NOVEL HYBRID API SECURITYASSURANCE ALGORITHM

Jasdeep Singh
FULL STACK SOFTWARE DEVELOPER
Chandigarh
jasdeep.chauhan@hotmail.com

Alok Srivastava
SCIENTIST-E (NIC HARYANA)
Chandigarh
alok.s@nic.in

***Abstract*: Cryptographic Encryption is a method, for the protection of useful information so that only those for whom it is intended can read and process it. Numerous applications are there which require the rapid and strong security against the unauthorized users. For example, securing Military related information, securing sensitive online transactions, securing online transmission of data for real time applications like stock market apps, electronic mails or data transmission of social applications and online personal photograph albums like applications demand for the high security as these are stored and transmitted throughout the internet. The application program interface also known as API is the key to many modern apps today. As it is based on connectionless model, its security is always a major discussion among programmers. This paper focuses on simple steps to secure any API, that can be send from any type of device to any type of server through internet. Index Terms: -Application program interface, API, Security, Cryptography, Symmetric keys, Hashing, Encryption, Decryption, Token based system.**

I. INTRODUCTION

APIs (application programming interfaces) are the biggest part of the web when we talk about data transmission. In 2013 there were over 10,000 APIs published by companies for open consumption 1. That is quadruple the number available in 2010. The use of API's now a days is very common. We can hardly find any application that includes mobile, desktop or store app that do not consume API for data transmission. Almost every server technology has the ability to consume API. Some of them are – asp.net, java, python etc. These server technologies run on servers like windows, Linux, Solaris etc. An Application Programming Interface (or API) is a way for web pages and/or pieces of software to communicate with each other irrespective of what hardware they are running on. An API works as a middleman, taking the request from one piece of software, and then replying with the appropriate response from the other. One example of an API, we send a request to allow us to post text on social media account such as Twitter. The Twitter API then responds by posting a status update on your social media account.

The Create API opens up the possibility to streamline a number of tasks in the day-to-day running of your business by allowing for the easy creation of Apps. For instance, an App could be set up using the API to automatically transfer your shop orders into your accounting software.

With so many companies investing in this new area of business, possessing a working understanding of APIs becomes increasingly relevant to careers in the software industry. Through this paper, I started by looking at some fundamental concepts around APIs then proceeding to define what an API is, where it lives, and giving a high level picture of how one can be secured to ensure data safety.

Websites are designed to cater to people's strengths. Humans have an incredible ability to take visual information, combine it with our experiences to derive meaning, and then act on that meaning. It's why you can look at a form on a website and know that the little box with the phrase "First Name" above it means you are supposed to type in the word you use to informally identify yourself.

Yet, what happens when you face a very time-intensive task, like copying the contact info for a thousand customers from one site to another? You would love to delegate this work to a computer so it can be done quickly, safely and accurately. Unfortunately, the characteristics that make websites optimal for humans make them difficult for computers to use.

The solution is an API. An API is the tool that makes a website's data digestible for a computer. Through it, a computer can view and edit data, just like a person can by loading pages and submitting forms.

There are 3 terms that are there while any action in client server interaction as follows:

- **Server**: A powerful computer that runs an API
- **API**: The "hidden" portion of a website that is meant for computer consumption
- **Client**: A program that exchanges data with a server through an API

II. PROBLEM FORMULATION

The proposed work is basically a sequence of steps for providing the high level security to any API that may work with any method like GET, POST, PUT and DELETE etc. The whole security of proposed steps is based on the username and password of the user who logged in into the current system. After once login, a secret key is shared among client and server for the encryption and decryption process (security process).

The proposed work is based on the already created and tested cryptographic security algorithm in which the API requests are encrypted using the mathematical functions in such a way that the data, requests and response are secured though

the key, and the same one will be required for the decryption and authentication process. The key that is used in this process will be never shared over the network except for login process that too transmitted in encrypted manner. The focus is to create a flow in such a way that the information travelling over the network is not understandable and yet guarantees the delivery of the original data to the destination host without any tampering. This algorithm flow will also take care of the importance of data to decide whether to provide encryption or not. As all data travelled over the network does not need security so this algorithm gives the flexibility to developers of application to include mentioned security for some requests only (that contains sensitive information) or to all requests. This algorithm is created by considering the fact that it can be used with web, mobile or any other device type that has local storage memory.The algorithm also assumes that the server from where the device will be communicating with API has database/ storage and that already contains user username and encrypted password of the users authorized to use the system. The whole algorithm steps are created in such a way that they can be made work with new as well as existing system.

III. ALGORITHM DESIGN AIM

1. To make complete algorithm security dependent on user credentials - USERNAME and PASSWORD and independent of device hardware and network.
2. To make login process very secure (To Authenticate user with credentials and share common private key between client and server).
3. To create each and every API request secure and identical (By using unique Tokens for each request).
4. To block tampering and reuse of same requests (To avoid Man in the Middle Attack).
5. To handle parallel API requests of same user (Requests with same token without waiting for next valid token, but still unique and genuine).
6. To handle common problems like Internet issue, slow internet or token not updated at client or server side, etc. effectively.
7. Use security power of already developed and proved cryptographic hashing and encryption algorithms.
8. Can work with any existing/ new project model, client, server, programming language and database (Client Server Architecture).
9. Has flexibility to choose among already developed hashing/ encryption algorithms as per project requirement (Saves a lot of time in testing security loopholes as we are aware of advantages and disadvantages of already developed algorithms).
10. To provide secure and easy adaptable options to application – logout/ login validity/ password change effect on tokens/ Audit/ Currently logged in devices (Same user is allowed to login on multiple devices).
11. Flexible with tokens having any defined time validity (Token validity as per Project requirement).
12. Easy upgradation and versioning.
13. Easy to understand and implement.

IV. ALGORITHM DESIGN PREREQUISTS

1. The username and password of user is already saved in the database (any type of storage) to start the authentication process.
2. The password stored in the database must be hashed with some already developed cryptographic algorithm. For example – MD5, SHA-1, SHA-2, SHA-3 etc.
3. If username – “user1” and password – “1234” and Hash Algorithm Selected – MD5
   Then password – MD5(1234)=
   81DC9BDB52D04DC20036DBD8313ED055
4. The database can have two tables – USER_MASTER and TOKEN_MASTER, for storing user data and token data separately. This approach gives flexibility to allow same user have multiple tokens for multiple devices. (NOTE: This is a general approach, in real it can vary as per application requirement).
5. The application client must have local storage. For example – browser or android/ windows/ iPhone application.
6. The cryptographic encryption algorithm used in this paper is AES (Advance Encryption Standard) algorithm, but in real application, any encryption algorithm can be used as per application requirement.

V. ALGORITHM BELIEFS

1. Hashing Algorithms output cannot be converted back to original string.
2. Encryption Algorithms requires same secret key for encrypting of plain text and decrypting of cipher text.
3. Data transmitted over the network with API can be in JSON or plain text.
4. It depends whether to provide security to all requests or to selected requests as per application requirement.

5. User username will be saved in the form of plain text and password will be saved with after applying hash algorithm on it.

VI. ALGORITHM APPROACH

❑ LOGIN PROCESS:

- User enter his/ her username(U) and password(P) in the application. (NOTE: Password text will never travel over internet, instead only tokens will travel and too are unique for each request).
- One Machine Key(MK) is generated on the client side. It can be IMEI (International Mobile Equipment Identity) number, Long date time string till milliseconds etc. (Will be secret key for encryption/ decryption algorithm).
- One Random number(R). It can be up to 20 digits and it will be transferred over network during user authentication.
- New Password hash(CSP) will be calculated at client side by formula – HASH (R + HASH(P)) = New Hashed Password will be transferred.
- Machine Key will be encrypted(MKE) – AES(MK) with secret key – (U + HASH(P)).
- Machine key hash will also be calculated to cross check integrity. (MKH) = HASH(MKE).
- Server Side will already have Username(U) and HASH(P).
- Information transferred over network – Username, New Hashed Password, Encrypted Machine Key, MKH.
- Client will save Machine Key(MK) and username(U) in local secure storage.

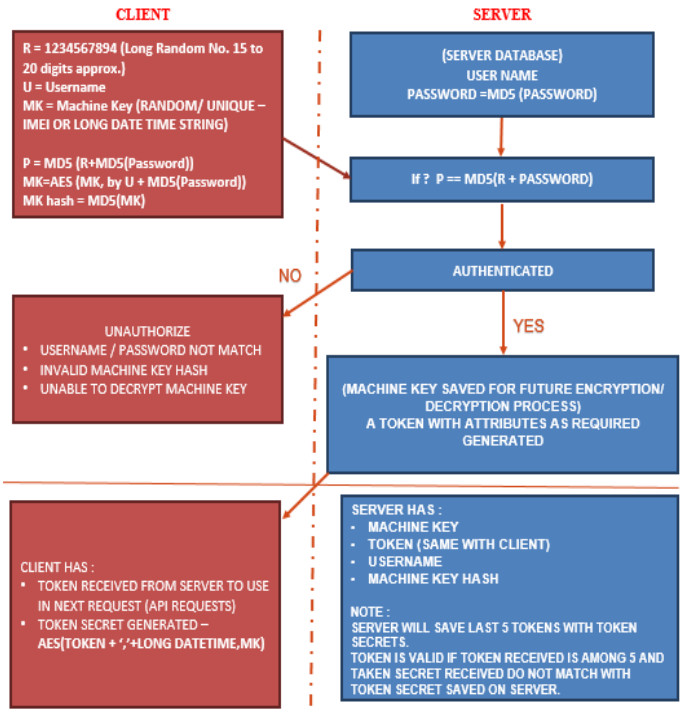
❑ AUTHENTICATION PROCESS:

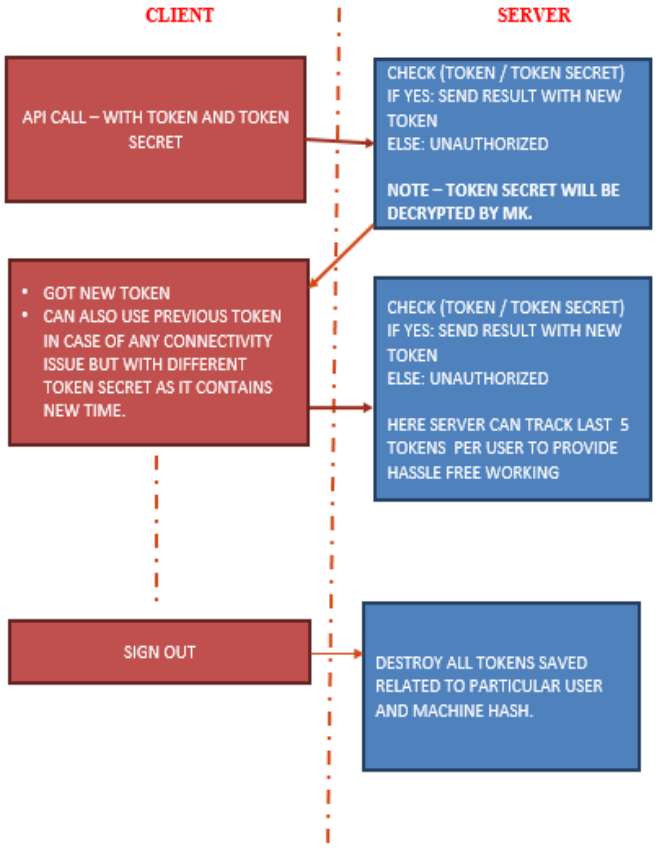
- As server already contains Username(SSU) and Hash of password(SSP) – So authentication will be done as follows:
- If (SSU = U and HASH(R+SSP) = CSP and HASH(MKE) = MKH and AES-DECRYPT(MKE) with (SSU + SSP)) then Authenticated Else Unauthorized.
- Server will now have Machine Key (MK), Username (U) to identify client device and username for further requests.
- Server will generate one token with attributes like time validity, "Token Secret" received from client (by default null) and send it to client in response, if user is authenticated.
- Server will have tendency to save last 5 tokens per user per Machine Key(MK) to authenticate future API requests and handle problems like slow internet, connectivity issue, token not able to reach genuine client, tampering of tokens.

❑ API PROCESS:

- Client will send API request with Token received from server and Token Secret – AES (TOKEN + ','+Long Date Time, MK).
- Token Secret is attached to avoid tampering and also allowing client to use tokens among last 5 in case of parallel requests or some connectivity issue.
- When Token and Token Secret reached server then server will decrypt the Token Secret with MK and authenticate process by following:
- IF (TOKEN from client = Token in last 5 saved at server and token secret linked to token is not same as that of token secret received from client and able to decrypt token secret then
  authorized else unauthorized.
- Server will then create new token and sent it to client for next request with data.
- In case client use previous token then server will send latest token to client in response with data.

VII. ALGORITHM WORKING FLOW





VIII. ALGORITHM SECURITY PROOF

❑ LOGIN PROCESS:

- Above mentioned algorithm is secured and safe because all security is dependent on username and password only which to known to individual user only.
- Password is never exposed in plain text. Password is hashed so cannot be converted back to plain text.
- Password is dependent on Random Number, so Password hash cannot be same for same user with same device or same user with different device or for different users.
- Machine Key is saved on client and Server only and never transmitted over network.
- Machine Key is dependent on username, password and random number so can never be judged by any pattern.

❑ REQUESTS UNIQUENESS:

- Request contains TOKEN and TOKEN SECRET – AES (TOKEN + ','+ LONG DATETIME (unique long string)), so no two requests either for same user or different users can be same.
- Even if there are parallel requests with same token, token secret for them will be different and that makes each request unique.

❑ SERVER SIDE TOKEN VERIFICATION:

- Server will maintain last 5 tokens to handle parallel requests for same user and to handle common network problems like slow internet or connection lost etc.
- If last (5^{th} or latest) token received by server then new token is generated and sent to client in response for the next request.
- If old token is there (among last 5 saved except 5^{th} token) in API request, then token secret linked to previous token is updated and latest (5^{th} token) is sent in response to client.
- For requests with old token, Token secret must be different from last used to confirm unique and genuine request from authorized user.

❑ OVER ALL SECURITY:

- Dependent on user username and password only

.

- Hashing and Encryption algorithm used.
- Unique Token Secret for each request, for both requests with same token or with different tokens.

IX. TOKEN MANAGEMENT

Every request to API is managed by a unique token. In response to every call to API, client will receive a unique token for the very next request. This will ensure the uniqueness of API call from same user or for different user.
To avoid reusability of same token, “token secret” is also sent in every request. This secret is produced by formula – AES(TOKEN+” |||||”+CurrentLongDateTime) by Machine key (which was shared secretly at the login time). When request is received at server, this token is cross checked by time to avoid reuse of the same token.
Above mentioned point will ensure the Authentication of user securely, passing sensitive information to and from server to client effectively, avoiding reusability of same API call, avoid any tampering of parameters to call API out of the user role (Role access is managed at server side by any mechanism).

CONCLUSION

In this paper I implemented Novel Hybrid API security assurance algorithm by applying cryptographic and logical functions on the REST API requests. This paper focuses on using already built and test cryptographic algorithms and using them to secure client to server and server to client requests without compromising the common secret key and user password. The decrypted data in API request is same as that of original data without any loss of information. The security algorithm mentioned uses MD5 hashing algorithm and AES for encryption, but applications developers can use any algorithm they wish as per application need in terms of speed, security, implementation and resources used. The another advantage of this algorithm is the whole security depends on one key, which is easy to manage and is never transmitted over the network.

I. REFERENCES

1. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. and L. Stewart, HTTP Authentication: Basic and Digest Access Authentication, Internet RFC 2617, June 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2617.txt
2. T. Dierks, C. Allen, The TLS Protocol Version 1.0; Internet RFC 2246, January 1999, URL: http://tools.ietf.org/html/rfc2246
3. "HTTP State Management Mechanism"; D. Kristol, L. Montulli, October 2000, Internet RFC 2965, URL: http://tools.ietf.org/html/rfc2965
4. "Architectural Styles and the Design of Network-based Software Architectures", Roy Fielding, Ph.D. thesis. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
5. Ryan Tomayko, "How I explained REST to My Wife", Referenced 2008-11-18, URL: http://tomayko.com/articles/2004/12/12/rest-to-my-wife

Stefan Tilkov, "A brief introduction to REST", Referenced 2008-11-18, URL: http://www.infoq.com/articles/rest-introduction

7. Leonard Richardson, "RESTful Web Services", Referenced 2008-11-18, URL: http://www.amazon.com/RESTful-Web-Services-Leonard-Richardson/dp/0596529260
8. T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. Internet RFC 2396, Aug. 1998, URL: http://tools.ietf.org/html/rfc2396
9. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "HTTP 1.1 Protocol", Internet RFC 2616, URL: http://www.w3.org/Protocols/rfc2616/rfc2616.html)
10. D. Wessels, K. Claffy, "Internet Cache Protocol (ICP), version 2"; Internet RFC 2186, Sept 1997, URL: http://tools.ietf.org/html/rfc2186
11. D. Wessels, K. Claffy, "Application of Internet Cache Protocol (ICP), version 2", Sept 1997, Internet RFC 2187, URL: http://tools.ietf.org/html/rfc2187
12. P. Vixie, D. Wessels, "Hyper Text Caching Protocol (HTCP/0.0)", January 2000, Internet RFC 2756, URL: http://tools.ietf.org/html/rfc2756
13. Apache HTTP server mod_cache module documentation; Referenced 2008-11-18. URL: http://httpd.apache.org/docs/2.2/caching.html
14. B. Ramsdell, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", Internet RFC 3851, July 2004, URL: http://www.ietf.org/rfc/rfc3851.txt
15. R. Housley, "Cryptographic Message Syntax (CMS)", Internet RFC 3852, July 2004, URL: http://www.ietf.org/rfc/rfc3852.txt
16. Mark Atwood & al., “OAuth Core 1.0"; URL: http://oauth.net/core/1.0/, December 4, 2007

II. AUTHOR’s PROFILE

Jasdeep Singh Chauhan. Born in 1991 Received Bachelor of Technology degree in 2013 from Lovely Professional

University, chehru Punjab, India and Master of Technology degree in 2016 from I. K. Gujral Punjab Technical University, Punjab, India. Currently working as a Full Stack Software developer in Mohali, Punjab, India.