Multiple JSON Web Tokens for Mobile Distributed Applications

Pedro Mestre, *Member, IAENG,* Rui Madureira,
Pedro Melo-Pinto, and Carlos Serodio, *Member, IAENG*

***Abstract*—Internet of (almost) everything brought to the spotlight the need for efficient, and yet secure, ways to transmit data between connected devices. In the context of securing RESTful web services to be used in (but not limited to) agricultural-related applications, authors have developed a system, based on JSON Web Tokens. The objective of that work was to develop a system able to prevent replay attacks. This objective was achieved by using multiple tokens, i.e., a system based on one-time tokens. The client, before using the service must request a token to an authentication service, however the issued token can be used only once. When the client accesses a service, the token is checked, and if it is valid the service will return, along with its results, a new token to replace the previous one. Because scalability is also one of the key objectives, a distributed token validation was implemented. Instead of generating new tokens using a centralized service, tokens are issued and verified by the (trusted) service providers. The system was tested in laboratory using virtualized Linux servers and clients. Because the objective of the authors is to use this system in the real life, in this paper are presented new tests made to the system, now using real and in production networks. Tests were made using a Linux client with a benchmarking software developed in Java and an Android client. Results show that it is feasible to implement the system in a real life scenario, however the trade-off is the greater complexity of the mobile application code, if parallel communications are needed.**

***Index Terms*—RESTful, web services, authentication, authorization, token, multiple tokens, JSON Web Token.**

I. Introduction

WITH the widespread of mobile distributed applications we need lightweight, efficient and secure methods to send data from and to mobile devices. Using the infrastructure provided by HTTP, many applications have been using RESTful web services, which are lightweight web services particularly well suited for creating APIs for clients spread out across the Internet [1]. Some examples where such web services have been used include projects related to agriculture and farm management [2],[3],[4].

Authors have presented, in [2], a work where RESTful web services are used to create remote data access services for projects also related to agriculture and wine production systems.

Those projects use a set of distributed applications and sensors that are used to collect georeferenced data from vineyards, such as multispectral images of grape bunches, photographies of vine leaves, micro-meteorological data, textual information inserted by staff (e.g. anomalies reporting), as described in [2].

In such a system only authorized principals can have access to the resources available on the servers. An authentication and authorization system is therefore needed. One very simple way of guarantying that only authorized clients access services is forcing clients to authenticate themselves using some credentials, and then use sessions to track users. As long as HTTPS is used, the system will be secure.

However, using sessions can have a negative impact on system scalability. Also, one of RESTful web services constraints is that communications between the client and the server must be stateless [5], so, as consequence sessions cannot be used. The solution presented in this paper is based on multiple one-time JSON Web Tokens, to avoid replay attacks (without the need of session tokens), and which uses a distributed token management system.

II. Managing Access to Resources Without using Sessions

Because of the scalability issues and the intrinsic proprieties of RESTful Web Services, session information cannot be used. Therefore all the information that the server will need to fulfill the client's request must be sent by the client itself, either in the request body, the URL query string or using cookies. The stateless nature of RESTful web services plays an important role in service scalability, because storing in the server session information would imply that if multiple servers were used, then those servers would have to share session related information.

Obviously that if a client sends information to the server, the server must trust that client. Otherwise how could the server trust the information that the client is providing. This means that the server must then be able to authenticate the client and check its authenticity for every HTTP transaction.

Client authentication can be made by sending the client's authentication information in every HTTP request, using

Manuscript received March 2, 2018; This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and POCI-01-0145-FEDER-006958, and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013 and UID/AGR/04033/2013, and by Integrated Research in Environment, Agro-Chain and Technology, NORTE-01-0145-FEDER-000017, in its line of research entitled VitalityWINE, co-financed by the European Regional Development Fund (ERDF) through NORTE 2020 (North Regional Operational Program 2014/2020).

P. Mestre is with Centro Algoritmi, University of Minho, 4800-058 Guimarães - Portugal, and Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, (phone: +351-259350363; email: pmestre@utad.pt)

R. Madureiria is with University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, email: rccmadureira@gmail.com)

P. Melo-Pinto is with Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, and Algoritmi Research Centre, Guimarães, Portugal (email: pmelo@utad.pt)

C. Serodio is with Centro Algoritmi, University of Minho, 4800-058 Guimarães - Portugal, and Centre for the Research and Technology of Agro-Environmental and Biological Sciences, CITAB, University of Trás-os-Montes and Alto Douro, UTAD, Quinta de Prados, 5000-801 Vila Real, Portugal, www.utad.pt, (email: cserodio@utad.pt)

Basic HTTP Authentication Scheme (RFC7617 [6]), or the HTTP Digest Access Authentication (RFC7616 [7]), for example.

In both cases the server must have access to the user credentials, stored for example in a relational Database (e.g. MySQL, PostgreSQL, etc.) or using LDAP (Lightweight Directory Access Protocol) [8], just to mention two examples.

These are very straightforward methods to send client authentication to the server, however some scalability issues arise. A very common method used to authenticate the client's credentials is to store them in a database. However the performance of such a system will depended on the database technology and on the size of the tables [9]. If for every transaction we will have to verify the user credentials in a relational database, it can be easily concluded that if the number of clients rises and consequently the number of transactions, the service performance will sooner or later start to decrease.

Another authentication method that can be used by services to authenticate clients include the use of tokens:

- The client sends its credentials to an authentication service;
- The authentication service validates the user information and, in case of success, sends a token to the client;
- When the client accesses the service, it uses this token to identify itself. If the token is valid, the service will process the client request.

Such a system has the advantage that the credentials are sent only at the beginning of the "session" or after the token has expired, not in every transaction. Therefore we reduce the load on the server.

To implement an authorization system, based on tokens, to authenticate client requests to web services we have many alternatives, including the use of simple tokens such as JSON Web Tokens (JWT) [10] or we can also use a more complex infrastructure such as OAuth 2.0 Authorization Framework [11].

The objective of this work is to develop a scalable authentication and authorization system based on tokens that allows anytime token revocation, that complies with the stateless constraint of RESTful web services and that prevents replay attacks. Because of its simplicity, instead of adapting an existing framework to the needs, authors decided to build a custom authorization system based on JWT, which was presented in [2].

III. Access to Web Services using Multiple JSON Web Tokens

JWT, defined in RFC7519 [10], is an open standard that provides a way of transmitting signed information, as a JavaScript Object Notation (JSON) object, between parties in a transaction. Some advantages of using JWT includes the fact that it is very compact, in comparison to XML based solutions such as SAML (Security Assertion Markup Language) [12], and the information included in the token can be verified and trusted because tokens can be digitally signed by the issuer.

A. JWT Format

A JSON Web Token is made of three distinct parts:

- Header – which usually has two fields: the type of the token (JWT) and the hashing algorithm;
- Payload – that contains the token reserved, public and private claims.
- Signature – used to verify if the token can be trusted or not.

A typical header for a token signed using HMAC SHA256 would be:

```
{
"alg": "HS256"
}
```

Statements about the client (claims) are sent in the payload. These can be:

- Reserved, which are predefined (in [10]) recommended claims: Expiration Time (exp), Not Before (nbf), Issued At (iat), JWT ID (jti), Issuer (iss), Audience (aud), and Subject (sub);
- Public claims that can be freely defined to best suit the application needs;
- Private claims, which are not part of Reserved or Public claims, and that a producer and consumer of a JWT may agree to use.

An example of a Payload using some of the Reserved Claims would be [2]:

```
{
"exp": 1520016178,
"user": "pmestre",
"iat": 1520016168
}
```

The above listing is a very simple example that uses two reserved claims (Issued At and Expiration Time) and one private claim ("user"). This token is valid from Friday, March 2, 2018 6:42:48 PM until Friday, March 2, 2018 6:42:58 PM.

Signature of a JWT token can be obtained either by using HMAC algorithm (as above) or a public/private key pair using RSA.

The resulting token is the concatenation of the above three parts (Header, Payload and Signature), encoded in base64 and separated by dots. As an example the above presented example token, signed using a shared key "Portugal" would look like the example below:

```
eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjAwMTYx
NzgsInVzZXIiOiJwbWVzdHJlIiwiaWF0IjoxNTIwM
DE2MTY4fQ.mlbgDBTzw-nmHa_-lkSWpvbDURezDRy
oqKa0AUn75mg
```

Please notice that line breaks were added for better visualization.

B. Multiple JWT Tokens

Once a token is issued we can use it until it expires. This means that if it has not expired (and not revoked) it can be used by the client to access services. As discussed in [2], if the token is compromised, we can have a security issue. Therefore authors presented a solution based on multiple one-time JSON Web Tokens.

So instead of using an architecture like that presented in Fig. 1, where the client first sends its credentials to the server

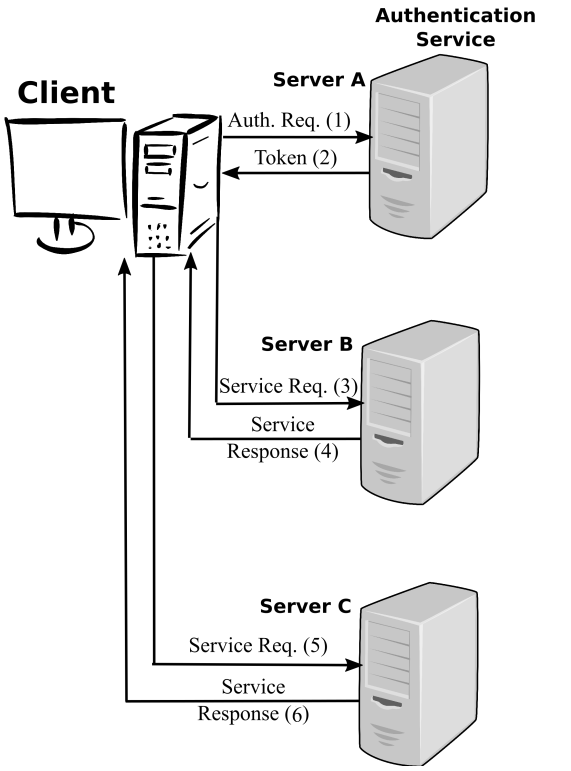

Fig. 1. Authentication and Service Access using Tokens [2].

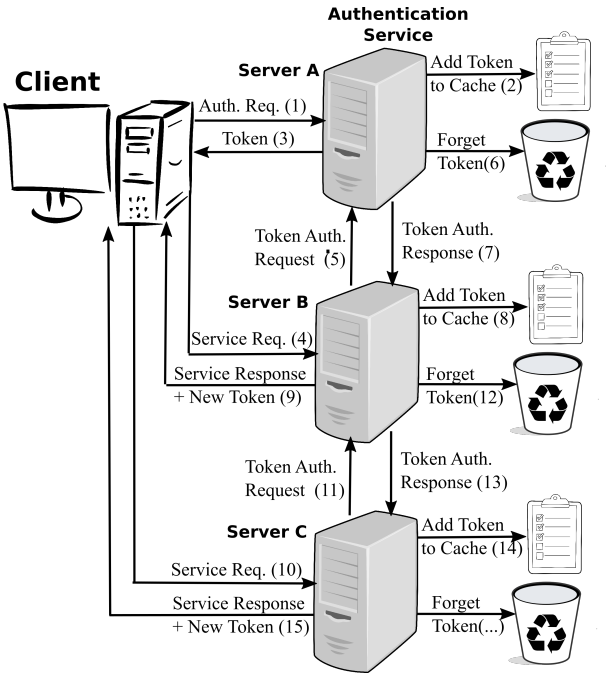

Fig. 2. Authentication and Service Access using Multiple Tokens [2].

(1), receives the token from the server (2) and then requests services (3,5) using that token, a different approach was used.

As shown in Fig. 2, in the proposed approach the client still has to send its credentials to an authentication server, which will issue a token to the client. However this token is valid only for a single transactions. After being used, the token is revoked and cannot be used in any other transaction.

Using a single server architecture this would be very easy to implement. If we have only one server then that server will know if a token, that it has previously issued, has already been used or not, and remove a token from its valid tokens list whenever a token is used.

Because the system architecture must be easily scalable, therefore a multi-server approach was be used. As depicted in Fig. 2 it works as described in [2]:

- (1) – The client sends its credentials to the Authentication service, requesting a new token to be issued;
- (2) – Credentials sent by the client are verified (e.g. using a Database) and if the principal is positively authenticated then a new token is issued. This token is stored locally in a token cache for later validation;
- (3) – The token is sent to the client;
- (4) – When the client makes a request to a server (in this example Server B) it has to send the previously received token. If the server can verify the token signature and it has not expired already it will then check which trusted server has issued the token (in this example it was Server A).
- (5) – Server B will contact the token issuer (Server A) requesting the token validation;
- (6) – If server A finds the token in its token cache (the token is valid) then the token is deleted to prevent further attempts from clients to use the same token;
- (7) – Server B is informed if the token was found in the tokens cache, i.e., if the token is still valid;
- (8) – The server will issue a new token and adds it to its token cache (for future validation);
- (9) – Service requested by the client is executed and when the server sends the response to the client it also sends the new token that the client must use in its next service request.

Whenever a client tries to access a server without a token, an expired token or an already revoked token, the server will respond with the corresponding HTTP code.

The above presented distributed token validation system is an alternative to the use of a central token management service, that would be a single point of failure in the system. Obviously that we could use a cluster of token management systems, but then we will have to deal with their synchronization. The proposed system has the service and the token management embedded in every server that makes part of the architecture.

For this token revocation procedure to work, servers must contact each other to validate tokens that they did not issued. As presented in [2], if a single server is contacted by the client, the performance loss is very low, in comparison to that of using a single token valid for all transactions. In the worst case scenario, all tokens are issued by another server, therefore all tokens must be validated by a remote server. In this case, there is a big performance loss, in comparison to the use of a single token, because the time per transaction doubles. However, has stated in [2] it is a more secure approach, that best suits the objective of the authors work, and, if we compare it to the use of Basic HTTP Authentication Scheme using a Database to authenticate every transaction, the performance is better(approximately three times faster).

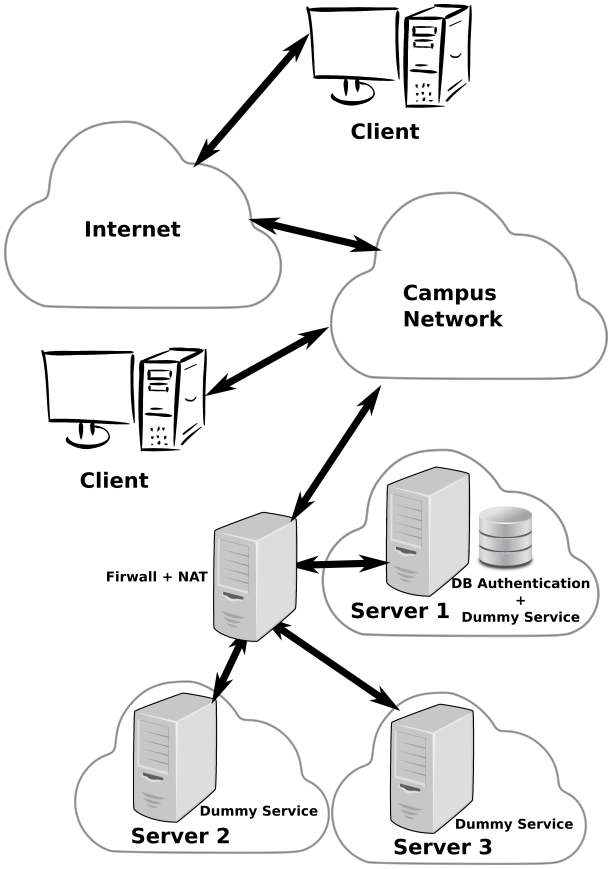

Fig. 3. Testing Scenarion.

IV. Testing Scenario

For tests it was used exactly the same servers and services that were used in [2]. However the previous tests where made in laboratory, in an controlled environment, and all clients and servers were virtual machines running in the same hardware. The main difference to the current scenario is that now services are accessed remotely, and not from a client in the same hypervisor, not even in the same Local Area Network (LAN).

As presented in Fig. 3 three servers were used in the testing scenario, one of them with a database authentication service and a test service. The other two servers only have a testing service. These testing services issue a new token if and only if they can validate the token sent by the client. The first token held by the client must be issued by Server 1 (authentication service).

These servers do not have public IP addresses, therefore are not directly connected to the the Campus network. Connecting those servers to the Campus network there is a firewall with NAT (Network Address Translation), that forwards the requests to the servers and their responses to the clients.

Clients will access the services either from the Campus network, using WiFi, and from a remote site, using an Internet Service Provider (ISP). There will not be any kind of control over the delay and available bandwidth between the clients and the servers.

Because the main objective of the software developed, in [2], to do the benchmarks is to send the most number of messages as possible per time unit, it was not developed to keep system responsiveness. This means that it cannot be used, as it is, in a real mobile application. Therefore for tests using a smart phone, it was used a new Android application that uses the Volley library [13]. Volley will be used to send requests to the servers. Notice that the objective of this paper is not to benchmark and/or analyse the performance of Volley, and should not be seen as such. Volley is used only as a tool, from the many options that could have been used to implement the testing software. Also it was used a simple (default) setup of Volley. Discussing details about how to best tune Volley for our system is out of scope of this paper.

Volley allows us to implement the application logic without having to concern about how to deal with multiple connections, queue and dequeue requests, etc. However, if we have a single token, i.e., the client made a single authentication (which seems the most logical action), we cannot have multiple parallel connections, this is the trade-off of using one-time tokens. If we had, for example, two parallel requests using the same token, the second request reaching the server would fail, because the token had already been used.

A solution is to have the client requesting multiple tokens, i.e., making multiple authentications, one per parallel connection that the application needs. These tokens can be stored in a FIFO (First In First Out) like structure, and whenever a new request is made, the oldest token is removed from the FIFO. When the response arrives to the client, the new token (that came with the response) is inserted in the FIFO. Because the number of requests in transit is limited to the number of tokens, it must also exist a queue to store requests waiting for a token to be available.

V. Tests and Numerical Results

In this section will be presented the results obtained in the tests made using the above presented testing scenario. For comparison purposes, in Table I, are presented some results previously obtained and published in [2]. As above mentioned, these results were obtained in laboratory.

In Table I, Table II, and Table III, the results are shown for 10, 20, 30, 40 and 50 (simulated) simultaneous clients, using:

- Database Authentication for every request (DB-A);
- A single Token and the service provided by a single server (ST-SS);
- Multiple Tokens and the service provided by a single server (MT-SS);
- Multiple Tokens and the service provided by two servers (MT-MS).

Times presented in all tables of this section are in ms.

TABLE I
Reference Values from [2].

# Clients	10	20	30	40	50
DB-A	1409	1,406	1,394	1,395	1,384
ST-SS	0,446	0,442	0,446	0,443	0,448
MT-SS	0,476	0,470	0,462	0,466	0,464
MT-MS	0,563	0,589	0,585	0,585	0,590

Because the objective of the present paper is to verify if the same implementation is feasible in the "real life", Table II and Table III present results obtained using clients connected to the WiFi network of the University of Trás-os-Montes and Alto Douro (UTAD), and remotely connected using an Internet Service Provider (ISP), respectively. In the first test each client made 150 requests to the server and in the latter each client made 100 requests.

Notice that for these tests it was used the same software that was developed to collect data shown in Table I. which was developed in Java, and that creates a new thread for each client that is simulates.

In the first case the real, in production, WiFi network of UTAD Campus was used. Data in the table was obtained in a regular working day with other users also using the wireless network in their activities.

Analysing the results we can see that we have a similar behaviour in laboratory and using the WiFi Network, i.e., using tokens (as expected) we have better performance than using the database authentication, and when we use multiple tokens the performance is worse than using a single token. Also, when using multiple server, i.e. the token must be validated in a remote server, the performance is worse.

However the relative performance loss is not stable in all data, because it was collected in real network with other clients using it. If we look for example (in Table II) for the case of 20 clients, the performance for a single server and two servers is very similar, however for 50 clients, using multiple servers will take the double of the time per transaction than a single server. Nevertheless, considering the performance increase, in comparison to the database authentication option and considering the security increase, for the objectives of our work the use of multiple tokens is feasible.

TABLE II
REMOTE CLIENT CONNECTED TO THE CAMPUS NETWORK.

# Clients	1	2	3	4	5	10	20	30	40	50
DB-A.	15,81	9,66	8,28	6,07	6,15	5,88	5,74	5,75	5,74	5,70
ST-SS	8,02	3,97	3,69	2,08	1,71	0,95	0,67	0,73	0,63	0,55
MT-SS	8,97	4,20	2,76	2,04	1,91	0,93	0,81	0,77	0,63	0,62
MT-MS	11,45	5,11	3,57	3,01	2,28	1,20	0,93	1,22	1,05	1,11

The same conclusions can be drawn form data in Table III. In this case data is more stable, only because of the access method to the network. In this case a very low number of clients were sharing the same WiFi network, although, the connection to the UTAD Campus network is made using a remote ISP. Also, there are 12 hops between the client and the servers.

It is noticeable the increase of the time per transaction, when the client is not connected to the Campus Network. These are more realistic data, considering that our end application is for clients that are outside our Campus.

In Table IV are presented the results using an Android Smartphone, connected to a remote ISP. Notice that unlike the previous tests, these results were obtained using a single client that made several requests (1, 2, 3, 4, 5, 10, 20, 30, 40 and 50). As above explained, the maximum number of parallel requests, using tokens, will depend on the number of tokens requested to the authentication server. Results presented in the table were obtained when the client requested 1, 2, 3, 4, 5 and 10 tokens. It is also presented data obtained using Database authentication (DB-Auth).

TABLE III
REMOTE CLIENT USING AN ISP.

# Clients	1	2	3	4	5	10	20	30	40	50
DB-A	51,92	30,00	21,92	19,21	13,98	7,06	5,91	7,10	5,90	5,85
ST-SS	43,15	20,91	14,36	10,77	8,65	4,24	2,24	2,81	2,36	1,75
MT-SS	44,10	21,87	14,64	11,10	8,87	4,33	2,25	2,91	2,19	1,83
MT-MS	43,78	22,78	15,29	11,44	8,89	4,76	2,76	3,34	2,66	2,39

For DB-Authentication, the application simply sent to the message queue all the requests at once, without concerning about how many messages could be in fact sent in parallel. Those concerns where left for the Volley library, which was used with its defaults parameters, without any tuning or any custom queue system (as above stated this is out of scope of this paper). Results for DB-Auth are very similar to those obtained when only a token is in use, i.e., a new request is sent to the server only when the response from the last one arrives. Therefore we can only compare those two results, and as it can be seen, using tokens the performance is slightly better.

Relatively to the other results, because the maximum number of messages in parallel is limited by the number of tokens, a request scheduler had to be developed. Therefore we had a scheduling policy overlapping the default queue policy of Volley. Notice that the objective of the scheduler was to send all the test request as fast as possible.

TABLE IV
MOBILE PHONE, SINGLE SERVER.

# Requests	10	20	30	40	50	100
DB-Auth	49,82	51,68	52,63	53,47	50,60	50,14
1 Token	46,36	45,75	45,81	45,48	48,47	49,20
2 Tokens	28,46	24,91	24,91	24,59	23,93	23,83
3 Tokens	17,70	16,45	15,68	16,40	15,53	16,16
4 Tokens	17,60	12,64	12,81	11,98	11,99	11,92
5 Tokens	16,31	11,42	11,35	11,72	11,92	11,80
10 Tokens	18,59	11,79	11,54	11,98	12,21	11,58

Table V presents the results obtained with the smart phone when two servers are used. Consistently with the previously obtained results, the performance was slightly worse, but the difference is very small. The network delay and the processing overhead masks the performance difference.

VI. CONCLUSION

As the main conclusion authors point out that it is feasible to implement the proposed system, using mobile devices. As a proof of concept platform Android was used. That said,

TABLE V
MOBILE PHONE, TWO SERVERS.

# Requests	10	20	30	40	50	100
DB-Auth	49,82	51,68	52,63	53,47	50,60	50,14
1 Token	43,08	47,45	47,36	48,93	49,51	49,39
2 Tokens	24,07	23,85	25,14	24,86	26,06	25,58
3 Tokens	17,56	16,98	17,36	18,92	16,06	17,01
4 Tokens	15,22	13,80	13,99	14,50	13,23	13,16
5 Tokens	15,72	14,04	13,94	12,95	12,36	12,26
10 Tokens	13,66	13,36	12,97	12,97	12,40	12,24

the increase of security for web services, by using one-time tokens, without losing much overall performance comes with the trade-off of increasing the complexity of the developed applications.

In scenarios where a single stream of data is needed, and when it is acceptable that the next request is sent only when the response of the previous one has arrived, the implementation if very straightforward. And probably any out of the shelf solution to send data can be used (Volley was used in this work).

However if more that one parallel request must be made, to increase the system performance, the software developer will have to implement a queuing system that takes into consideration the specificities of the token validation and revocation system. Besides the queuing system it is also needed a token revalidation thread that revalidates tokens that are bout to expire. Because Volley supports external queues, this is a good topic for future work.

REFERENCES

[1] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase, and W. Markito. The Java EE 7 Tutorial: Volume 2. Oracle. [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/

[2] P. Mestre, R. Madureira, P. Melo-Pinto, and C. Serodio, "Securing RESTful Web Services using Multiple JSON Web Tokens," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2017, 5-7 July, 2017, London, U.K.*, 2017, pp. 418–423.

[3] M. Arroqui, C. Mateos, C. Machado, and A. Zunino, "RESTful Web Services improve the efficiency of data transfer of a whole-farm simulator accessed by Android smartphones," *Computers and Electronics in Agriculture*, vol. 87, pp. 14 – 18, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169912001305

[4] A. Kaloxylos, A. Groumas, V. Sarris, L. Katsikas, P. Magdalinos, E. Antoniou, Z. Politopoulou, S. Wolfert, C. Brewster, R. Eigenmann, and C. M. Terol, "A cloud-based farm management system: Architecture and implementation," *Computers and Electronics in Agriculture*, vol. 100, pp. 168 – 179, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169913002846

[5] I. Porres and I. Rauf, "Modeling Behavioral RESTful Web Service Interfaces in UML," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 1598–1605. [Online]. Available: http://doi.acm.org/10.1145/1982185.1982521

[6] J. Reschke, "The 'Basic' HTTP Authentication Scheme," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7617, September 2015.

[7] R. Shekh-Yusef, D. Ahrens, and S. Bremer, "HTTP Digest Access Authentication," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7616, September 2015.

[8] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 4511, June 2006, http://www.ietf.org/rfc/rfc4511.txt. [Online]. Available: http://www.ietf.org/rfc/rfc4511.txt

[9] A. Saikia, S. Joy, D. Dolma, and R. Mary, "Comparative Performance Analysis of MySQL and SQL Server Relational Database Management Systems in Windows Environment," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 3, 2015.

[10] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7519, May 2015, http://www.ietf.org/rfc/rfc7519.txt. [Online]. Available: http://www.ietf.org/rfc/rfc7519.txt

[11] D. Hardt, "The OAuth 2.0 Authorization Framework," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 6749, October 2012, http://www.ietf.org/rfc/rfc6749.txt. [Online]. Available: http://www.ietf.org/rfc/rfc6749.txt

[12] B. Campbell, C. Mortimore, and M. Jones, "Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7522, May 2015.

[13] Transmitting Network Data Using Volley. [Online]. Available: https://developer.android.com/training/volley/index.html