A TECHNICAL INSIGHT INTO THE CONCEPTS AND TERMINOLOGIES BEHIND OAUTH – AN OPEN STANDARD FOR AUTHORIZATION

Lija Mohan[1], Sudheep Elayidom M[2]

[1]*Research Scholar, School of Engineering, Cochin University of Science & Technology, Kerala, India*
[2]*Associate Professor, School of Engineering, Cochin University of Science & Technology, Kerala, India*

Abstract

As the world wide web matures, more and more sites rely on distributed services and cloud computing for a better scalability and efficiency to meet their enhanced needs. Some examples are: a printer printing the Flickr photos, a Facebook like social network using your Google account to find friends, or any third-party programs utilizing APIs from multiple websites. The problem is, in order for these external applications to access user data from other sites, they ask for your usernames and passwords. Not only does this require exposing your secure credentials to non trustable sources ; but also provides these application unlimited access to access your account as they wish. If they get this credentials then they have unlimited access to your account and at the worst, they can change your passwords and lock your access as well. Often the same passwords may be used for online banking and other secure transactions. OAuth is an Open Standard to allow users to grant a third-party access to their resources without sharing their actual passwords. It also provides a way to grant limited access to resources with respect to scope, duration, location etc.

Keywords: *OAuth, Delegated Access, Open Standard*

----------***----------

1. INTRODUCTION

The term **OAuth [2]** implies '**O**pen standard to **AUTH**orization'. Utlimate aim of OAuth is to provide 'secure and fine grained access' to third party resources on behalf of the owner of the resources and this is achieved without sharing the actual credentials.

1.1 Introducing OAuth using a Simple Real Life Analogy

Imagine some real life scenarios.
Scenario 1: Your apartment has different keys to open different doors including the main door and lockers. It will be very difficult to keep these many keys safe.

Scenario 2: To make key management simple, you designed a 'Master Key' such that, this single key could open each and every rooms and lockers in your apartment. The advantage of such a master key design is that it will make the 'key management' easy. You need to keep only a single key to open every lock.

Scenario 3: The down side of single key (scenario 2) is that if you need to leave your key with a maid for cleaning the apartment she will be able to access every rooms as well as lockers. But you do not want your maid to access your lockers. So as an alternative, you designed a new key called 'Delegated Key' with which your maid can open all rooms in your apartment but not lockers. Here you can keep the Master key for yourself and leave the delegated key to the maid.

Scenario 3 will result in easy key management and it will serve the security purpose as well.

1.2 Extending OAuth to Meet Real World Applications

Consider you are applying for a job in a job-portal named 'JOBS.com'. You need Jobs.com to access your academic details alone from your google account. But if you share your actual google credentials, then jobs.com will be able to access every bit of information of you, stored in google, say your mails, personal data etc. Here OAuth comes to rescue. Here instead of sharing the actual credentials, you could authenticate directly with google and give permission to jobs.com to access a part of your stored information.

OAuth as a framework will provide a method for users to grant third-party access to their resources without sharing their actual credentials. It also provides a way to grant limited access to resources in scope, duration, location, etc.

If we consider the traditional client server model, there are only two user roles: The Server and Client. If client need to access some server functionality, then it should provide its credentials to authenticate itself and if the client is authentic then the service is provided by the server. But in OAuth we consider the extended needs of users in distributed environment and hence apart from client and server there come one more role i.e. Resource Owner.

If Facebook needs to access your google address book to find your friends then the server role is for Google, Facebook comes in Client role and Resource Owner is you yourself. i.e. a client acts on behalf of Resource Owner to access the resources owned by the Resource Owner.

Instead of sharing the actual credentials, Resource Owner issues a token and a shared secret to Client to temporarily access the resources. The tokens are issued with limited scope and duration and can be revoked at any time.

1.3 Development of OAuth

The OAuth started basically as an extension to OpenID. Blan Cook investigated a technique to extend OpenID such that Twitter APIs can access other resources without sharing their actual user names and passwords. This happened in November 2006 and later Chris Messina also joined his team. In 2009 Twitter released a delegated access control solution called 'Sign-in with Twitter' with the help of OAuth. Then the standard was submitted to IETF and a group was formed to develop and manage the OAuth principles. Thus it evolved as a web authorization standard.

2. FAMILIARIZING OAUTH TERMINOLOGIES

We have already discussed the 3 roles defined for OAuth in section 1. They are the client, server, and resource owner. These three roles will be present in any OAuth transaction; in some scenarios the client and resource owner can be the same entity.

In the basic client-server authentication model, the client uses **its** secret user name and password to access **its** resources hosted on the server. But Server does'nt check the authenticity of these credentials. If some other user stole the credentials and uses it on behalf of actual client this is not identified in existing scenario.

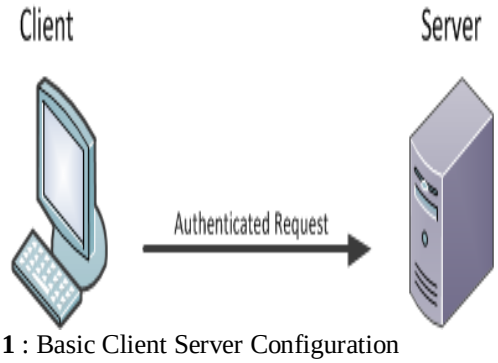


Fig 1 : Basic Client Server Configuration

In OAuth standard, a third actor or any other client can use the Owner's resources provided, the client is authorized to use those resources. Here there is no need for the client to provide his credentials but resource owner can share with him a temporary credential which enables him to access the resources.

Thus the advantages of OAuth are:

i. There is no need for resource owner to share his actual credentials with a third party.
ii. Access can be given for a particular duration, scope, etc.
iii. Access can be revoked easily.

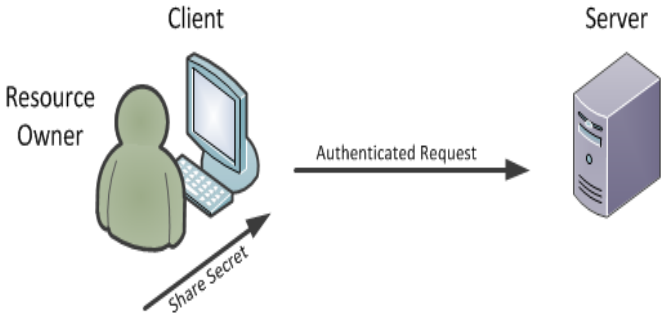


Fig 2: OAuth Terminologies

If we consider a web-based application, the role of client can be split to two entities viz. a front end application running on user side (within web browser) and a back end program running on client's server. Here Resource owner interacts with front end application where as server receives and process requests from back end program. But according to the OAuth principle, the scenario is much simplified as we can consider client as a single entity.

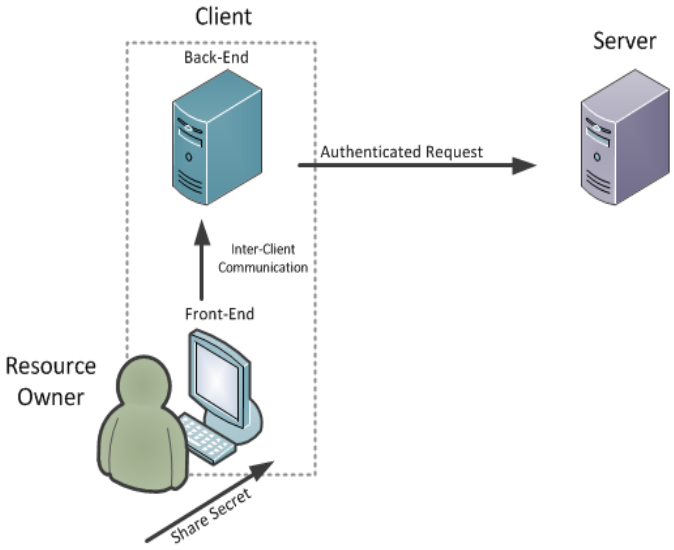


Fig 3: Different roles in OAuth

2.1 Protected Resources

A Resource owner stores some resources on server which can be accessed by third party users. But to access the resources the users should be correctly authorized. Server enforces access to resources strictly based on authorization. Such resources are called protected resources. The protected resources can be a document, audio files, video files, photos, posts, services, etc. Even if OAuth can be used with any transport protocol, it can be used for only http/https resources.

2.2 2-Legged, 3-Legged, n-Legged

Number of parties involved in OAuth implementation is specified as 'legs'. 3 legged OAuth means there will be 3 roles defined; the client, server and the resource owner. A 2 legged OAuth implementation means there will be only 2 roles client and server (here resource owner and client will be same entity). Additional legs imply that more users and re-delegated to provide access to protected resources.

2.3 Credentials and Tokens

OAuth make use of three types of credentials: client credentials, temporary credentials, and token credentials.

The client credentials helps to authenticate the client. Client credential helps the server to identify its clients. Based on this credential some special privileges and services can be granted. But the problem is that any user who gets this client credential could try to connect to the server as and when needed. Hence client credentials are allowed for some special applications like desktop applications.

Token credentials are issued temporarily to users by the Resource Owner on request. These are temporary in the sense that the tokens will have a limited scope and duration. Thus resource owner can grant access to his resources without sharing the original credentials. Token credentials are constructed such that the it contains an id which is a random string of letter and numbers that is hard to guess and the token should be paired with a secret key from user so that it will prevent unauthorized misuse. Even if the owner withdraws one token it will not affect other users.

Temperory credentials are used to identify authentication requests. In OAuth 1.0, one half of this temperory credential will be a symmetric shared secret which is paired between a client and server. However, OAuth supports an RSA[4]-based authentication method which uses an asymmetric client secret.

3. OAUTH PROTOCOL WORKFLOW

Here we explain the concept of OAuth and its workflow based on a real life requirement [1]:

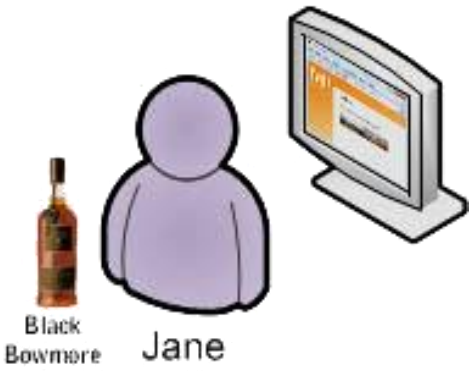


Fig: 4 A Real Life Application

Jane went for a vacation and after her return she plans to upload some of her photos to a social media photo sharing website like Faji. For that she login to her Faji account with her secret credentials like user name and password and uploads the photos. She set its accessibility as private and shares that album with only her friends.

If we compare this analogy to OAuth terminology, Jane becomes the Resource Owner, Faji becomes the Server and the photos in the album are actually the protected resources.

Fig 5: Jane uploads photos to Faji

Now Jane has made available her photos to friends online. Now she needs her grandmother who is residing at a far of place to see her photos. But since grandmother is less familiar with using computers Jane decided to sent some hardcopy of the photos to her. For that Jane preferred Beppa- an environment friendly printing service to take the print out of her photos.

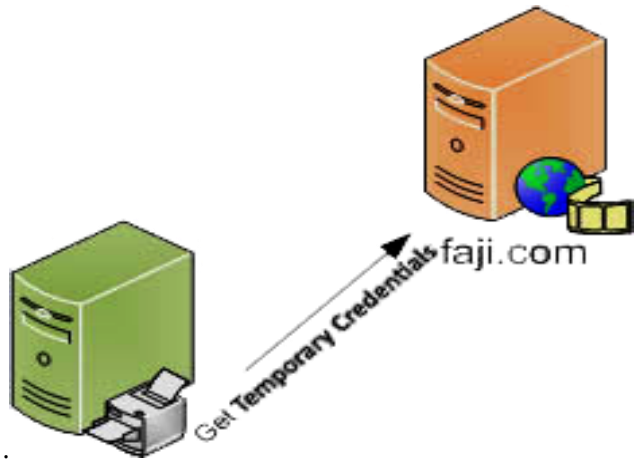
For the above mentioned scenario, Beppa acts in Clinet role of OAuth. It want to access the protected resources (photos) on behalf of the Owner, Jane. For that Beppa should gain temporary credentials from Jane.

Jane loads beppa.com on her browser and selects 'photo printing' service. Then Beppa.com asks for the photo source like Flickr, Faji, dropbox etc. Here Jane selects Faji.

Fig 6: Select photo source

According to OAuth principle, Beppa supports photo import from Flickr, Dropbox and Faji implies Beppa is a client and it has already obtained a set of client credentials that can be used to access photos from these servers.

When Jane clicks Continue, in the background Beppa requests a set of temporary credentials from Faji. Actually the temporary credentials are not resource-owner-specific; they can be used by Beppa to gain resource owner approval from Jane to access her private photos.



.

Fig 7: Beppa accessing Faji using Temperory Credentials.

Since Beppa has got temperory credentials to authorize with OAuth API of Faji, when Jane clicks continue, the Faji webpage will get loaded. But currently Beppa has not been authorized to access Jane's protected resources stored inside Faji. For that Faji will ask Jane to login to her account and ask whether she need to share her protected resources with Beppa. If she clicks yes then Faji will allow access permission to Beppa. Here the access can be restricted by Jane. For e.g. She can a grant a read only access, access for 1 hour etc.

Fig 8: Jane enters username and password to Faji

Thus OAuth allows Jane to share her photos with Beppa on behalf of Faji without revealing her secret credentials like username or password to Beppa.

Fig 9: Jane granting privileges to Beppa

Fig 10: After gaining the access, Beppa fetches photos from Faji

Thus Beppa exchanges wuthorized request token to get an access token and uses this access token to access the photos of Jane.

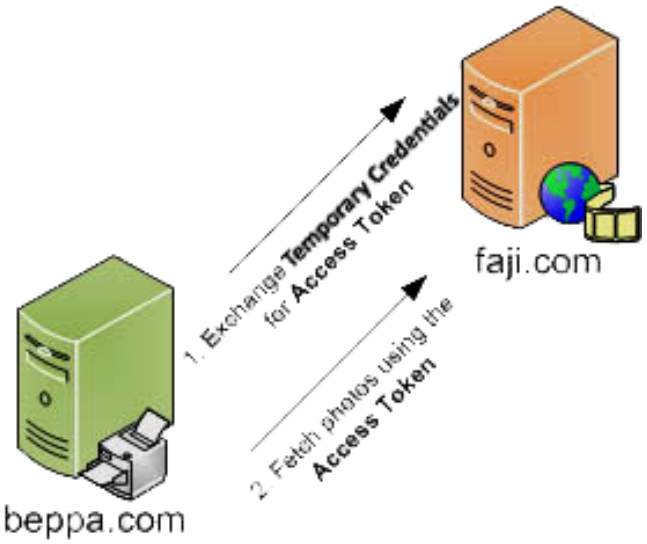


Fig 11: Access token obtained

Fig 12: Beppa fetches the photos using Access Token

3.1 Overall Oauth Workflow in a Single Diagram

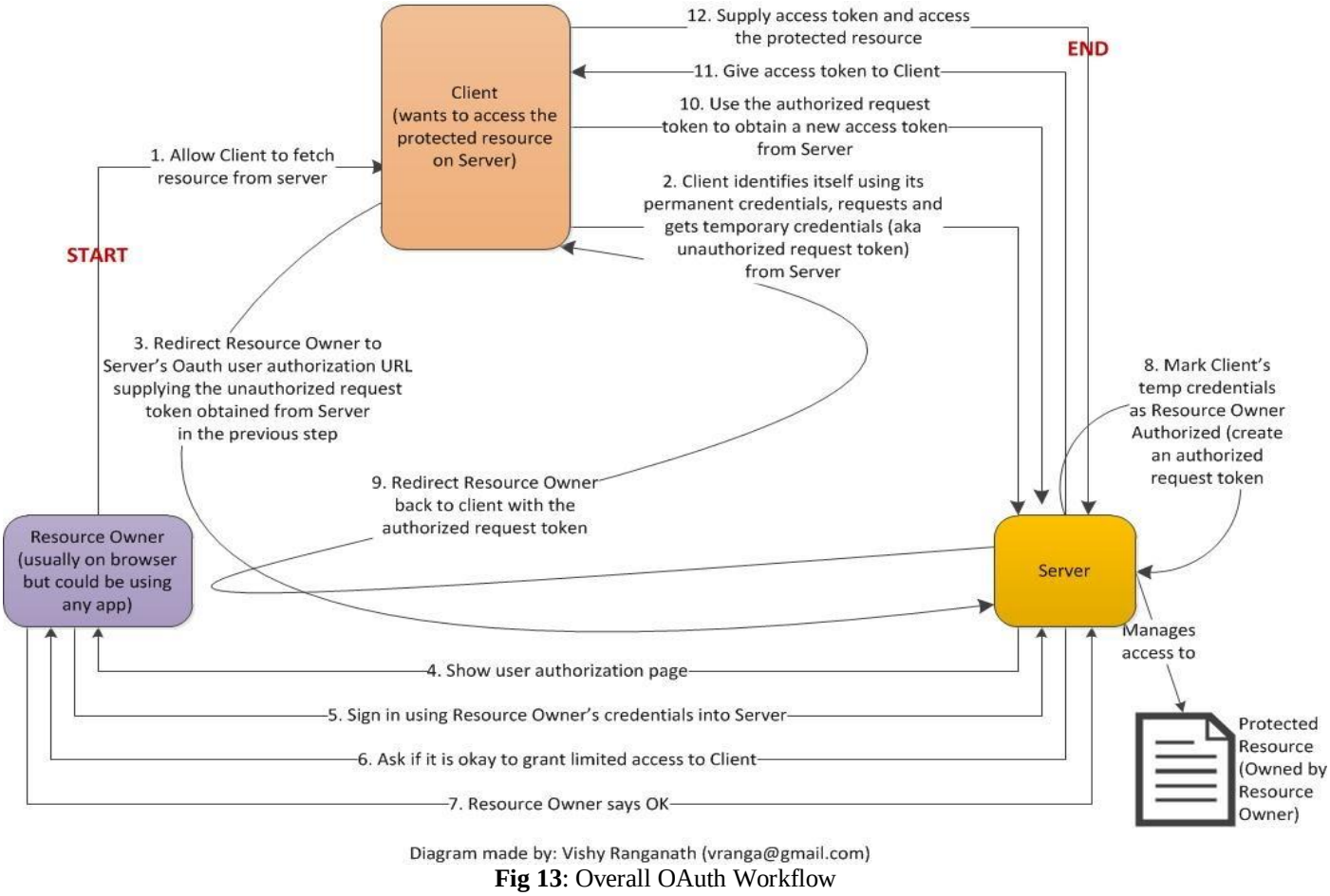


Fig 13: Overall OAuth Workflow

Fig 13 illustrates the overall workflow taking place in an OAuth architecture. A 3-legged system is considered with Client, Server and Resource Owner as 3 entities.

4. SECURITY FRAMEWORK

4.1 Beyond Basic

'Basic' is the Authentication protocol used by HTTPS. But in this protocol user name and passwords are send in an unencrypted form and hence any one listening to the network could capture the secrets. There is no mechanism to identify compromised secrets. Also another problem associated with Basic is that delegation of access control is not possible. OAuth is designed in such a way as to rectify all the problems associated with BASIC. OAuth defined over HTTPS will eliminate man-in the middle[5] attack also. Without HTTPS the security is enhanced using HMAC-SHA1[6] and RSA-SHA1[7].

4.2 Credentials

Instead of using user name and password to gain access to resources, OAuth uses client identifiers and access tokens to gain access to protected resources. Delegated access to protected resources is made possible using two set of credentials: Client identifies itself using a client identifier and client secret, whereas the resource owner is identified by an access token and token secret. Access tokens are granted and assigned by a backend application and hence they are hidden to even the Resouce Owner.

4.3 Signature and Hash

OAuth uses digital signatures to verify the integrity of the message being transmitted. The basic idea of using digital signature is that, in addition to the original request, sender will send a signature value which is calculated by applying some mathematical operations over the requested data. At the other end receiver will apply the same set of mathematical operation over the requested data and check whether it matches with the signature value received. If they matches, then receiver could confirm that the requested data has not been modified while transmission.

OAuth uses digital signatures instead of sending the full credentials (specifically, passwords) with each request. Similar to the way people sign documents to indicate their agreement with a specific text, digital signatures allow the recipient to verify that the content of the request hasn't changed in transit. To do that, the sender uses a mathematical algorithm to calculate the signature of the request and includes it with the request.

Usually digital signatures are implemented using hashing algorithms. A good hashing algorithm will always be collision resistant, one-way and strictly depends on the data being hashed. But using the hashing technique we cannot check whether message comes from the intended sender. To verify that too we need to include some 'Shared Secret' concept to the hashing technique. For example apply hashing algorithm after appending the shared secret with the message or apply hashing algorithm with shared secret as the key.

4.4 Implementing Shared Secret

According to the principles of OAuth, the shared secret generated as a part of the credentials depends on the signature algorithms used. For Plaintext as well as HMAC-SHA1 the shared secret is a part of client secret and token secret. But for RSA-SHA1 it makes use of asymmetric shared secret. In RSA asymmetric crypto system there will be 2 keys ; a public key and a private key. Private key will be known to the Owner alone where as public key will be available to server. Private key is used to sign the request whereas public key is used to verify the message. Therefor the public and private key pair must match for correct verification of message. RSA crypto system is more secure compared to others because even the server is unaware of the private key of owner.

4.5 Timestamp and Nonce [10]

In OAuth technology, Digital signatures provide mechanism for ensuring the data integrity while shared secret helps to check whether the message comes from the authorized user. Even if an attacker captures the message from network [9] it is of no use to him since he cannot make any modification without knowing the shared secret. But he will be able to initiate a replay attack by continuously resending the same message which makes the server busy and thus lead to a denial of service attack.

To eliminate this possibility, OAuth uses the concept of Nonces (Number Used Once) [10] apart from signatures and shared secret. With each request the user will append a nonce value which is a unique random value generated for that particular request. When it reaches the server, the server will store this nonce value. If a message with same nonce value arrives, it is considered as a duplicate request and hence not processed. Thus solves the possibility of replay attack.

But as the number of requests increases server will require more space to store all these nonce values and comparison will also take more time. To rectify this issue, apart from nonce value, the timestamp is also appended to the request. Hence the nonces generated within a particular time span can be deleted from server.

OAuth in fact generates Nonce as a combination of a random number and timestamp.

4.6 Signature Methods

PLAINTEXT, HMAC-SHA1 and RSA-SHA1 are the three signature methods supported by OAuth among which PlainText method is preferred only over HTTPS connection.

HMAC uses symmetric key cryptography whereas RSA uses asymmetric key cryptography. Both use SHA1 as the hashing algorithm.

The signature method used should be specified in the OAuth request itself. Then only the server can correctly verify the request.

4.7 Signature Base String

As explained in the above section, both client and server should know each other about the signature methods used and different parameters. Then only the request could be processed correctly. For that the parameters passed through the url should have a common format known to the client and server. This formatted url is called Signature Base String in OAuth. A sample Signature Base String generated for different parameters are shown in figure.

```
GET /photos?size=original&file=vacation.jpg HTTP/1.1Host:
photos.example.net:80Authorization: OAuth
realm="http://photos.example.net/photos",
oauth_consumer_key="dpf43f3p2l4k3l03",
oauth_token="nnch734d00sl2jdk",
oauth_nonce="kllo9940pd9333jh",
oauth_timestamp="1191242096",
oauth_signature_method="HMAC-SHA1",  oauth_version="1.0",
oauth_signature="tR3%2BTy81lMeYAr%2FFid0kMTYa%2FWM%3D"
```

Fig 14: A Sample Oauth Query Passed Through URL

5. CONCLUSION

In this article we explained an emerging 'Open Source tool to Authentication (OAuth)' and we tried to explain OAuth as simple as possible so that any beginner can directly get familiarized with it. We initially considered some real world examples to illustrate the need of such Delegated Access Provisioning schemes and then explained some practical applications where such protocols become useful. Then the article provides some technical insight to how actually the security is implemented in OAuth. As a conclusion we can say that OAuth is an **open protocol** to allow **secure authorization** in a **simple, elegant** and **standard** method from web, mobile and desktop applications. Today many of the social media websites like Google, Facebook, Twitter etc are making use of OAuth technology to implement their delegated access control needs.

REFERENCES

[1] Credit:Some content adapted from hueniverse.com.

[2] More information is available on OAuth.net

[3] O'Reilly: OAuth 2: The Definitive Guide

[4] Rivest, R.; Shamir, A.; Adleman, L. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21 (2): 120–126. doi:10.1145/359340.359342

[5] Man in the middle Attack: http://www.veracode.com/security/man-middle-attack

[6] "The Keyed-Hash Message Authentication Code (HMAC)" Federal Information Processing Standards Publication Category: Computer Security Subcategory: Cryptography.

[7] **Secure Hash Algorithm (SHA-1),** National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard,", U.S. Department of Commerce, May 1994.

[8] Introduction to Secret Sharing, http://www.cs.berkeley.edu/~daw/teaching/cs276-s04/22.pdf

[9] Tutorial on Network Attacks: http://backtracktutorials.com/backtrack-wireless/packet-sniffing-and-injecting/

[10] Phillip Rogaway,"Nonce-Based Symmetric Encryption", http://web.cs.ucdavis.edu/~rogaway/papers/nonce.pdf