cookies validity period), HttpOnly flag (defines if the cookie can be accessed by client-side scripts) and Secure flag (defines if the cookie can only be sent over a secure channel; i.e., HTTPS).

### B. Cookie Hijacking Attack

Although cookies can be conveniently used by users, they are transferred without any security mechanism over the internet. Thus, personal information stored in values can be easily acquired by an eavesdropper using various cookie theft technologies. Tools such as Firesheep [13] are used to intercept unencrypted session cookies. Cross-site scripting (XSS) attacks can be used to send a malicious script to an unsuspecting user's browser. The malicious script can access any information on a cookie stored by the user's browser, and this information can then be used on the associated website. Cross-Site Tracing (XST) and DNS cache poisoning attacks can also be used to steal cookies from the browser. Moreover, cookies can be tampered with on browsers. Any parameters of a cookie can be altered for malicious purposes. For example, the expiration parameter could be modified to extend the login session.

## IV. Proposed Architecture

Our proposed model consists of two modules:

1) Reverse Proxy Server: RPS is responsible for issuing and checking OTC. We follow the design of the reverse proxy server proposed in [11].
2) Cryptography Operations Module (COM): It is responsible for assurance of Confidentiality, Authenticity and Integrity for OTC. Cryptography operations performed on session cookies in [7] provide a general guideline for our proposed module.

Fig. 1 shows the sequence of operations performed in the proposed architecture. When a user sends a login request, RPS forwards this request to the server. If the server authenticates a user, RPS generates an OTC and encrypts the sensitive part comprising of a session ID and expiry time. RPS also creates

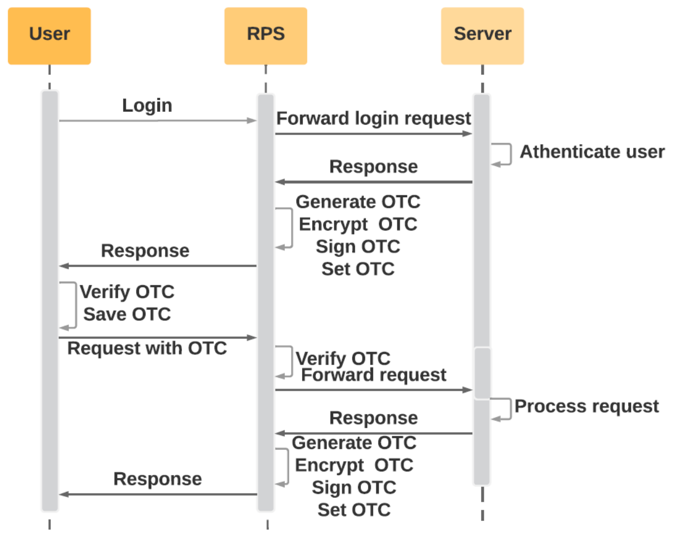


Fig. 1. Proposed Architecture

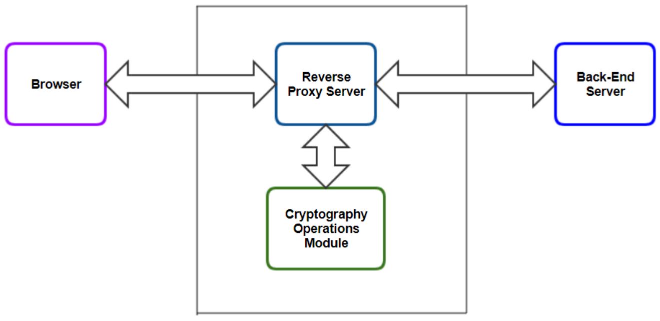


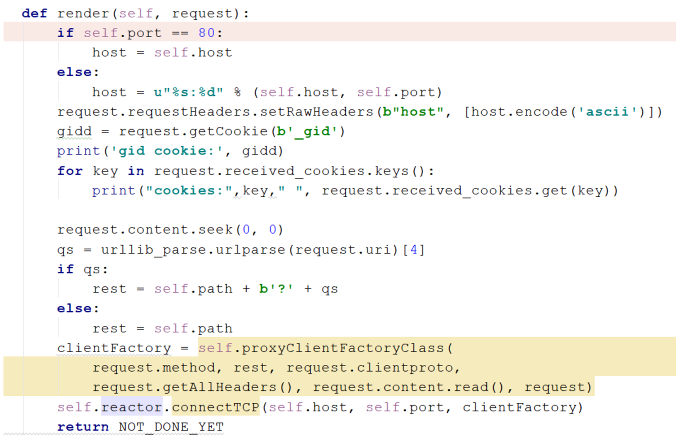
Fig. 2. Interactions of RPS

a digital signature from the contents of the OTC. The response from the server is sent to the user along with OTC. Client's browser verifies digital signature contained in the OTC and if verification is successful, then it stores OTC and includes it in the next request to the server. Again, this request comes to RPS first and RPS verifies OTC. The request is forwarded to the server only if provided OTC is correct. The response from the server to the client is also routed through RPS so that RPS can generate a new OTC, encrypt sensitive part and create a digital signature. Every subsequent request and response between client and server follows this same procedure until the session ends.

## A. Reverse Proxy Server(RPS)

All communication between the client's browser and back-end server is routed through RPS. An OTC is issued to the client's browser per request. RPS works in tandem with COM to ensure confidentiality, authenticity and integrity of OTC. Interactions of RPS with other components are shown in Fig. 2.

The functionalities of the proposed RPS are:



```
def render(self, request):
    if self.port == 80:
        host = self.host
    else:
        host = u"%s:%d" % (self.host, self.port)
    request.requestHeaders.setRawHeaders(b"host", [host.encode('ascii')])
    gidd = request.getCookie(b'_gid')
    print('gid cookie:', gidd)
    for key in request.received_cookies.keys():
        print("cookies:",key," ", request.received_cookies.get(key))

    request.content.seek(0, 0)
    qs = urllib_parse.urlparse(request.uri)[4]
    if qs:
        rest = self.path + b'?' + qs
    else:
        rest = self.path
    clientFactory = self.proxyClientFactoryClass(
        request.method, rest, request.clientproto,
        request.getAllHeaders(), request.content.read(), request)
    self.reactor.connectTCP(self.host, self.port, clientFactory)
    return NOT_DONE_YET
```

Fig. 3. Code Snippet from RPS

- Collection of IP address and browser fingerprint from the client side
- Generation of a session ID and OTC
- Matching IP address, browser fingerprint and session ID along with OTC

TwistedWeb has been used to implement RPS. When a request comes, an OTC is generated and set in RPS. A code fragment from the implementation of RPS is shown in Fig. 3.

## B. Cryptography Operation Module(COM)

As the name suggests, COM performs all cryptography operations required in the proposed architecture to ensure confidentiality, authenticity and integrity of the OTC. Main functionalities of this module are as follows:

- Generating long term asymmetric key pair
- Breaking OTC into nonsensitive component ($C_{i1}$) and sensitive component ($C_{i2}$)
- Selection of different secret parameter (k) for each OTC
- Generation of Symmetric Key (SK) from $C_{i1}$ and k
- Encryption of $C_{i2}$ using SK
- Generation of digital signature for this partially encrypted OTC
- Verification of digital signature by browser
- Retrieving k from the digital signature during verification of OTC
- Reconstruction of Symmetric Key (SK) from k and $C_{i1}$ during verification of OTC
- Detection of any modification in the OTC sent from the client

COM has been implemented with the help of Python Cryptography library.

RSA algorithm has been implemented to generate Asymmetric Key Pair. The implementation process is described below:

- Randomly selecting a large Prime Number p
- Calculating a Primitive Number $g \in GF(p)$
- Randomly selecting Private Key $x \in [1, p-1]$
- Calculating Public Key $y = g^x \mod p$

```python
def get_secret_parameter(p):
    while True:
        k = random.randrange(1, p - 1)
        if gcd(k, (p - 1)) == 1:
            break
    return k


def generate_secret_key(data, k):
    value = data + str(k)
    h = hashlib.sha256(value.encode('ascii')).digest()
    return base64.urlsafe_b64encode(h).decode('ascii')
```

Fig. 4. Symmetric Key Generation

During generation of each OTC, a secret parameter k is calculated such that $k \in [1, p-1]$ and $gcd(k, p-1) == 1$.

Non-sensitive content of OTC and k are concatenated and hashed using the SHA256 algorithm to generate a symmetric key. This process can be expressed as follows:

$$SK = h(C_{i1}||k) \tag{1}$$

The code fragment in Fig. 4 shows generation of a symmetric Key.

Symmetric key is used in Python Cryptography Module to encrypt sensitive content. It can be described as $T_i = E_{SK}(C_{i2})$. Here $E_{SK}()$ is the Encryption function.

Digital signature (r,s) of OTC is created using the following equations:

$$r = g^k \mod p \tag{2}$$

$$s = x * (r + h(C_{i1}||T_i) - k \mod (p-1) \tag{3}$$

where, $C_{i1}, t_i, r, s$ are sent to client as part of the OTC. To check the authenticity of OTC, the client's browser can verify the digital signature using the following equation:

$$y^{r+h(C_{i1}||T_i)} = r * g^s \mod p \tag{4}$$

Client's browser includes provided OTC in next request.

During verification of an OTC, k is retrieved from digital signature using following equation:

$$k = x * (r + h(C_{i1}||T_i) - s \mod (p-1) \tag{5}$$

symmetric key is reconstructed using Eqn. (1).

Digital Signature Creation and Secret Parameter Retrieval functions are shown in Fig. 5

Symmetric key is used in Python Cryptography Module to decrypt the encrypted sensitive content. It can be described as $C_{i2}) = D_{SK}(T_i$. Here $D_{SK}()$ is the Decryption function.

Encryption and Decryption functions are shown in Fig. 6

All the above-mentioned equations used in cryptography operations have been taken from [7].

```python
def create_digital_signature(data, g, k, x, p):
    r = power(g, k, p)
    m = int(hashlib.sha1(data.encode('utf-8')).hexdigest(), 16)
    val_1 = x * (r + m)
    val_2 = k % (p - 1)
    s = val_1 - val_2
    return r, s


def retrieve_secret_parameter(data, r, s, x, p):
    m = int(hashlib.sha1(data.encode('utf-8')).hexdigest(), 16)
    val_1 = x * (r + m)
    val_2 = s % (p - 1)
    k = val_1 - val_2
    return k
```

Fig. 5. Digital Signature Creation and Secret Parameter Retrieval

```python
def symmetric_encryption(plain_text, key):
    plain_text_bytes = plain_text.encode('utf-8')
    f = Fernet(key)
    cipher_text_bytes = f.encrypt(plain_text_bytes)
    cipher_text = cipher_text_bytes.decode("utf-8")
    return cipher_text


def symmetric_decryption(cipher_text, key):
    cipher_text_bytes = cipher_text.encode('utf-8')
    f = Fernet(key)
    plain_text_bytes = f.decrypt(cipher_text_bytes)
    plain_text = plain_text_bytes.decode('utf-8')
    return plain_text
```

Fig. 6. Encryption and Decryption

## V. Security Analysis

In this section, we explain how our proposed system can ensure confidentiality, authenticity and integrity through security analysis of our proposed system.

### A. *Ensuring confidentiality*

The sensitive part of the OTC which contains session ID and expiry time is encrypted by secret key SK. SK is not stored in RPS or transmitted to the Client over the network. Hence, any eavesdropper cannot sniff it from the transmission link. It can be regenerated only by the RPS from $C_{i1}, k$.

On the other hand, k is not stored anywhere. RPS alone can retrieve k from the equations of r and s. Attacker must solve the complex discrete logarithm problem from Eqn. (2) to retrieve k by using the equation of r. It is also impossible to retrieve k from the equation of s because two unknown secret parameters (k and x) are involved, as presented in Eqn. (3). As a result, no attacker can have access to SK and sensitive component of a cookie is accessible to only RPS. Hence, the confidentiality of the sensitive part of the cookie is ensured.

### B. *Ensuring authenticity*

RPS signs OTC with its private key using Eqn.(2) and Eqn.(3). Client's browser can use the public key of RPS to check the authenticity of OTC by using Equation 4. If an

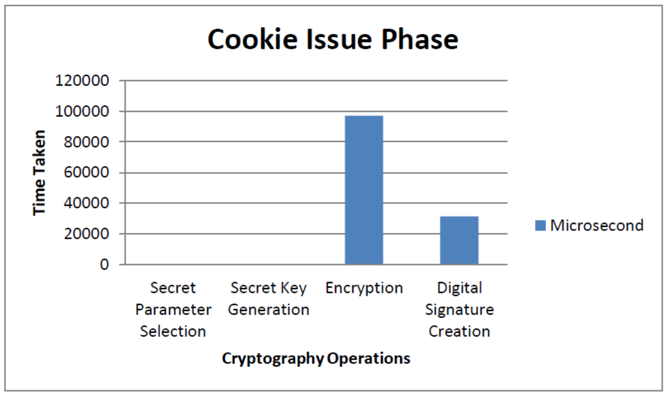


Fig. 7. OTC Issue Phase

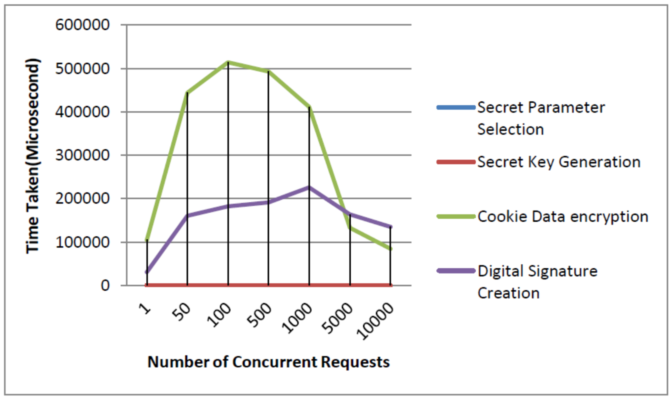


Fig. 8. Performance of OTC Issue Phase for varying number of requests

attacker forges a signature without using the private key of RPS, the signature verification fails. This way the client's browser can be sure of the authenticity of the cookie.

### C. Ensuring integrity

Encryption of cookie also ensures its integrity. If any of $C_{i1}, C_{i2}, t_i, r, s$ of OTC is modified, Equation.5 gives wrong k. Consequently, regenerated SK is wrong. As a result, decryption operation fails and cookie modification is detected.

### D. Prevention against replay attack

For each request, an One Time Cookie(OTC) is generated by RPS with a unique session ID and very short expiry time. RPS matches session ID of OTC returned from browser with expected value. It checks expiry time too. Hence an attacker cannot perform replay attack by using an expired or already used OTC with a new request. Our mechanism also ensures that an attacker cannot modify OTC to extend expiry time. So replay attack through this way is not possible either.

## VI. Timing Analysis

We have performed timing analysis for both OTC issue phase and OTC verification phase.

A bar chart showing the timing break down for all operations in the OTC issue phase is shown in Fig. 7. Cryptography operations are placed along X-axis and corresponding relapsed times in microseconds are placed in Y-axis. Secret parameter selection and secret key generation operations are almost instant. On the other hand, Encryption and Digital signature creation operations take a small amount of time. As a result, cryptography operations in OTC issue phase do not negatively impact the overall performance.

There are three main operations during OTC Verification Phase: secret parameter retrieval, secret key generation and decryption. All these operations occur instantly taking zero time when measured in microseconds. So, OTC verification phase does not have any additional overhead because of cryptography operations.

We have also experimented on performances of all cryptography operations during OTC issue phase and OTC verification phase for different number of concurrent requests.

Fig. 8 shows behaviours of cryptography operations during OTC Issue Phase. In this line chart, the number of requests is placed in X-axis and the time taken by concerned operations is placed in Y-axis. Each operation is represented by a line of a different colour. From Fig. 8, we can see that Secret Parameter Selection and Secret Key Generation are along the X-axis, meaning a very small amount of time is required for these two operations. These two operations take almost constant times irrespective of the number of requests. On the other hand, lines for Cookie Data Encryption and Digital Signature Creation have somewhat parabolic shapes. The time required for Cookie Data encryption and Digital Signature Creation at first increases along with the number of requests and after obtaining a certain value they decrease despite an increased number of requests.

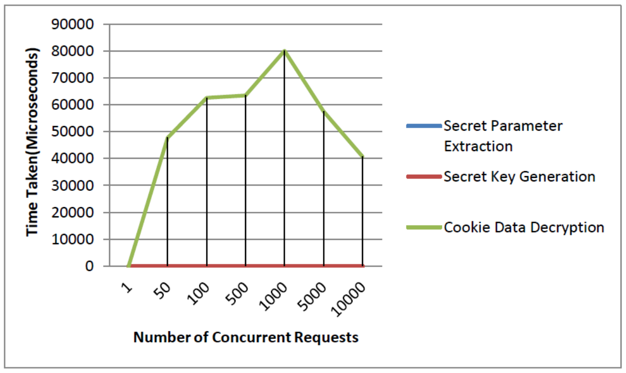


Fig. 9. Performance of OTC Verification Phase for varying number of requests

A line chart showing the time taken by different cryptography operations during OTC Verification Phase is given in Fig. 9. We have a varied number of concurrent requests and show the time taken by concerned operations. From Fig. 9, we can see that Secret Parameter Extraction and Secret Key Generation requires a very small amount of time and these times are constant irrespective of the number of concurrent requests. On the other hand, the line for Cookie Data Decryption has an almost parabolic shape. The time required for Cookie Data encryption at first increases with the increase of the number of concurrent requests and after obtaining a certain value it decreases despite increase of number of requests.

## VII. Conclusion

In this paper, we have proposed a novel solution to simultaneously ensure cookie confidentiality, authenticity and integrity and to prevent session hijacking through replay and cookie poisoning attacks. We have implemented a reverse proxy server to issue and verify one time cookies as session cookies and a custom cryptography operations module to manage encrypted one time cookies. We have performed a security analysis for the verification of our proposed system. Our scheme can prevent session hijacking through replay attack and cookie poisoning attacks by using OTC instead of expensive HTTPS connections.

## References

[1] K. Fu, E. Sit, K. Smith, and N. Feamster, “The dos and don’ts of client authentication on the web,” in *USENIX Security Symposium*, Washington, DC, USA, August 13-17, 2001, pp. 251–268.

[2] J. S. Park and R. Sandhu, “Secure cookies on the web,” *IEEE Internet computing*, vol. 4, no. 4, pp. 36–44, 2000.

[3] C. Blundo, S. Cimato, and R. De Prisco, “A lightweight approach to authenticated web caching,” in *Symposium on Applications and the Internet*. Trento, Italy: IEEE, 4 Feb. 2005.

[4] J.-P. Yang and K. H. Rhee, “A new design for a practical secure cookies system,” *Journal of information science and engineering*, vol. 22, no. 3, pp. 559–571, 2006.

[5] A. Juels, M. Jakobsson, and T. N. Jagatic, “Cache cookies for browser authentication,” in *IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE, 21-24 May, 2006.

[6] I. Dacosta, S. Chakradeo, M. Ahamad, and P. Traynor, “One-time cookies: Preventing session hijacking attacks with stateless authentication tokens,” *ACM Transactions on Internet Technology*, vol. 12, no. 1, p. 1, 2012.

[7] W.-B. Lee, H.-B. Chen, S.-S. Chang, and T.-H. Chen, “Secure and efficient protection for HTTP cookies with self-verification,” *International Journal of Communication Systems*, vol. 32, no. 2, 2019.

[8] A. X. Liu, J. M. Kovacs, C.-T. Huang, and M. G. Gouda, “A secure cookie protocol,” in *14th International Conference on Computer Communications and Networks*. San Diego, CA, USA: IEEE, 17-19 Oct. 2005.

[9] A. Alabrah and M. Bassiouni, “Preventing session hijacking in collaborative applications with hybrid cache-supported one-way hash chains,” in *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Miami, FL, USA: IEEE, 22-25 Oct. 2014.

[10] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta, “Surviving the web: A journey into web session security,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1–34, 2017.

[11] A. M. Sathiyaseelan, V. Joseph, and A. Srinivasaraghavan, “A proposed system for preventing session hijacking with modified one-time cookies,” in *International Conference on Big Data Analytics and Computational Intelligence*. Chirala, India: IEEE, 23-25 March 2017, pp. 451–454.

[12] S. Calzavara, A. Rabitti, and M. Bugliesi, “Sub-session hijacking on the web: root causes and prevention,” *Journal of Computer Security*, vol. 27, no. 4, pp. 1–25, Oct. 2018.

[13] E. Butler, “Firesheep,” 2010. [Online]. Available: https://codebutler.github.io/firesheep/