Hash-Chain-Based Authentication for IoT

António Pinto[a,b] and Ricardo Costa[a]
[a]GCC, CIICESI, ESTG, Polytechnic of Porto, Portugal
[b]CRACS & INESC TEC, Porto, Portugal
apinto@inesctec.pt, rcosta@estg.ipp.pt

KEYWORD

IoT; Secure; Authentication

ABSTRACT

The number of everyday interconnected devices continues to increase and constitute the Internet of Things (IoT). Things are small computers equipped with sensors and wireless communications capabilities that are driven by energy constraints, since they use batteries and may be required to operate over long periods of time. The majority of these devices perform data collection. The collected data is stored on-line using web-services that, sometimes, operate without any special considerations regarding security and privacy. The current work proposes a modified hash-chain authentication mechanism that, with the help of a smartphone, can authenticate each interaction of the devices with a REST web-service using One Time Passwords (OTP) while using open wireless networks. Moreover, the proposed authentication mechanism adheres to the stateless, HTTP-like behavior expected of REST web-services, even allowing the caching of server authentication replies within a predefined time window. No other known web-service authentication mechanism operates in such manner.

1. Introduction

The Internet of Things (IoT) can be seen as a distributed network of devices that interact with human beings and with other devices (Xia et al., 2012; ABIresearch, 2014). New applications for such devices appear on a daily basis and these, typically, use sensors to collect data. The IoT is expected to become a key source of big data and analytics (Press, 2014). Example sensors are accelerometers, gyroscopes, magnetometers, barometric pressure sensors, ambient temperature sensors, heart rate monitors, skin temperature sensors, GPS, video cameras, microphones, among others.

A possible classification of IoT devices can be done with respect to their communication capabilities. The adopted reference scenario is depicted in Figure 1 and comprises three types of sensors. Type A sensors are characterized by requiring a specific Wireless Sensor Network (WSN) gateway, typically from the same manufacturer of the devices, and by being built for ultra low power operation. These run on (coin shaped) batteries and minimize wireless communications in order to expand their lifetime. The security, authentication and confidentiality of the collected data is achieved by means of pre-built, per device, cryptographic encryption keys that are exchanged with the WSN gateway upon initial set-up. Type B are characterized by being more autonomous, not requiring a gateway, and by being able to interact directly with the on-line central server. These are either connected to a power outlet or run on batteries with larger capacity, which are also recharged more frequently (daily or more). The security, authentication and confidentiality of the collected data can be achieved by any available mechanism. Type C are characterized by requiring a type B device in order to upload the collected data to the on-line central server. These use short range wireless communication capabilities, such as Bluetooth, to communicate with a type B device that has standard IP connectivity. The security, authentication and confidentiality of the collected data can be achieved by any mechanism available in the type B device.

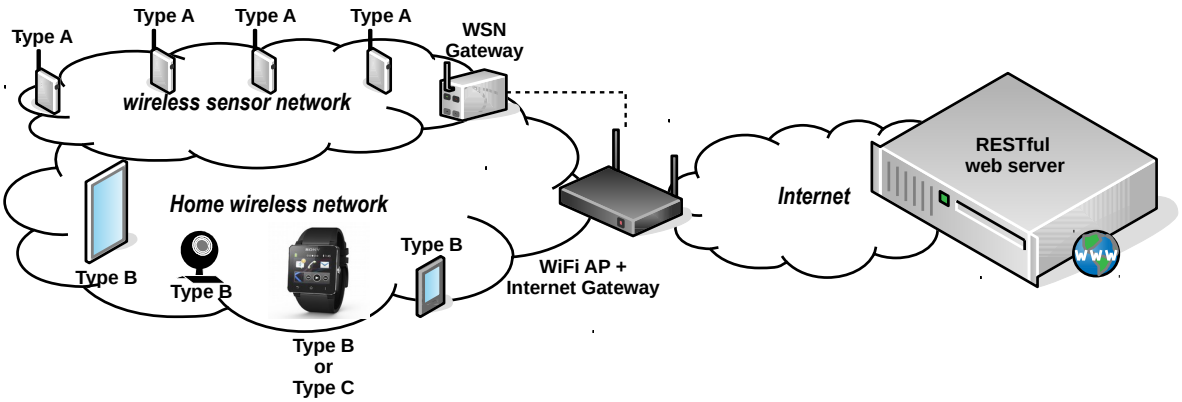

Figure 1: Adopted reference scenario.

The data collected by all these sensors tend to be personal, sensible and private. The privacy and control over the collected data was already addressed by the authors with the Sec4IoT framework (Costa and Pinto, 2015). Nevertheless, the need to assure that the collected data is always maintained within the control scope of the Sec4IoT framework, an end-to-end device authentication mechanism is required.

The paper is organized in sections. Section 2 describes and compares the work of others that is related to ours. Section 3 describes the proposed authentication mechanism and performs its security validation. Section 4 describes the experimental set-up, the implemented authentication prototype and presents the obtained results. Section 5 concludes the paper.

2. Related work

Representational State Transfer (REST) (Fielding, 2000) can be seen as a distinct way of deploying web-services where the web application can be seen as a finite-state machine that allows users to switch from one state to the other by following links between states. Web-services deployed in such a way are becoming quite common to do its simplistic and HTTP-like behavior. In REST, the server should not store any state information, delegating the burden of storing and maintaining state information to the clients. The key REST principles are threefold: 1) explicit use of HTTP methods; 2) stateless; and 3) resources must be named using URLs (Uniform Resource Locators).

The hyped stateless operation of a REST server may not be completely possible, especially if one considers clients authentication and authenticated sessions management. Current solutions make use of standardized HTTP related authentication mechanisms, such as Basic and digest access authentication (Leach et al., 2015), or of Open standard for authentication (OAuth) (Hardt, 2015; Hardt and Jones, 2015), or use proprietary, in-house developed, solutions.

2.1 HTTP Authentication

The HTTP protocol supports several authentication mechanisms in order to perform the access control to pages and other resources. This solutions make use of the 401 status code and the WWW-Authenticate response header (Leach et al., 2015). The following mechanisms use an user/password combination and some of them are only available in HTTP/1.1 (Fielding and Reschke, 2015):

Basic

This was the first authentication mechanism provided by the HTTP protocol. The client sends the user in clear text and the password as a base64 encoded text (unencrypted) to the server, making the user/password combination easily capturable and potentially reusable. It should not be used without HTTPS.

Digest HTTP/1.0

The client sends the user in clear text and the result of a hash function, having the user's password as input, to the server. Although the password cannot be captured or even reverted back to its clear text form, it is possible to resend previously captured requests that the hashed password (replay attack).

Digest HTTP/1.1

The digest authentication was improved in the version 1.1 of the HTTP protocol, now enabling mutual authentication, clients can now authenticate servers as well as servers can also authenticate clients, and additional message integrity protection. This new digest scheme is also resistant to replay attacks, since it includes a eight digit nonce count that increments each time the client makes a request, making replay request useless since the server will not honor them.

2.2 Form-based Authentication

Developers, instead of relying on authentication at the protocol level, can make use web-based applications or HTML code embedded into their web pages. They can use INPUT elements in HTML Forms to request the client's credentials (User and Password) as a normal part of their web application. In particular, the TYPE=PASSWORD INPUT element will allow users to insert their password while visually hiding the password from close eyes. This approach maintains the access control within the web application and hampering the HTML capabilities for both internationalization and accessibility.

2.3 OAuth

The Open standard for Authentication (OAuth) (Jammer-Lahav, 2015), which recently evolved to version 2.0 (Hardt, 2015; Hardt and Jones, 2015), is described as *"an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications"*. It provides a method for web application to grant third-party access to their resources without, actually, sharing their clients credentials (user and password). It may also provide a way to limit access (in scope, duration, etc.). In its version 2.0, a token-based approach is introduced that, after the initial authorization, is used to orchestrate and approve the interaction between the resource owner and the HTTP service or to allow access to the third-party applications.

2.4 Token based REST authentication

In (Peng et al., 2009) the authors propose the use of a token based approach for authentication in REST-based web services. Their proposal consists in extending the HTTP authentication to include a UsernameToken as a secondary password verification. This would allow providers to customize their own authentication according to their specific need, improving flexibility and security, and introducing the possibility of the server challenging the client in order to authenticate him. The key advantage of this solution was lost as it addresses the same shortcomings of the first HTTP Digest authentication, as does HTTP1.1.

Solution	Data enc.	Mutual auth.	Replay resistant	MiTM resistant	3rd parties	Auth. control	OTP
Basic	N	N	N	N	N	HTTP	N
Digest	Y	N	N	N	N	HTTP	N
Digest 1.1	Y	Y	Y	Y	N	HTTP	N
OAuth	Y	N	Y	Y	Y	App	N
AuthToken	Y	N	Y	Y	N	HTTP	N

Table 1: Related work comparison.

2.5 Summary

REST is a distinct way of deploying web-services that is becoming quite common due to its simplistic and HTTP-like behavior. The key REST principles are threefold: 1) explicit use of HTTP methods; 2) stateless; and 3) resources must be named using URLs (Uniform Resource Locators). The hyped stateless operation of a REST server may not be completely possible, especially if one considers clients authentication and authenticated sessions management. Current solutions make use of standardized HTTP related authentication mechanisms, such as Basic and digest access authentication (Leach et al., 2015), or of Open standard for authentication (OAuth) (Hardt, 2015; Hardt and Jones, 2015), or use proprietary, in-house developed, solutions.

The HTTP protocol supports several authentication mechanisms (Fielding and Reschke, 2015) in order to control the access to pages and other resources. This solutions make use of the 401 status code and the WWW-Authenticate response header (Leach et al., 2015). In form-based authentication, developers, instead of relying on authentication at the protocol level, can make use of web-based applications or HTML code embedded into their web pages. They can use INPUT elements in HTML Forms to request the client's credentials (User and Password) as a normal part of their web application. The Open standard for Authentication (OAuth) (Jammer-Lahav, 2015; Hardt, 2015; Hardt and Jones, 2015) provides a method for a web application to grant third-party access to their resources without, actually, sharing their clients credentials. In (Peng et al., 2009) the authors propose the use of a token based approach for authentication in REST-based web services. Their proposal consists in extending the HTTP authentication to include a UsernameToken as a secondary password verification. This would allow providers to customize their own authentication according to their specific need, improving flexibility and security, and introducing the possibility of the server challenging the client in order to authenticate it. The key advantage of this solution was lost as it addresses the same shortcomings of the first HTTP Digest authentication, as does HTTP1.1.

Table 1 compares the solutions identified as related to ours. For instance, the version 1.1 of the HTTP digest authentication performs data encryption (2^{nd} column), authenticates both server and client (3^{rd} column), resists attacks that resend previously exchanged messages (4^{th} column), is secure against eavesdropping and Man in The Middle (MiTM) attacks (5^{th} column), does not require trust third parties (6^{th} column), the authentication is performed at the HTTP layer (7^{th} column) and has no support for OTP (8^{th} column). Regarding the authentication control, it can either be controlled by the web server (identified in the table with HTTP) or by the web-service (identified with App). None of the presented solutions supports transaction control by means of OTP. The authors believe that such OTP per transaction approach is the one that better suites the REST design philosophy.

3. Minimalist Authentication Mechanism

The proposed Minimalist Authentication Mechanism (MAM) requires a secure client register procedure, deployed as a secure web page (HTTPS), that is assumed to be in operation. The secure register procedure will enable the

secure generation and exchange of a per device secret (A_{sec}). The proposed algorithm implies that any request made by clients to the server must comprise, aside other parameters, the client identification and an OTP. A, per device, set of OTPs is generated with the login procedure. The login is initiated when the client calls the login procedure made available by means of a REST-based web-service. The client computes the non-guessable random value $nonce_{A1}$, calculates both the cli_n_2 and the $Time$ values and passes them as parameters to the login procedure. The nonce generation function is assumed to be secure. $Time$ value is obtained by rounding up the current time in intervals of 10 minutes. The server will compute a local n_2 value using a secure hash function over $nonce_{A1}$, n_1 and the $Time$ value. The computed n_2 value, if equal to the cli_n_2, will be used to create the initial security token (tk_0). After the generation of the initial token, tokens tk_1 to tk_{512} will be calculated by using the cryptographic hash function over the previous token. Both server and client will store the set of 512 tokens to be used as OTPs, in reverse order, in the subsequent 511 requests of that client. Such will enable anti-replay protection and prevent both man-in-the-middle and Denial of Service (DoS) attacks. Additionally, and due to the fact that the server returns both the $seed^{Time}$ and the token tk_{512}^{Time} to the client, both server and client are mutually authenticated.

The way the OTPs are obtained depends on the type of sensor (Figure 1). On the one hand, type B sensors have the required capabilities, in terms of wireless communications, CPU and available battery lifetime, to perform a complete login by themselves. At the end of the login procedure, the device will have a set of OTPs to be used in the subsequent requests to the REST web-service. On the other hand, both Type A and C require additional devices in order to complete a successful login procedure. Currently, type A devices require a specific WSN gateway, whereas type C devices require a paired smart-phone.

In the proposed solution, a smart-phone will be used to complete a successful login and to obtain a set of OTP which will then be sent to the type A or C sensor by any means available in the sensor. Such will avoid two major drawbacks of the current solutions. Firstly, the WSN specific gateway will no longer be required as it can be replaced by a existing wireless Access Point (AP) with Internet connectivity. Secondly, none of the cryptographic material to be stored on the sensors, the set of OTPs, can be used to perform a new login in the system. A sensor login procedure that will authenticate the smart-phone is assumed to be in operation. Nevertheless, the REST web-service does not take part in the sensor/smart-phone authentication procedure.

Type A sensors run on ultra low power hardware, use small sized batteries and communicate periodically in order to save power. The size of the OTPs set must be adapted to each case. For instance, if a sensor communicates with the server once per hour, it will require 24 OTPs per day, 167 per week, or 672 OTPs per month. The REST web-service will have a set of URLs that will enable the login procedure to reply with sets of OTPs of different sizes and using different secure hash functions.

Table 2 shows the notation adopted throughout this work. For instance, A_{sec} represents the symmetric encryption key of the entity A, and $\{M\}_K$ represents the result of the encryption of the message M with the encryption key K, also known as ciphertext. If $H()$ is assumed to be a secure hash function, then $H(M)$ will represent the result of the referred secure hash function having the message M as input.

Algorithm 1 describes the procedure for device login on the server side. A secure client register procedure, deployed as a secure web page (HTTPS), is assumed to be in operation. The secure register procedure will enable the secure generation and sharing of a secret per client (A_{sec}). The proposed algorithm implies that any request made by clients to the server must comprise, aside other parameters, the client identification and a security token. The set of security tokens for each client is generated upon the login procedure.

The login procedure is initiated when the client calls the login procedure made available by means of a REST-based web-service. The client computes 2 non-guessable random values (eg.: $nonce_{A1}$ and $nonce_{A2}$) and passes them as parameters to the login procedure. In turn, the server will compute a set of values (n_2,n_3 and n_4) using secure hash functions over $nonce_{A1}$, n_1 and the current time. These three values differ only in the used time value to permit up to a 2 minute temporal window between the issuing of the login procedure on the client and its

Notation	Meaning
A_{sec}	Symmetric secret key of entity A
$[item_1, item_2]$	Array containing $item_1$ and $item_2$
$item_1 : item_2$	Concatenation of $item_1$ with $item_2$
$\{M\}_K$	Message M encrypted with key K
$H(M)$	The result of hash function H over input$M$
$nonce_A$	A non-guessable random value generated by A

Table 2: Adopted notation.

Require: Pre shared secret A_{sec}
Receive ($nonce_{A1}$,$nonce_{A2}$) from A
$n_1 \leftarrow Hash(A_{sec})$
$n_2 \leftarrow Hash(Time : n_1 : nonce_{A1})$
$n_3 \leftarrow Hash(Time - 1 : n_1 : nonce_{A1})$
$n_4 \leftarrow Hash(Time - 2 : n_1 : nonce_{A1})$
$seed \leftarrow Hash(Random())$
if $n_2 = nonce_{A2}$ **then**
  $tk_0 \leftarrow Hash(seed : n_2)$
else if $n_3 = nonce_{A2}$ **then**
  $tk_0 \leftarrow Hash(seed : n_3)$
else if $n_4 = nonce_{A2}$ **then**
  $tk_0 \leftarrow Hash(seed : n_4)$
end if
if tk_0 **not** NULL **then**
  for $i = 1$ to 512 **do**
    $tk_i \leftarrow Hash(tk_{i-1})$
  end for
  return $(seed, tk_{512})$
end if

Algorithm 1: Procedure for device login on the server side

processing on the server. The selected n value will be used to create the initial security token (tk_0). After the generation of the initial token, the tokens tk_1 to tk_{512} will be calculated by using the cryptographic hash function over the previous token. Both server and client will store the set of 512 tokens to be used, in reverse order, in the subsequent 511 requests of the related client. Such will enable both anti-replay protection and prevent man-in-the-middle attacks. Additionally, and due to the fact that the server returns both the *seed* and the token tk_{512} to client, both server and client are mutually authenticated.

3.1 Security analysis

The Automated Validation of Internet Security Protocols (AVISPA) (Armando et al., 2005) tool was used to perform the security validation of the proposed authentication mechanism. The AVISPA tool performs the automated validation of security protocols described in High Level Protocol Specification Language (HLPSL) (Chevalier et al., 2004). HLPSL enables the description of both the protocol and the required security properties, such as secrecy and authentication. AVISPA adopts by AVISPA is the Dolev-Yao intruder model (Dolev and Yao, 1983) where the intruder is in complete control of the network.

A HLPSL protocol specification for AVISPA requires that both the environment and the security goals are described. The environment in HLPSL is a top-level role consisting of a set of protocol sessions, each session being described by the involved participants and their shared knowledge, if any.

```
1  role environment()
2  def= const sec_token, auth_token : protocol_id,
3  h : hash_func,  a, b : agent,
4  nab, nib, time : text
5  intruder_knowledge = {a, b, nib, time}
6  composition
7           session(a,b,nab,time,h)
8       /\  session(a,i,nib,time,h)
9       /\  session(i,b,nib,time,h)
10 end role
```

Listing 1: Environment specification.

In our HLPSL specification[1], the environment describes three sessions, as shown in the lines 7, 8, and 9 of Listing 1. One session is the legitimate session (line 7), involving only honest participants (a and b), while on the other two sessions the intruder impersonates either one of the honest participants (lines 8 and 9). As initial knowledge, we assume the intruder knows the honest participants, and that it has a nonce (*nib*) pre-shared with the server. In this way, it is possible to verify the protocol security even when the intruder is a legitimate user, but tries to impersonate other users.

[1] Available at http://www.estgf.ipp.pt/~apinto/mawr.hlpsl

```
1  role client(
2  ...
3   /\ RCV(A.B.Seed.Token)
4   =|>
5   State' := 4
6   /\ Token' := H(Seed.Na2)
7   /\ request(A,B,auth_token,Random)
8  end role
9  role server(
10 ...
11  /\ RCV(A.Na1'.Na2')
12  =|>
13  State' := 3
14  /\ Random' := new()
15  /\ Seed' := H(Random')
16  /\ Na1' := H(Nab)
17  /\ Na2' := H(Time.Na1'.Na1')
18  /\ Token' := H(Seed'.Na2')
19  /\ SND(A.B.Seed',Token')
20  /\ witness(B,A,auth_token,Random')
21  /\ secret(Random',sec_token,{A,B})
22 end role
23 ...
24 goal
25   secrecy_of sec_token
26   authentication_on auth_token
27 end goal
```

Listing 2: Security goals specification.

Listing 2 is an excerpt of the specification of our authentication mechanism that shows our definition of the required security goals. These goals are the secrecy of *Random'* (lines 21 and 25), and its ability to serve has an authentication token (lines 7, 20, and 26) between the participating entities. We performed the security verification with all four techniques available in AVISPA. None of them was able to find an attack to our authentication mechanism.

3.2 Additional security considerations

The proposed secure authentication was also analyzed with common security attacks in consideration, and as follows:

1. Eavesdropping attacks
   The client-server communication confidentiality is obtained by means of a pre-shared secret between that specific client and the server. This pre-shared is assumed to be available on the client and on the server and to be refreshed frequently. For instance, such pre-shared key may be refreshed upon every client successful login. If an attacker obtains a capture of the exchanged messages, and while being able to obtain the

$nonce_{A1}$ and $nonce_{A2}$ values, these are the result of one way hash functions. Meaning that eavesdropping attacks are not possible.

2. Man-in-the-middle attacks
   An attack such as a man-in-the-middle attack is only possible if the pre-shared key is compromised. The proposed solution assumes that this key is secure. Nonetheless, the pre-shared key is never transmitted on the link an is assumed to be a result of a specific pre-shared key creation procedure that may be triggered by the user whenever he wants to, by means of a user management web site requiring a two-factor authentication.

3. Key control attacks
   Despite the fact that both sides make use of random values to generate or verify an authentication token (tk_{512}), these values are never exchanged in clear text. The only value exchanged between client and server, besides the authentication token, is the seed value. The seed value is, in turn, the result of a secure hash function. Neither entity can force the other in to generating a specific seed or authentication token. Meaning that attacks that are based in key control are also not possible.

4. Replay attacks
   DoS attacks are based on overwhelming a server with requests so that it is not able to respond to legitimate requests. The proposed solution, uses the *Time* value, rounded up to 10 minute intervals, so that all requests sent by the same client within this time interval (up to 10 minutes) will have the exact same reply. Due to being a REST-based solution, it can easily be deployed within a Content Delivery Network (CDN) (Pathan et al., 2008), i.e. such reply could be cached, making it very difficult to perform a successful DoS attack. This approach limits the number of per device successful logins to one authentication per time period of 10 minutes. This is a drawback of the proposed solution but the time period can be reduced and fined tuned to each implementation.

4. Experimental results

Multiple sets of experiments were conducted in order to validate de proposed solution. Both client and server were prototyped and evaluated. Concluding the section, a comparison of the proposed solution with the related work is presented.

4.1 Server prototype

The server prototype was developed using Java and the *Netbeans Java* IDE. The Web application ARchive (WAR) file was built and deployed on a Glassfish 4.1 server, running on a 64 bit Linux system (kernel 3.18.8-201.fc21.x86_64) with 8GB of memory and a dual-core Intel Pentium G645 2.9GHz processor. All results shown in this section were obtain by running multiple sets of 1000 executions each. The authentication prototype accepted the URLs shown in Table 3. SHA-256 and SHA-512 where the selected secure hash algorithms, from the list of the algorithms available in the *Java* language, mainly because the remaining ones are currently considered insecure by multiple sources. Stevens demonstrated a collision attack on the MD5 algorithm (Stevens, M.M.J., 2006) and Liang, et. al, later presented an improved collision attack to the same algorithm (Liang and Lai, 2007). The SHA-1 algorithm was also demonstrated to be less complex that the initial expectation. In particular, Wang et al. demonstrated that the theoretical number of 2^{80} hash operations that where assumed to be required to find a collision could be reduced to a lesser value of 2^{69} operations (Wang et al., 2005).

Table 4 shows the average number of requests processed per second and the average time required by the server to process one request, for both the SHA-256 and the SHA-512 secure hash algorithms. As can be seen, a

URL	Description
/users	Obtain a list of all available users
/users/{id}	Obtain the user identified by {id}
/users/id/tokenset	Log in user identified by {id} and return the user security token (using SHA-256)
/users/id/tokenset/SHA-512	Log in user identified by {id} and return the user security token (using SHA-512)

Table 3: URLs available on the web-service prototype.

Digest algorithm	Requests/sec	Request (ms)
SHA-256	207.5	4.7
SHA-512	211.3	4.8

Table 4: Request processing capabilities by the server.

user login procedure takes approximately 5ms to be completed and the server is able to process about 210 requests per second. While there is a difference between the results obtained using different secure hash algorithms, this difference is very small can be neglected.

Figure 2 shows the time taken by the server to perform 6 sets of 1000 login requests, 3 sets per each secure hash algorithm. The server took approximately 4.8 seconds, on average, to reply to the login requests issued by 1000 clients while using the SHA-256 secure hash algorithm, for instance. The use of the SHA-256 algorithm does not present a performance advantage over the more secure SHA-512, even presenting a slightly more uncertain behavior.

Figure 3 shows the CPU used by the server while replying to the login requests of the clients, in percentage. The results were obtained using the *top* command, which used a percentage scale of 0% to 200% (100% per core), while monitoring only the java process (the Glassfish server). From the results we can observe that, during the majority of the time, more than one and half CPU cores were used by the server. During this time, the CPU reached and maintained the 200% mark. The remaining processes used the unaccounted CPU.

The bandwidth usage tend to be slightly more (approx. 8% more) when using the SHA-512 secure hash function. If we consider that login procedure, described in the previous section, returns the token tk_{512} that is the result of the used hash function and, evidently, will be larger when using the SHA-512 when compared to the use of SHA-256. The remaining elements of the messages exchanged between server and client are of equal size and independent of the secure hash algorithm that was used.

4.2 Client prototype

A client application for Android was developed. The key issue was to access the performance of the operation of its key functions. In particular, the client side of the device login procedure, described by Algorithm 1, was tested. The client prototype was developed using Android Studio 2.2.1. The resulting application file was deployed to multiple *smartphones* versions, with different hardware and software specifications as shown in Table 5. The client prototype performance was evaluated and all results shown in this section were obtain by running multiple sets of 1000 executions each.

Figure 4 shows two screenshots of the client side application developed for the performance testing. The left screenshots is the main user interface that accepts a number of repetitions (here set to 1000 repetitions) as has two buttons. Each one enables the start of the test either using the SHA256 or the SHA512 secure hash

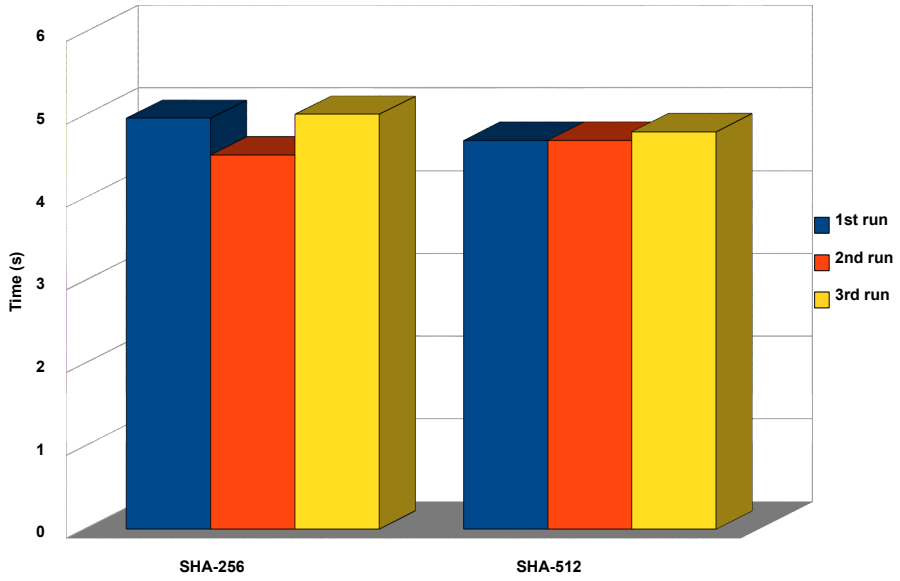

Figure 2: Global run time per groups of 1000 executions.

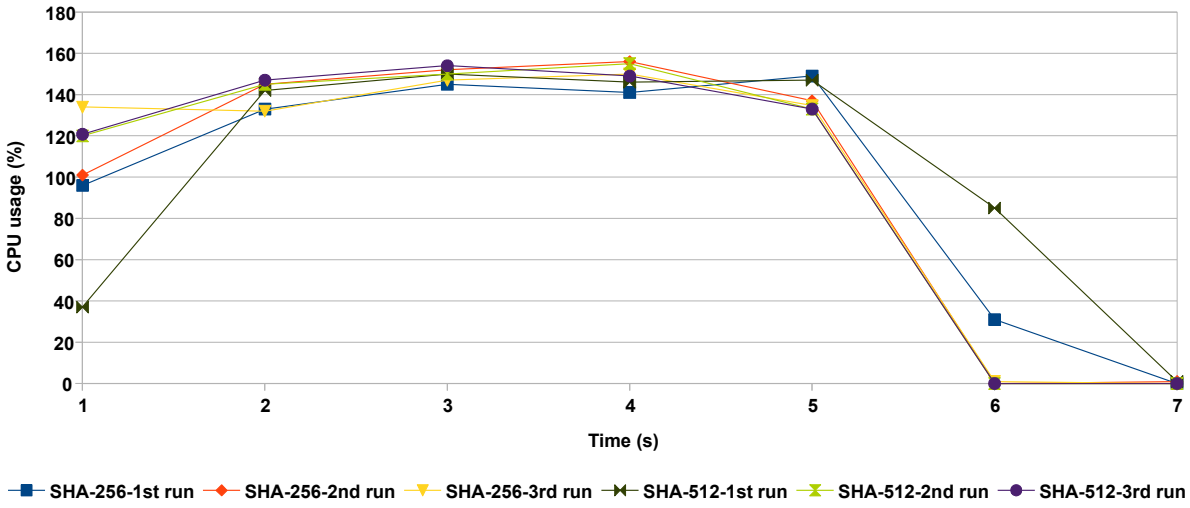

Figure 3: CPU utilization.

Brand	Model	Android version
Ecoo	e4	5.0 (API 21)
Huawei	P9 Lite	6.0 (API 23)
LG	L90 D405	7.1 (API 25)
Samsung	Dou S Galaxy	4.2.2 (API 17)
Samsung	Galaxy S3	4.4.4 (API 19)
Samsung	Galaxy S4	6.0.1 (API 23)
Sony	Xperia Z1	5.1.1 (API 22)

Table 5: Smartphones used in tests.

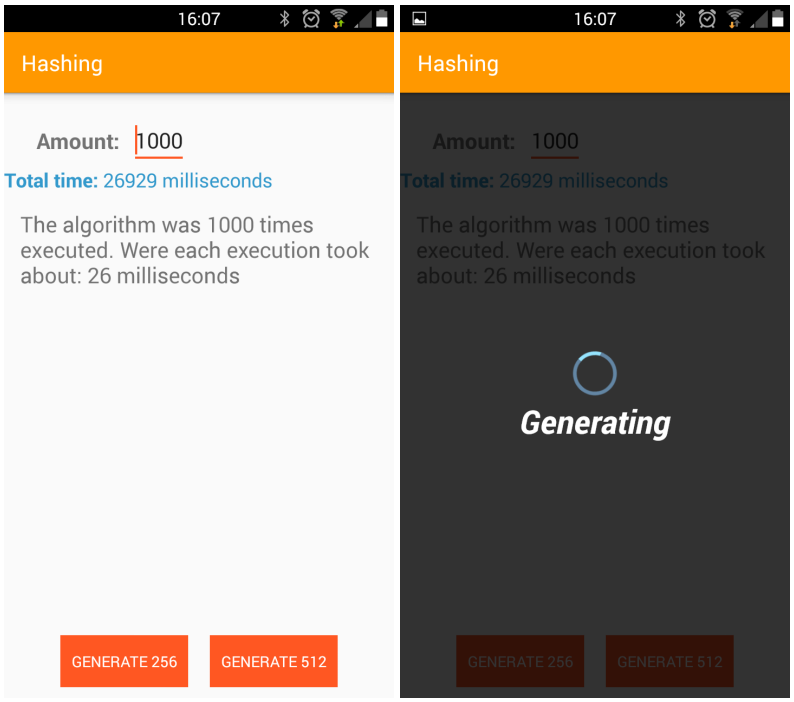

Figure 4: Screenshots of the test client application.

Smartphones	SHA256 (ms)	SHA512 (ms)
e4	33.8	32.2
P9 Lite	19.3	19.1
L90 D405	63.7	59.3
Duo S Galaxy	48.4	51.0
Galaxy S3	26.9	29.5
Galaxy S4	81.3	75.0
Xperia Z1	64.8	62.5
Overall average	48.3	46.9

Table 6: Average client request processing time.

HTTP Authentication Mechanism	Request (ms)
Basic	4,8
Digest HTTP1.0	64,7
Digest HTTP1.1	64,0
MAM (SHA-256)	4,7

Table 7: Average request processing time required per HTTP authentication mechanism.

algorithms. The left screenshot shows the application user interface while executing the multiple runs of the selected test. After the execution, the labels are updated to present the results. The results shown were performed in a Galaxy S3 smartphone.

Table 6 presents and summarizes the results obtained from the tests of the client side of the login procedure. For each smartphone, the time taken, in milliseconds, by the login procedure using the SHA256 (second column) or the SHA512 (third column) has the secure hash function. For instance, the P9 Lite took 19.3ms to conclude the login procedure while using the SHA256, and 19.1ms while using the SHA512. Table 6 shows that both the operating system version and the hardware architecture heavily influence the obtained results. The values range from 19.1ms to 81.3ms, which motivated a overall average calculation. The client side login procedure takes, on average, approximately 50ms. The authors consider this value as adequate and suitable.

4.3 Comparison with the related work

Table 7 shows the average processing time for each authentication mechanism made available by the HTTP protocol. The results show that both digest authentication mechanisms supported by HTTP take 64 ms, or longer, on average, to process a client authentication. The insecure basic authentication, that does not encrypt user credentials, was the only one that obtained processing times similar to those obtained by the proposed solution.

The results shown in Table 7 were obtained on a system running a 64 bit Linux with 8GB of memory and a dual-core Intel Pentium G645 2.9GHz processor. A HTTP server was setup in a virtual machine running Linux. The Apache web server was installed and configured to support the three authentication mechanisms identified. The Linux *curl* command was used as the HTTP client. All results shown in this section were obtain by running 3 sets of 1000 executions each. The proposed solution (MAM) is the fastest one, requiring only 4,7 ms to conclude. It is even slightly faster that the HTTP basic authentication.

5. Conclusion

The IoT is becoming the next big technological hype. New services that make use of IoT devices appear every day. These new services use web-based storage and the IoT devices are starting to interact directly with such web-storage. Two problems arise from such a scenario: 1) the privacy and control over the collected data; and 2) end-to-end authentication for low specked devices.

This work addresses the problem while still maintaining a REST-like API so it can be integrated within any existing web-service. Moreover, the proposed authentication mechanism does not require trust in third-parties, maintains the authentication control in the web application and, by requiring low computational power, it can be deployed in low end IoT devices. It is also secure even when deployed over open wireless networks.

6. References

ABIresearch, 2014. The Internet of Things Will Drive Wireless Connected Devices to 40.9 Billion in 2020.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Heam, P. C., Kouchnarenko, O., and Mantovani, J., 2005. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. volume 5, pages 281–285. Springer.

Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Mantovani, J., Modersheim, S., and Vigneron, L., 2004. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. *Proc. SAPS*, 4:193–205.

Costa, R. and Pinto, A., 2015. A framework for the secure storage of data generated in the IoT. *Advances in Intelligent and Soft Computing*.

Dolev, D. and Yao, A., 1983. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29:198–208.

Fielding, R. and Reschke, J., 2015. Hypertext Transfer Protocol (HTTP/1.1): Authentication.

Fielding, R. T., 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD, University of California, Irvine.

Hardt, D., 2015. The OAuth 2.0 Authorization Framework.

Hardt, D. and Jones, M., 2015. The OAuth 2.0 Authorization Framework: Bearer Token Usage.

Jammer-Lahav, E., 2015. The OAuth 1.0 Protocol.

Leach, P. J., Franks, J., Luotonen, A., Hallam-Baker, P. M., Lawrence, S. D., Hostetler, J. L., and Stewart, L. C., 2015. HTTP Authentication: Basic and Digest Access Authentication.

Liang, J. and Lai, X.-J., 2007. Improved Collision Attack on Hash Function MD5. *Journal of Computer Science and Technology*, 22(1):79–87. ISSN 1000-9000, 1860-4749. doi:10.1007/s11390-007-9010-1.

Pathan, M., Buyya, R., and Vakali, A., 2008. Content Delivery Networks: State of the Art, Insights, and Imperatives. In Buyya, R., Pathan, M., and Vakali, A., editors, *Content Delivery Networks*, volume 9 of *Lecture Notes Electrical Engineering*, pages 3–32. Springer Berlin Heidelberg. ISBN 978-3-540-77886-8.

Peng, D., Li, C., and Huo, H., 2009. An extended UsernameToken-based approach for REST-style Web Service Security Authentication. In *2nd IEEE International Conference on Computer Science and Information Technology, 2009. ICCSIT 2009*, pages 582–586. doi:10.1109/ICCSIT.2009.5234805.

Press, G., 2014. It's Official: The Internet Of Things Takes Over Big Data As The Most Hyped Technology.

Stevens, M.M.J., 2006. Fast Collision Attack on MD5. Technical report.

Wang, X., Yin, Y. L., and Yu, H., 2005. Finding Collisions in the Full SHA-1. In Shoup, V., editor, *Advances in Cryptology – CRYPTO 2005*, number 3621 in Lecture Notes in Computer Science, pages 17–36. Springer Berlin Heidelberg. ISBN 978-3-540-28114-6, 978-3-540-31870-5.

Xia, F., Yang, L. T., Wang, L., and Vinel, A., 2012. Internet of Things. *International Journal of Communication Systems*, 25(9):1101–1102. ISSN 1099-1131. doi:10.1002/dac.2417.