Hash-chain Based Authentication for IoT Devices and REST Web-Services

António Pinto[1] and Ricardo Costa[2]

[1] GCC, CIICESI, Escola Superior de Tecnologia e Gestão de Felgueiras, Politécnico do Porto and INESC TEC, Porto, Portugal
apinto@inesctec.pt

[2] GCC, CIICESI, Escola Superior de Tecnologia e Gestão de Felgueiras, Politécnico do Porto, Portugal
rcosta@estgf.ipp.pt

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 smart-phone, can authenticate each interaction of the devices with a REST web-service using One Time Passwords (OTP). 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 [1, 2]. 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 [3]. 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.

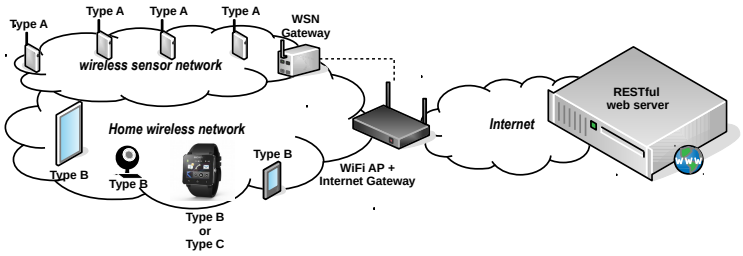

Fig. 1. Adopted reference scenario

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 [4]. 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

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 [5], or of Open standard for authentication (OAuth) [6,7], or use proprietary, in-house developed, solutions.

The HTTP protocol supports several authentication mechanisms [8] 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 [5]. 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) [6, 7, 9] provides a method for a web application to grant third-party access to their resources without, actually, sharing their clients credentials. In [10] 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. Related work comparison

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 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.

3.1 Security analysis

The Automated Validation of Internet Security Protocols (AVISPA) [11] 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) [12]. HLPSL enables the description of both the protocol and the required security properties, such as secrecy and authentication. AVISPA adopts the Dolev-Yao intruder model [13] where the intruder is in complete control of the network. Our HLPSL specification is available online[3].

The client-server communication confidentiality is obtained by means of a pre-shared secret between that specific client and the server. This pre-shared secret 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 cli_n_2 values, these are the result of one way hash functions. Meaning that eavesdropping attacks are not possible.

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.

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.

The proposed solution makes use of the the $nonce_{A1}$ that is assumed to be securely generated and only used once. If a second request is received with a repeated *nonce* value, the server will ignore the request and, thus, reject 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) [14], 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

The 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. SHA-256 and SHA-512 where the selected secure hash algorithms, from the list of the algorithms available in the *Java*

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

language, mainly because the remaining ones are currently considered insecure by multiple sources. Stevens demonstrated a collision attack on the MD5 algorithm [15] and Liang, et. al, later presented an improved collision attack to the same algorithm [16]. 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 [17].

Table 2. Request processing capabilities by the server

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

Table 2 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 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.

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.

Table 3. Average request processing time required per HTTP authentication mechanism

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

Table 3 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 3 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. Our previous solution (Sec4IoT) dealt with the first problem, the later subsisted.

This work addresses the second 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.

References

1. F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1101–1102, 2012.
2. ABIresearch, "The internet of things will drive wireless connected devices to 40.9 billion in 2020," 2014.
3. G. Press, "It's official: The internet of things takes over big data as the most hyped technology," 2014.
4. R. Costa and A. Pinto, "A framework for the secure storage of data generated in the iot," *Advances in Intelligent and Soft Computing*, 2015.
5. P. J. Leach, J. Franks, A. Luotonen, P. M. Hallam-Baker, S. D. Lawrence, J. L. Hostetler, and L. C. Stewart, "HTTP Authentication: Basic and Digest Access Authentication."
6. D. Hardt, "The OAuth 2.0 Authorization Framework."
7. D. Hardt and M. Jones, "The OAuth 2.0 Authorization Framework: Bearer Token Usage."
8. R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication."
9. E. Jammer-Lahav, "The OAuth 1.0 Protocol."
10. D. Peng, C. Li, and H. Huo, "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*, pp. 582–586, Aug. 2009.
11. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. C. Heam, O. Kouchnarenko, and J. Mantovani, "The avispa tool for the automated validation of internet security protocols and applications," vol. 5, pp. 281–285, Springer, 2005.
12. Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, J. Mantovani, S. Modersheim, and L. Vigneron, "A high level protocol specification language for industrial security-sensitive protocols," *Proc. SAPS*, vol. 4, pp. 193–205.

13. D. Dolev and A. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, vol. 29, pp. 198–208, 1983.
14. M. Pathan, R. Buyya, and A. Vakali, "Content delivery networks: State of the art, insights, and imperatives," in *Content Delivery Networks* (R. Buyya, M. Pathan, and A. Vakali, eds.), vol. 9 of *Lecture Notes Electrical Engineering*, pp. 3–32, Springer Berlin Heidelberg, 2008.
15. Stevens, M.M.J., "Fast Collision Attack on MD5," tech. rep., Mar. 2006.
16. J. Liang and X.-J. Lai, "Improved Collision Attack on Hash Function MD5," *Journal of Computer Science and Technology*, vol. 22, pp. 79–87, Feb. 2007.
17. X. Wang, Y. L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," in *Advances in Cryptology – CRYPTO 2005* (V. Shoup, ed.), no. 3621 in Lecture Notes in Computer Science, pp. 17–36, Springer Berlin Heidelberg, 2005.