ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Architecting Secure REST APIs with Authentication and Authorization Approaches for Web Services

Rajesh Kotha

Software Development Engineering Advisor at Fiserv, USA

ABSTRACT

A collection of best practices and instructions for handling API inquiries are introduced in the REST standard for software architecture. It eliminates the need for complicated documentation by streamlining the request and answer process in a novel and straightforward manner without using HTTP and URI addresses instead of extra encapsulation, like in "Simple Object Protocol." The article expounds on the HMAC technique, which permits authentication and ensures the secrecy, integrity, and non-repudiation of REST services. Slim Framework was used to create certain examples. "The HMAC Algorithm is presented in Figure 1". The technique should be used to secure REST APIs since it is an easy-to-use and more secure web-service protection system.

***Corresponding author**

Rajesh Kotha, Software Development Engineering Advisor at Fiserv, USA.

Received: June 01, 2023; **Accepted:** June 04, 2023, **Published:** June 10, 2023

Keywords: REST APIs, Authentication, Authorization, HMAC (Hash-Based Message Authentication Code), Security Protocols, API Security, Slim Framework, Message Authentication Codes (MACs), OWASP, Cryptographic Hash Functions, Secure Communication

Introduction

Today, Java s, or "Application Programming Interface" (APIs), are used and shared by apps more and more. It is important to think about making sure these services have the right amount of security. The security level is primarily provided via procedures known as "Message Authentication Codes, (MACs), which depend on secret keys. Figure 2 "depicts how to compute a MAC over the data 'text' using the HMAC function" [1]. The latter is mostly employed to authenticate data. HMAC is a hashing algorithm using stored information in modern cryptography [2]. It is used by MAC codes. This paper introduces the HMAC technique, which guarantees the secrecy, credibility and non-repudiation of REST services. Using the Slim Framework, the paper uses a Restful API demonstration. A few endpoints have been designated for login, a test route that is only accessible to registered users, and authentication is done through the use of the HMAC technique. The technique is a secure, simple and easy-to-use option.

Steps	Description
1	If the length of K = B: set K_0 = K. Go to step 4.
2	If the length of K > B: hash K to obtain an L byte string, then append (B-L) zeros to create a B-byte string K_0 (i.e., K_0 = h(k) ‖ 00...00). Go to step 4.
3	If the length of K < B: append zeros to the end of K to create a B-byte string K_0 (e.g., if K is 20 bytes in length and B = 64, then K will be appended with 44 zero bytes x'00').
4	Exclusive-Or K_0 with *ipad* to produce a B-byte string: $K_0 \oplus$ *ipad*.
5	Append the stream of data *'text* to the string resulting from step 4: ($K_0 \oplus$ *ipad)* ‖ *text*.
6	Apply H to the stream generated in step 5: H(($K_0 \oplus$ *ipad)* ‖ *text)*.
7	Exclusive-Or K_0 with *opad*: $K_0 \oplus$ *opad*.
8	Append the result from step 6 to step 7: ($K_0 \oplus$ *opad)* ‖ H(($K_0 \oplus$ *ipad)* ‖ *text)*.
9	Apply H to the result from step 8: H(($K_0 \oplus$ *opad)*‖ H(($K_0 \oplus$ *ipad)* ‖ *text))*.

Figure 1: HMAC Algorithm [1].

```
MAC(text) = HMAC(K, text) =
H((KO ⊕ opad ) || H((KO ⊕ ipad) || text))
```

Figure 2: "How to Compute a MAC Over the Data 'Text' using the HMAC Function" [1].

The transition from on-premises infrastructure to serverless cloud-based solutions suggests a dramatic change. This evolution entails several innovative changes that will completely change how businesses and software developers approach their technological environments. Conventional infrastructure supervision involves several operational tasks, such as setting up servers performing repairs, and monitoring them (Figure 2). These responsibilities are transferred to the cloud provider via serverless cloud systems [3]. As a result, developers can focus on tasks that bring value rather than infrastructure issues because of this laissez-faire management approach. Digital market, enterprises can remain ahead of the competition, produce new products faster, and drive company success by utilizing the advantages of serverless computing.

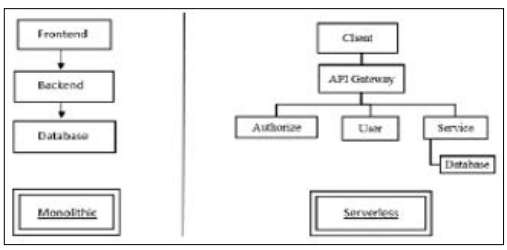

Figure 3: Traditional Infrastructure Vs. Serverless Cloud

Scope

The article provides a detailed explanation of the HMAC technique, which enables authentication and ensures the secrecy, integrity, and non-repudiation of REST services, drawing on a number of sources. Slim Framework was used to create certain examples, which included multiple login endpoints, test routes accessible

exclusively to registered users, and the assignment of authenticated HMAC techniques. When compared to other established techniques for authentication and authorization, the study's prefers HMAC mechanism as an easy-to-implement substitute to secure REST APIs for Java web services.

Problem Statement

Given how vital APIs are to the success of modern businesses, attacks against them have become more frequent. However, because APIs or Java are essential to the success of modern enterprises, they are becoming a common target for attacks. The Open Web Application Security Project (OWASP) was prompted to create a separate project to monitor the top ten most critical API security risks due to the growing threat posed to APIs due to their greater privileges and connectivity [4]. OWASP currently maintains a list of the top ten general security threats to web applications. For users with various positions, most APIs offer varying levels of authorization. This is why problems with permission and authentication frequently result in massive security holes as they let people perform tasks and access information that they are not authorized to access.

API injection occurs when malicious code is injected through an API, often within the parameters or body of a request [5]. This will be executed on the server side if it is included in HTML content and has not been properly escaped previously. Users of an API can obtain blog articles by submitting a GET request similar to the one below (Figure 3). It will run on the server side, for example, in a SQL query or during data deserialization. This may potentially lead to uncontrolled remote code execution or alter database records, which must be considered sacrosanct. In order to ensure that no improperly formatted or dangerous data enters the workflow, it is crucial that developers utilize Web Application Firewalls and perform input validation, sanitization, and encoding at the method level. The prevalence of this vulnerability can be reduced by using other libraries, such as OWASP's ESAPI. "The API will return post 12358 as a result of this request (Figure 4). Using a SQL query, the server will obtain the relevant blog post from the database; Post-id is the user-provided id that the user submitted via the URL" [5].

```
GET /api/v1.1/posts?id=12358
```

Figure 4: Users of an API can Obtain Blog Articles by Submitting a GET Request [5].

```
SELECT * FROM posts WHERE post_id = 12358
```

Figure 5: The API will Return Post 12358 [5].

These vulnerabilities can infiltrate an API and even Java in a number of ways. An organization could, for example, neglect to carry out the method-level permission checks required to confirm that an object a user is reading or writing to is inside their allowed scope. The next vulnerability—a lack of granularity in read and write—is also indicated by the possibility of incorrectly checked permissions in situations where permissions on individual objects are much finer during the authorization process to API endpoints. Additionally, by utilizing cryptography, such public key encryption, these teams could be unable to safeguard the user authentication procedure, opening the door to spoofing. APIs provide communication between various internal systems that often transfer or retain sensitive data, in addition to enabling user interaction with apps. Therefore, an application's security posture can be substantially compromised by insecure API, which can act as an entry point for attackers. Attacks that start within Web services are prevented and lessened by supporting API security with the HMAC method. API security is one of the most essential building blocks for any organization's comprehensive security plan.

Literature Review

The REST standard introduces a collection of best practices and instructions for handling API inquiries which is part of the software architecture. It eliminates the need for complicated documentation by streamlining the request and answer process in a novel and straightforward manner. URI (Uniform Resource Identifier) addresses and HTTP are used instead of extra encapsulation, as in SOAP [6]. Take the Conventional State Access Protocol or the Hypertext Transfer Protocol, for instance. The behavior and state of an application are broken down into components known as resources. Every resource uses the same interface to change the state. Leonard Richardson created the "Richardson Maturity Model," which is shown in Figure 5. It explains the foundations of REST. Using the HTTP layer as the transport is where the maturity model begins.

Level 0: This involves invoking a remote procedure. Data sent as POX (Plain Old XML) using XML-RPC or SOAP technology is included in Level 0. There is only usage of the POST techniques. This is the simplest technique for developing SOA (Service-Oriented Architecture) applications.

Level 1: "REST resources. In Level 1, POST methods are covered, and REST URIs are utilized in place of functions and parameter passing". A single HTTP method is employed. Level 1 has an advantage over level 0 in that it can divide a complex functionality into several resources.

Level 2: Additional HTTP verbs. Other HTTP verbs including GET, POST, PUT, and DELETE are covered in Level 2. "When many HTTP verbs are used to request methods and the system may contain a range of resources, it is a more realistic usage of REST technology".

Level 3: "Hypermedia as the Engine of Application State. The most developed level in Richardson's model is HATEOAS". Hypermedia controls are included in the answers to the client's requests, which might help the client determine what to do next. This level fosters simple discoverability and simple understandability.

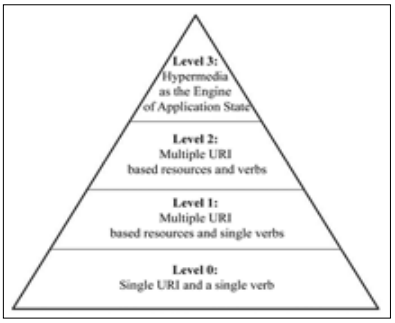

Figure 6: Richardson Maturity Model [1].

Any technique that does not change the server's state is safe. A technique is said to be idempotent if the consequences of using it for further requests are the same as the consequences of making a single request. Figure 6 provides an illustration of them. GET is secure since it merely permits resource access and does not

change the server's state. This method sends requests that contain parameters and are cached. PUT and POST methods, on the other hand, are risky as they alter or create server resources. Also, the DELETE approach is dangerous because it would erase a resource from the server.

Method	Safe		Idempotent	
	YES	NO	YES	NO
GET	X		X	
POST		X		X
PUT		X	X	
DELETE		X	X	

Figure 7: Safety and "Indempotence" Methods [1].

The RESTful API demo's entire design, development, and testing process was broken down into multiple stages. There are general scheme design principles that ought to be followed in such a process. This included assigning multiple login endpoints, test routes that were only accessible to registered users, and authentication via the HMAC mechanism.

Stage 1: This involves "identifying the resource URIs" [1]. In REST technology, one of the most crucial ideas is resources. Anything that piques curiosity enough for users to get to know more is considered a resource. A resource URI can be used to address any RESTful resource. Since resources are addressed via URIs, REST is extendable. Two categories of resource addresses exist: the collection and single address. Each of the preceding samples (Figure 7) displays an easily legible pattern that the client is able to decipher.

URI	Description of the URI
/v1/users	this is used to represent all users
/v1/users/1234	this is used to represent a user in a system identified by '1234'
/v1/users/1234/auctions	this is used to represent all the auctions for a user identified by '1234'

Figure 8: Sample URIs [1].

Stage 2: "Identifying the methods supported by the resource. The consistent interface constraint, which establishes the relationship between the activities indicated by the verb and the noun-based REST resource, is mostly composed of HTTP verbs" [1]. Figure 8 provides a summary of HTTP methods along with an explanation of their actions. "The HTTP verbs instruct the server on what to do with the data that is sent with the URL" [1]. Figure 9 presents an overview of some significant HTTP methods within the framework of REST.

HTTP method	Resource URI	Description
GET	/users	gets a list of users
GET	/users/1234	gets a user identified by '1234'
POST	/users	creates a new user
PUT	/users/1234	updates a user identified by '1234'
DELETE	/users	deletes all users
DELETE	/users/1234	deletes a user identified by '1234'

Figure 9: "Summary of HTTP Methods and Description of its Actions" [1].

HTTP method	Method description
GET	enables access to a resource. Whenever the client clicks a URL in the browser, the latter sends a GET request to the address specified by the URL. The GET requests are cached and can contain parameters.
POST	is used to create resources. Multiple invocations of the POST requests can create multiple resources.
PUT	is used to update resources. Multiple invocations of the PUT requests should produce the same results by updating the resource. The PUT requests should invalidate the cache entry if it exists.
POST versus PUT	the difference between PUT and POST methods concerns URI Request. The URI identified by POST defines the entity that handles the POST request. The URI in the PUT request includes the entity in the request. PUT and POST can both be used to create or update resources. The usage of the corresponding method depends on the idempotence behavior expected from the method as well as the location of the resource to identify it.
DELETE	is used to delete resources. The delete resource disappears and re-calling the same method multiple times does not change the outcome.

Figure 10: Summary of HTTP methods [1].

Stage 3: "Identifying the different resource representations. RESTful resources need to be serialized into a format that can be easily seen by the client. The most widely used resource representations are plain text, XML, JSON, and HTML. A resource can give a client representation based on what the client can manage" [1]. A customer can designate the languages and media formats that they prefer.

Stage 4: "Implementation of RESTful services using Slim Framework and authentication using HMAC mechanism" [1]. Programmers may protect REST API using a variety of methods or protocols, each with unique benefits and drawbacks. The authenticity and authorization status of a particular user is not stored by the server. The request from the client must have all the necessary information. One of the best methods for safeguarding a REST API is the authentication process that uses the HMAC technology. HMAC uses a wide range of parameters (Figure 10). HMAC is generally based on a secret key which is never sent directly at the moment of authentication.

B	Block size (in bytes) of the input to the Approved hash function.
H	An Approved hash function
ipad	Inner pad; the byte x'36' repeated B times.
K	Secret key shared between the originator and the intended receiver(s).
K_0	The key K after any necessary pre-processing to form a B byte key
L	Block size (in bytes) of the output of the Approved hash function.
opad	Outer pad; the byte x'5c' repeated B times.
text	The data on which the HMAC is calculated; text does not include the padded key. The length of text is n bits, where 0 < n < 2B - 8B.
x 'N'	Hexadecimal notation, where each symbol in the string 'N' represents 4 binary bits.
$\|\|$	Concatenation.
$\oplus$	Exclusive-Or operation.

Figure 11: HMAC Parameters [1].

An excellent way to guarantee data integrity and authenticity for online services is through the use of the HMAC method. For message integrity and authenticity, HMAC generates a code using a secret key and a cryptographic hash function as its identifier. An HMAC is always created when the secret key and the message content are transmitted when a client initiates a request. Using the same secret key, both HMACs are recalculated over the received message and compared to the received one [7]. One technique to ensure the message is genuine and unaltered is to check if it corresponds with that. This procedure will detect any changes made mid-transmission of the message. This ensures that no data is altered without authorization, so the data that is received is accurate.

Solution

HMAC provides a wide range of solutions through a secure authentication code. "The key steps involve: Pre-processing the Key, Padding and XOR Operations, and Hashing Iterations" [1, para. 6].

Pre-processing the Key: Certain pre-processing steps will ideally make the key compliant with the selected hash function before it can

be utilized in the HMAC. In order to make the key's size or format consistent with the needs of the hash function, the overall process of key pre-processing frequently entails changing the key by adding zeros or performing other operations. This stage guarantees that the key and the other HMAC processes operate in unison [Figure 11].

Padding and XOR Operations: The secret that has to be authenticated is padded to a block size that works with the chosen hash algorithm. In general, padding is the process of adding bits or characters to an input message such that its overall length satisfies the requirements for performing consistent block-sized cryptographic operations on it. Lastly, a few XORs happen between the padded message blocks and the secret key. When comparable bits in the key and message match, XORing produces a new value based on their inequality, which generates a sequence of intermediate values.

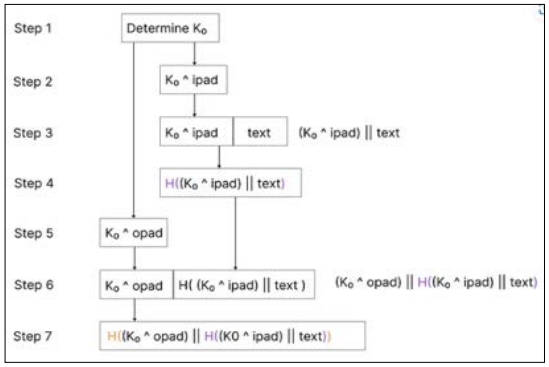

Figure 12: Diagram of HMAC

Hashing Iterations: The chosen hash function is then repeatedly fed the XOR'ed intermediate values from the previous phase. The intermediates are subjected to the hash function several times during these rounds, and the results from one iteration impact the next. Through this repetition, a series of hashed values is produced that are utilized as an authentication code (HMAC). Figure 12 below depicts the HMAC authorization.

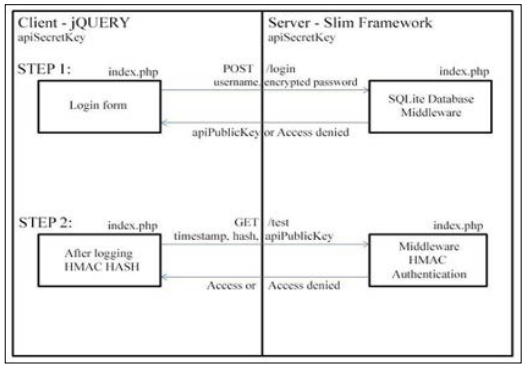

Figure 13: HMAC Authorization [1].

A MAC is produced and this is the result of combining the input data with a secret key through a set of operations and iterations defined by the HMAC algorithm. The MAC accompanies the transmitted data. The receiver on the other end executes the MAC algorithm using the shared key and the received data. This way, the sender can verify whether the data is actually being transmitted as such without alteration; they verify its authenticity by matching the recalculated MAC against the one that was sent. Any discrepancy in the calculated and received MAC indicates possible alteration or tampering of data. "The formula below [Figure 13] is used to compute a MAC over the data 'text' using the HMAC function" [1].

$$MAC(text) = HMAC(K, text) = H((K0 \oplus opad) \,||\, H((K0 \oplus ipad) \,||\, text))$$

Figure 14: "How to compute a MAC over the data 'text' using the HMAC function" [1].

Impact

HMAC provides a robust sender identification approach from an authentication standpoint. Since only the communicating peers share the secret key, every request with a valid HMAC does ensure a trustworthy sender. When standard password-based authentication would be inadequate or open to attack, this may be very helpful. Web services may avoid replay attacks by using HMAC as depicted by Figure 14. In a replay attack, an attacker transmits a legitimate message again in an attempt to obtain unauthorized access. By incorporating this safeguard into a timestamp or a number that is only used once in the HMAC computation, every request becomes distinct and is hence non-reusable. As a result, HMAC not only verifies the sender's identity but also guarantees message freshness, preventing an adversary from intercepting and replaying the message.

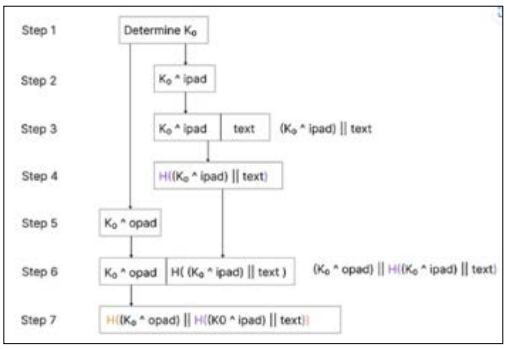

Figure 15: Diagram of HMAC

HMAC is necessary for authorization since it makes sure that unauthorized users cannot access certain resources [8]. Through the use of HMAC, the server would make sure that the request was authentic and unaltered. This depends on the endpoint, which may have varying levels of control, particularly when it comes to API security. HMAC can offer an extra degree of protection against the usage of API keys to guarantee that particular sensitive endpoints are only accessed by clients who have the required credentials. HMAC may also be used in conjunction with other security protocols, like OAuth, to provide a layered permission system. Furthermore, the HMAC system Because FFE offers a dependable method for ensuring message authenticity and integrity in transmitted and received communications, enhancing the security of Web services by limiting access to the protected resources to authorized users only.

Uses

HMAC utilizes a variety of applications in its operation. The company first selects a cryptographic hash function, such SHA-256 or SHA-3 [Figure 15]. Then they figure out a method wherein two people who are about to start talking to one another will exchange a secret key. It is important to keep this key private since sharing it might allow unauthorized people access. The sender sends after sharing; the message is hashed with the key to form the HMAC. The message will be forwarded to the recipient with the generated HMAC code attached. Using the same hash function and secret key, the receiver computes a new HMAC after extracting the HMAC code from the received message. The message is genuine if the computed and received HMACs match.

```
SHA-256('hello') = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
SHA-384('hello') = 59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90125a3c
SHA-512('hello') = 9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3
```

Figure 16: Examples of SHA-3 [3].

Applications for HMAC are widely used in security protocols, including IPsec, TLS, and JSON Web Tokens. JSON Web Tokens has three main elements (Figure 16). The fact that HMAC offers data integrity and authenticity without the need for intricate public key infrastructures is one of its main advantages. HMAC prevents tampering and fraud by utilizing a shared secret key to guarantee that only those holding the key may produce or validate the HMAC code. HMAC has good performance and may be used with any cryptographic hash function, making it flexible enough to meet a variety of security requirements. HMAC is an extremely reliable technique that, in general, encrypts data transfer and handles message authenticity verification in a variety of applications.

Figure 17: Three main Elements of JSON Web Tokens [3].

One widely utilized method for authenticity and integrity is the HMAC mechanism. Practitioners base the HMAC cryptographic approach on the cryptographic hash function. With this method, a single code would be created for each unique message. Both the source and the destination of this HMAC scheme need to have a secret key. The procedure starts with the sender sending the message together with the secret key and running it through a hash function that returns an HMAC code. The recipient creates their own HMAC code by passing the same secret key through the same hash function after receiving the message. That code verifies the message if it matches the sender's code, guaranteeing its authenticity.

Conclusion

A flaw with a lot of online services that need authentication is that they just include the request parameters, which are the user's username and password. While it does not encrypt the password, basic authentication masks it. There is a better method: digitally signing service requests with a secret key while employing a message authentication code (here, an HMAC). HMAC assigns a public and secret key to the server and each client. Only that server/client combination is aware of the secret key, although the public key is well known. After receiving the request, the server creates a new, distinct HMAC. When the two HMACs are equivalent, the server verifies the client's identity and sends the request.

There are two significant benefits. The first is that the password, or secret key, may be verified using the HMAC without users activity. "The second is that the request's fundamental integrity is likewise confirmed by the HMAC" [7, 9]. This means that a totally random key is considerably superior to a collection of characters as every bit is created at random. The block size and the key's optimal size are the same. The hash function of the key is utilized if it is too lengthy. Block size is the length of the hash output in any case. Compared to other popular techniques for authentication and authorization, HMAC offers a different approach that is simpler to put into practice.

References

1. Nowakowski G (2016) Rest API safety assurance by means of HMAC mechanism. Department of Automatic Control and Information Technology, Cracow University of Technology (PK) 5: 358-369.
2. Khan MA, Salah K (2018) IoT security: Review, blockchain solutions, and open challenges," Future Generation Computer Systems 82: 395-411.
3. Cryto-book (2019) Secure Hash Algorithms | Practical Cryptography for Developers. 2019, https://cryptobook.nakov.com/cryptographic-hash-functions/secure-hash-algorithms.
4. Scripting CS (2021) The Open Web Application Security Project (OWASP). https://owasp. org/www-community/attacks/xss/.
5. Li V (2022) API Security 101: Injection - ShiftLeft blog. Available: https://blog.shiftleft.io/api-security-101-injection-a7feea1d4fd.
6. Nottingham M (2019) Well-Known uniform Resource Identifiers (URIs). https://www.rfc-editor.org/info/rfc8615.
7. Peng F, Jiang WY, Qi Y, Lin ZX, Long M (2020) Separable robust reversible watermarking in encrypted 2D vector graphics. IEEE Transactions on Circuits and Systems for Video Technology 30: 2391-2405.
8. Zhou T, Zhu Y, Jing N, Nan T, Li W, et al. (2020) Reliable SoC Design and Implementation of SHA-3-HMAC Algorithm with Attack Protection. https://ieeexplore.ieee.org/document/9265950.
9. Dalimunthe S, Reza J, Marzuki A (2022) Model for Storing Tokens in Local Storage (Cookies) Using JSON Web Token (JWT) with HMAC (Hash-based Message Authentication Code) in E-Learning Systems. Journal of Applied Engineering and Technological Science (JAETS) 3: 149-155.