Middle Man: An Efficient Two-Factor Authentication Framework

Costa, J. and Michalas, A.

This is a copy of the author's accepted version of a paper subsequently published in the proceedings of the *3rd IEEE International Conference On Computing, Communication, Control And Automation,* Pune, India 17th to 18 Aug 2017, IEEE.

The final definitive version is available online at:

https://dx.doi.org/10.1109%2FICCUBEA.2017.8463686

Middle Man: An Efficient Two-Factor Authentication Framework

Jose Costa
Cyber Security Group,
Department of Computer Science
University of Westminster, UK
jose.costa@my.westminster.ac.uk

Antonis Michalas
Cyber Security Group,
Department of Computer Science
University of Westminster, UK
a.michalas@westminster.ac.uk

***Abstract*—Two-factor authentication (TFA) is increasingly becoming a go-to for user security and identification. With an increase in cyber crimes each year more and more businesses (ranging from financial institutions to retail) are implementing TFA mechanisms as a way to ensure user credibility within their systems which in turn decreases the risk of any malicious users infiltrating their systems. In this invited paper, we describe a lightweight two-factor authentication system where legitimate users are using their mobile devices in order to get access to certain services. In addition to that, our service can be used as a single-sign-on framework since our system allows many different services to connect to our platform and give the option to their users to connect to their services via our TFA framework. To achieve that, we have built an Application Programming Interface (API) that can receive requests from authorised (i.e. registered) businesses. Finally, users are able to login to a server by using an iOS app, that we have developed, to receive a dynamic one-time-password (OTP). The OTP generated in a dynamic and random way with high entropy and it is valid only for a short period of time.**

***Keywords*—Security; Two-Factor Authentication; Software Token; One-Time-Password;**

I. Introduction

Protecting users' online accounts from unauthorized access is a problem that has been thoroughly studied during the last decades. However, and despite the considerable effort that have been put in to providing more robust and secure authentication mechanisms, most of the existing online services are still relying on the traditional single password-based systems.

An alternative authentication method that has recently gained in visibility is the so called two-factor authentication (TFA). TFA is becoming an increasingly popular approach that is adopted by many companies across various fields in order to provide a more reliable and secure user identification. With an increase in cyber attacks, each year more and more businesses (ranging from financial institutions to retail) are implementing TFA mechanisms as a way to ensure user credibility within their systems which in turn decreases the risk of malicious users infiltrating into their systems.

Even though there are a multitude of ways that TFA can work with numerous of combinations available, the most widely used for secure user identification is based on the use of One Time Passwords (OTP). The concept behind this approach is based on the fact that in a single password-based system, if a malicious adversary manages to get hold of a legitimate user's password, then the entire security and privacy of the underlying user is compromised. To make things worse, it has been observed that users tend to create easily memorable passwords while at the same time they use the *same password* for several different web services. In addition to that, users tend to authorize multiple third-party application to access their accounts and in some cases by giving full access rights. However, giving access to a third-party application implies that users share their login information (i.e. secret information) with an entity that can run under a weak security profile. Hence, if such a third party becomes compromised, then users' security and privacy are immediately under attack.

TFA has the potential to provide a realistic and reliable solution against such attacks. By adding an extra layer, typically in the form of a physical token, such as biometrics (fingerprints or retina scans), or an electronic token such as an OTP generated at the time of the login request, TFAs can make sure that each login attempt will be verified by the actual owner of the account. More precisely, during the login procedure the account holder will be required to prove that she owns unique information that is bound to her actual account (e.g. fingerprint, an OTP sent to her mobile phone etc.). As a result, even if an attacker manages to get access to the actual/main user's password, they would also need to get the user's current OTP to login. To this end, such authentication methods are considered more secure than traditional ones since it is much harder for an attacker to succeed in infiltrating an account.

While there are some available TFAs, most are either privately made in house, for a single company or are available but come with a large overhead which comes attached to drawbacks; this can be expensive to implement and time-consuming. The aim of our project is to provide a secure TFA platform that will enable other businesses to provide their users with the form of security without any change to their current infrastructure, making the transition effortless while keeping users' data as secure as possible.

A. Our Contribution

The main contribution of this work is the design and implementation of a lightweight and easy to use two-factor authentication framework. In addition to that, the overall framework has been designed in such a way that it allows users to login to different services without having to completely re-authenticate themselves. While this is close to a single-sign-on (SSO) approach it is not considered as such since the user will eventually need to prove that she knows an OTP that will allow her to properly login to a service. As a result, our approach requires the user to enter her credentials once (during the first login) and then she can move between different services only by using the corresponding OTPs that they will receive.

B. Organization

The rest of the paper is laid out in the following order. In Section II we study related work on two-factor authentication protocols and examples of existing implementations while in Section III we describe both the system and the adversarial model. In Section IV we describe the proposed two-factor authentication framework while in Section V we describe the system setup and present extensive experimental results showing the effectiveness of our approach. Finally, in Section VI we conclude the paper.

II. Related Work

This section presents the most important services that follow a two-factor authentication approach for the identification of users. In addition to that, we briefly describe how the underlying TFA protocols work and we present a list of advantages and drawbacks. Finally, for each described work we provide a brief comparison with our approach in order to highlight the differences and clearly present the contribution of our work.

In [1] authors proposed a two-factor authenticator protocol which consists of a biometric formulation known as $\mathsf{BioHash}$. This combines a user specific fingerprint B_i with a tokenized random number T which in turn produces a set of n binary bit strings $\mathcal{B} = \{\mathsf{b}_1, \ldots, \mathsf{b}_\mathsf{n}\}$. This method makes it hard for an adversary to get hold of the required authentication elements $(\mathcal{B}, \mathcal{T})$ to even make the final product needed for a successful authentication. One of the main drawbacks of this approach is that biometrics have a high implementation cost. Therefore, the proposed approach would not be a viable option to use for many organizations. Moreover, there are some additional drawbacks related to the accuracy of such systems. More precisely, false recognition rate (FRR) and false acceptance rates (FAR) can lead to many unsuccessful authentication attempts from legitimate users that are giving the correct input.

Authors in [2] proposed a two-factor authentication mechanism using keystroke analysis to identify users in a unique manner. Furthermore, the proposed solution requires users to provide a traditional fixed password during the initial login while authentication via the keystroke analysis is used to re-authenticate user's open session. Even though the proposed solution does not require any extra hardware, it has been observed that the keystroke analysis may not give accurate results when it is used on its own.

In [3], authors argued that using OTPs that are sent to users' devices in order to perform a two-factor authentication login process is a costly approach due to the SMS charges. Based on that, they proposed a two-factor authentication scheme a user's device produces multiple OTPs from an initial seed. The initial seed is produced by a set of unique parameters that are generated by the involved entities. As a result, applying the many from one function to a certain seed removes the requirement of sending SMS-based OTPs to users, and reduces the restrictions caused by the SMS system. Our approach does not require the use of SMS to send an OTP to a user who wishes to login. Even though users receive the OTPs on their mobile devices, this is not done via SMS but through the mobile application that we have built.

To successfully compile the requirements for our framework, we extensively studied the most important and widely used existing forms of TFA and how they would compare to our proposed system. To this end, we covered a variety of different approaches to TFA and dismissed examples which would not be suitable for our needs. An example of this would be using biometrics as TFA, firstly the complexity of this task would take longer than the allotted time given to complete this task. Another problem would be that biometric scanning needs specialised equipment which tends to be expensive, we would need to supply businesses using our platform with such devices which takes away from our aim of making it simple and easily integrable into any business. Hence, we focused our search on mobile based approaches that had a form of two or multi-factor authentication implemented.

The popular gaming platform Steam offers a mobile TFA which is available on both Android and iOS. Once the user has signed up to Steam Guard using her steam account, she can add her phone number which steam uses to send a verification code to finalise the setup and bind the device to the actual user. After this initial verification, each time the user tries to access her account the application will produce a code which changes regularly.

Duo is a company which focuses on verifying the identity of users for their customers as well as monitoring their devices for performance. They incorporate agents which monitor and enforce stronger policies by using TFA. They focus more on remote login side and ensuring users only have access to specific applications. This is a useful tool to have in large corporations that do not have have their own TFA system that would also like to add some monitoring options to their systems.

All of the compared systems are TFAs of their own, they are all at industry standard, some being more focused on specific areas than others – for example Duo must have an agent to get system health and performance reports. All of these, generate dynamic OTPs which is exactly the approach that we follow in our framework. Additionally, they also all extend their services onto the mobile platform – an approach that we also followed. The Steam application and Duo layouts seem more suitable for our needs as we need the underlying application to be simple but serve its purpose of providing the OTP to the user.

Duo is the only one which can provide services to different businesses such as Sofware-as-a-Service (SaaS) [4] cloud-based services [5], [6], [7], [8] have the drawback of the extra baggage of needing an agent to be introduced into the client system. Hence, adoption and implementation are considered difficult. For the needs of our platform we decided to keep the functionality as simple as possible in order to achieve high efficiency while at the same time allowing businesses to use our service without requiring to make any changes to their infrastructure.

III. System Model and Threat Model

In this section, we identify all the participating entities in our model and we describe their operations and how they relate to each other. Furthermore, we define the threat model that we consider by explicitly describing the capabilities of a malicious adversary.

Our system consists of three participating entities. The end-users, a set of webservices that the users have access to and our actual two-factor authentication server that manages users' access to the webservices.

User (u): Let $\mathcal{U} = \{u_1, \dots, u_n\}$ be the set of all users that are registered with our service. Users within the system can use our service to access/login various *webservices*[1]. In order for a user u_i to successfully login and/or switch between different webservices, she will first have to download our mobile application. Through this application, u_i will send a login request and if the request is legitimate, she will successfully receive an OTP that will allow her to authenticate herself and login to the requested service.

Webservice (ws): Let $\mathcal{WS} = \{ws_1, \dots, ws_m\}$ be the set of all webservices that our framework provides service to. Each $ws_i \in \mathcal{WS}$ has a public/private key pair denoted as $\mathsf{pk}_{\mathsf{ws_i}}/\mathsf{sk}_{\mathsf{ws_i}}$. The public key is shared with all other entities in the system model while the private key is kept private. A ws_i communicates directly with our TFA server (AS) to verify users' credentials to which AS can respond with a user validation (i.e. a login response) or a registration response if the user does not already belong to the underlying webservice.

Two-Factor Authentication Server (AS): AS can communicate both with the users and webservices. AS issues credentials for users to login onto registered webservices. Furthermore, it also generates and assigns API Keys when webservices register to the platform. These API keys will be used to validate and authenticate the received requests for the webservices'.

A. Adversarial Model

Our adversarial model is based on the Dolev-Yao adversarial model [9]. In this threat model, a malicious $\mathcal{ADV}$ can intercept, overhear, replay and change messages which are sent and received by any of the involved entities. More precisely, we assume that $\mathcal{ADV}$ can intercept and change all the exchanged messages between two or more legitimate users.

Cryptographic Security: We assume that any used encryption scheme is semantically secure. Hence, $\mathcal{ADV}$ cannot find any valuable information about the content of an encrypted message as long as she does not have access to the corresponding decryption key. Furthermore, we assume that a signature scheme is unforgeable and $\mathcal{ADV}$ cannot forge a valid signature. In addition to that, $\mathcal{ADV}$ cannot predict the output of a pseudorandom function and cannot guess a random number.

Asset Security: We assume that $\mathcal{ADV}$ cannot deny access to a valid user, such as making a web server unavailable. Hence, denial-of-service (DoS) attack [10], [11], [12], [13] is out of the scope of this paper. We also assume physical security (i.e. $\mathcal{ADV}$ does not have access to the memory of any of the participating devices/entities).

IV. Protocol Description

In this section, we describe our TFA protocol which constitutes the main contribution of the paper. The described protocol is successively applied to deploy a two-factor authentication infrastructure providing web-services user authentication as well as data integrity and security. We cover the 2FA architecture and what programming languages were used.

a) Setup: During the initialization phase, AS and each $ws_i \in \mathcal{WS}$ obtain a public/private key pair ($\mathsf{pk_{AS}}/\mathsf{sk_{AS}}$, $\mathsf{pk_{wsi}}/\mathsf{sk_{wsi}}$). The public key of each entity is shared with everyone while the private remains secret.

b) Registration: For each $ws_i \in \mathcal{WS}$ that wishes to gain access to our platform, they must first sign up providing details about their actual webservice. Upon completion, they will be assigned a unique API key which solely identifies the specific webservice. This key will be used to authenticate requests made by ws_i when they wish to validate user access.

In addition to that, each user u_j must also register to our platform [2]. When u_j sends a registration request, AP is responsible to verify the validity of the request (e.g. user u_j is not already registered). If the request is legitimate, then AP initiates the registration process.

When u_j registers they are required to provide personal details along with a username and password combination. Then, u_j calculates $m_1 = \langle username||H(password)||details\rangle$ and encrypts it with $\mathsf{pk_{AP}}$. The generated ciphertext (c_1) is sent to the AP. Upon reception, AP uses $\mathsf{sk_{AP}}$ to recover $m_1 = \mathsf{Decsk_{AP}(c_1)}$. Next, AP checks if said u_j is part of their system already, if not it saves their details into a secure database along with a timestamp. Additionally, AP generates a salt s_j, using a cryptographically secure pseudorandom number generator (CSPRNG) which is unique to u_j. Finally, AP calculates $SP_{u_{jp}} = H(H(password) + s_j)$ which is the stored password for user u_j. Furthermore, AP also generates another random binary sequence of length n, where the resultant sequence becomes a unique key (uk_j) for u_j which will later be used to calculate the OTP.

[1] These webservices are third-party services operating in a different host from ours.

[2] Note that registration process of user is different from the registration of ws_i.

c) Requests: User u_j wishes to login to a webservice $ws_i \in \mathcal{WS}$. First, u_j provides their username and password so that ws_i can verify that u_j is a legitimate and registered user. To this end, u_j constructs the following message $m_2 = \langle username || H(password) || ws_{i_{APIkey}} \rangle$. Then, ws_i encrypts m_2 with $\mathsf{pk_{AP}}$ and sends it to AP along with a signature σ_{m2} of the hashed version of the actual message.

Upon reception, AP uses $\mathsf{sk_{AP}}$ to decrypt the cipertext and recover m_2, calculates its hash and verifies the signature. Then, the validation of the API key $ws_{i_{APIkey}}$ follows. Providing the key is valid and the user exists, AP proceeds to find the username supplied in its users table and calculates the salted hash of the hashed password provided in m_2, $H(H(password) + s_i)$, comparing it to the stored hash. Once the users credentials have been certified, AP sends back a signed response to ws_i. Upon reception, ws_i verifies that AP accepted the initial request as valid and prompts u_j for an OTP.

d) OTP Generation: When u_j is requested for an OTP they open their application and an OTP will be displayed for a given amount of time. The generation of the OTP is the output of the following function $OTP = \mathsf{PBKDF}(uk_i || s_i || (C + \eta) || PRF)$ where PBKDF is a key derivation function such as the one described in Definition 1, C is the base iteration and η is defined as $\eta = \delta(CT - timestamp)/\alpha$, where CT is the current time and α denotes the generation rate.

Definition 1 (Key Derivation Function): KDFs are deterministic algorithms which are used to derive cryptographic keys, K, from a secret value. Password Based KDFs (PBKDF) include input of a password denoted as P, a salt, S, an iteration count , C, the length of the desired key $|\mathsf{K}|$ and the pseudorandom Function PRF to use. We denote K as: $\mathsf{K} = \mathsf{PBKDF}(P, S, C, |\mathsf{K}|, \mathsf{PRF})$

By hashing uk_i η times we can always ensure that the OTP is always different as η will always change every α seconds. Our algorithm will give us a hexadecimal string which we will take a substring of which will serve as our OTP.

e) OTP verification: In the verification step we first verify the signature of the decrypted ciphertext and AP repeats the OTP generation algorithm above since the required data is also stored.

During the last phase of the protocol, u_j provides ws_i their current OTP and then ws_i makes an API request to the AP by constructing message m_3 consisting of ws_i API Key, OTP, u_j username such that $m_3 = \langle username || OTP || ws_{i_{APIkey}} \rangle$ and then signing with $\mathsf{sk_{ws_i}}$ and encrypting with $\mathsf{pk_{AP}}$.

V. Experimental Results

We now describe the implementation and setup of our system and which technologies were chosen to fully meet our needs and requirements. We will discuss the languages and frameworks that we used and how they were used throughout the project.

a) Server-side Setup: We run our authentication platform on a Node JS server, Node JS is an environment which allows JavaScript to be run outside a Web Browser. Due to the lack of object oriented nature of JavaScript we decided to use Typescript. Typescript is a superset of JavaScript developed my Microsoft which adds class based objected oriented programming into JavacScript.

b) Setup: For our server, we needed something which had asynchronous and non blocking capabilities which would allow us to upscale or downscale depending on the user base. Node JS framework matched our criteria on every front; it also makes it possible to deploy new instances effortlessly in the event that our user base grows to the point where our current system model cannot handle the volume of requests. In [14], Tilkov et al cover the main aspects of using the Node framework for a high-performance server.

In order to prove the effectiveness of our approach, we tested the performance of the basic cryptographic algorithms that we have used throughout our project as well as the main functions that we built for our framework. For the needs of our experiments, we used Nodes own built in timer [15] in the process module. Process.hrtime() method returns a high-resolution real time tuple Array comprised of <seconds, nanoseconds>. This method snapshots the time at the start of a code block and another snapshot at the end and calculates the time difference. These times are relative to an arbitrary time in the past and not related to the time of the day and therefore not subject to clock drift. All the experiments were conducted on a resizable Virtual Private Server (VPS) with 512 MB Memory, 20 GB Disk running a Debian 8.7 x64 kernel version 3.16.0-4-amd64.

First, we measured the time that it needed to complete the PBKDF function on the server with the iteration count being the variable that changes. The results were gathered by simulating 4000 requests with jMeter [16] to the server to which the server returned the time taken to execute the block of code. Figure 2 illustrates the results.

Our second experiment, included to measure the execution timer for the decryption (figure 4) process as well as the execution of verification the verification process (figure 5). Both experiments we made 1000 iterations.

A. Server performance under load recordings

Using jMeter we also conducted and recorded load testing on the server. We can vary how many threads (users) we have active at one time and how many requests are made per thread as can be seen in figure 6 and figure 7. These tests were mading while running the PBKDF at 10000 iterations, which is considered as efficient if we take into consideration the connection time and loading of all plugins for the website and security agreements.

B. Keys, Certificates, Certificate Authorities and Safety features

For our platform to incorporate our desired security standards relaying hashed data via an unsecured channel was not

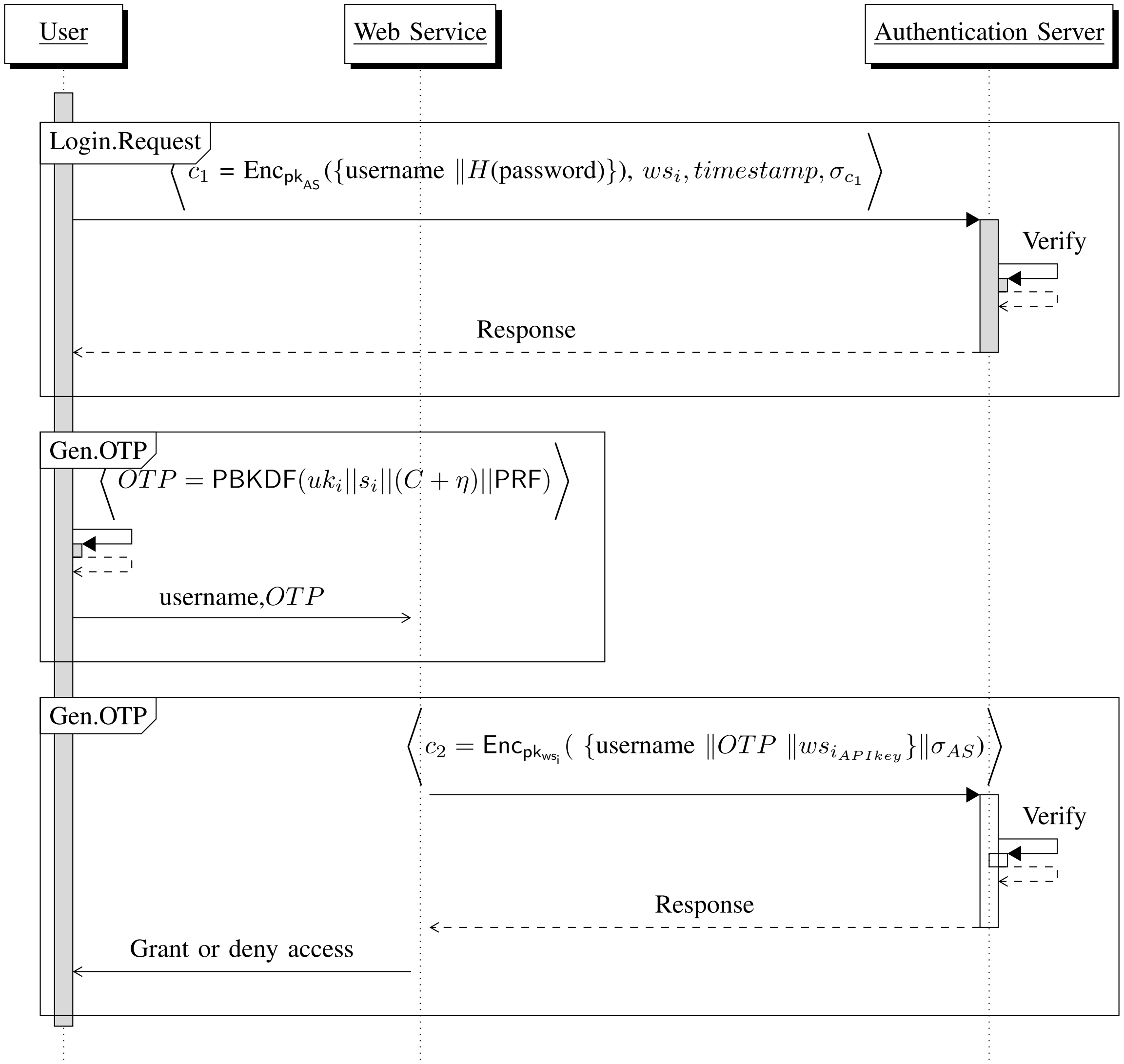
User
Web Service
Authentication Server
Login.Request
$\left\langle c_1 = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{AS}}}(\{\text{username } \| H(\text{password})\}), ws_i, timestamp, \sigma_{c_1} \right\rangle$
Verify
Response
Gen.OTP
$\left\langle OTP = \mathsf{PBKDF}(uk_i||s_i||(C+\eta)||\mathsf{PRF}) \right\rangle$
username,OTP
Gen.OTP
$\left\langle c_2 = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{ws}_i}}(\ \{\text{username } \|OTP\ \|ws_{i_{APIkey}}\}\|\sigma_{AS}) \right\rangle$
Verify
Response
Grant or deny access

Fig. 1: Sequence Diagram of Login Request and OTP Generation & Verification

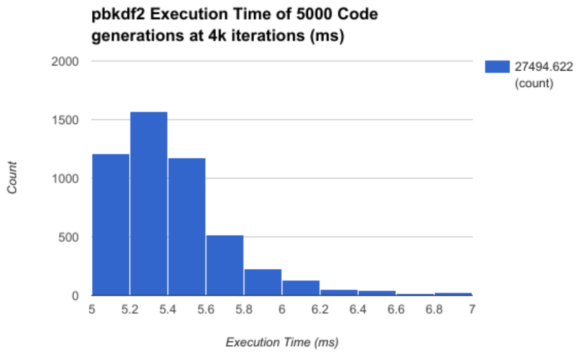

Fig. 2: Histograms of Execution time of the PBKDF algorithm (4 thousand Iterations)

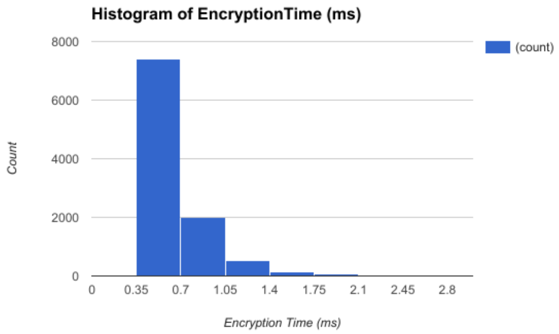

Fig. 3: Histogram of Encrypting execution time (ms), 1000 iterations

enough. This could easily be intercepted by an $\mathcal{ADV}$ and changed via a man-in-the-middle-attack (MITM) which could change the data without the recipient ever being aware.

At the first stages of development we produced our own

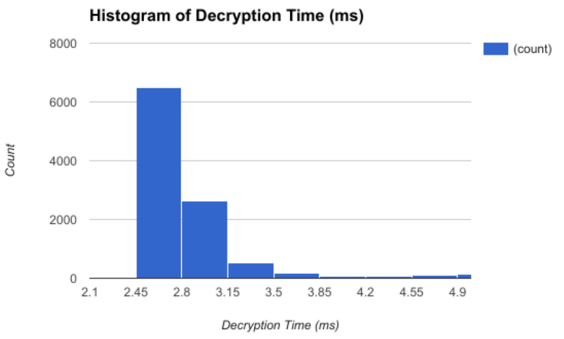

Fig. 4: Histogram of Decrypting execution time (ms), 1000 iterations

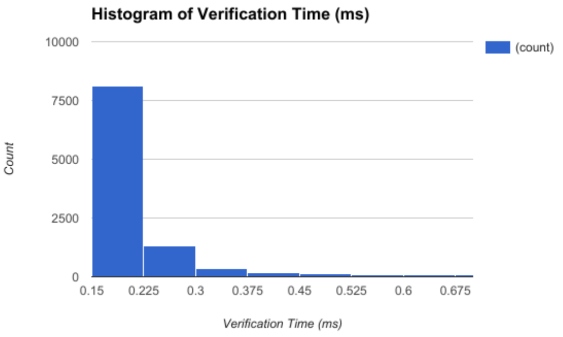

Fig. 5: Histogram of Verification execution time (ms), 1000 iterations

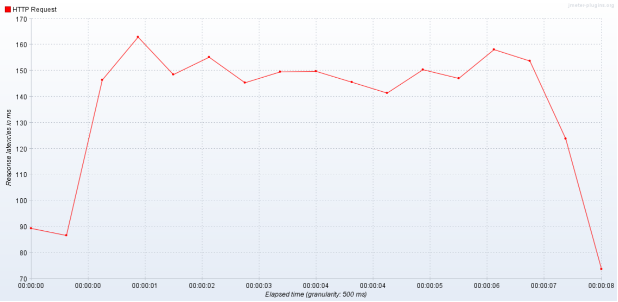

Fig. 6: Response Time graph of 20 threads making 50 requests each

self-signed certificate via OpenSSL [17]. This provides the means to create our own private key and certificate signing request which is required to create an SSL certificate. With our certificate being self-signed it still provided the protection a Certificate Authority (CA), the entity that is responsible for issuing certificates, signed certificate would provide but users would have to add our certificate to their exception list which would would turn them away. To solve this we turned to **Let's Encrypt** [18]. Let's Encrypt is a non profit automated CA whose aim is to promote the adoption of HTTPS for a more secure and privacy-respecting Web. They provide the means to obtain, renew and manage SSL/TLS certificates. Technology to encrypt web communication has been around for a long time but it's been a challenge to use even for technical people, Let's Encrypt allows us to acquire, install and manage our certificates at no cost.

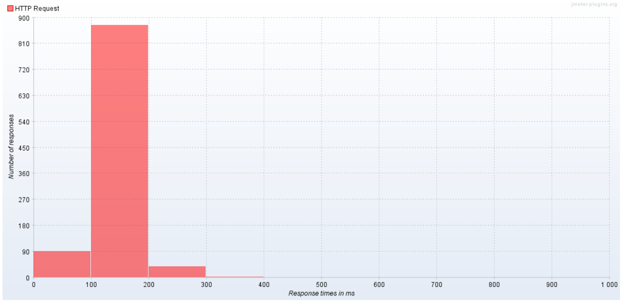

Fig. 7: Histogram Response Time graph of 20 threads making 50 requests each

After successfully setting up our certificate and our 2048-bit RSA keys we ran tests on SSL Labs and HTBridge [19], [20] which provide the means to perform deep analysis on configuration of a web server by check NIST guidelines [21], PCI DSS compiances [22], HIPPA guidance [23] as well as the industry's best practices and assigns it a rating.

VI. Conclusion

In this invited paper, we proposed a lightweight two-factor authentication system where legitimate users are using their mobile devices in order to get access to certain services. In addition to that, our service can be used as a single-sign-on framework since our system allows many different services to connect to our platform and give the option to their users to switch to different services via our TFA framework. To achieve that, we have built an Application programming interface (API) that can receive requests from authorised (i.e. registered) businesses. Furthermore, users are able to login to a server by using an iOS app, that we have developed, to receive a dynamic one-time-password (OTP). The OTP generated in a dynamic and random way with high entropy and it is valid only for a short period of time.

In the future we plan to provide detailed experimental results that we had to omit in this version due to space constraints. In addition to that, we plan to implement our protocol in a cloud environment [24], [25], [26] and measure its performance. Furthermore, we plan to explore the incorporation of our protocol with mobile sensing applications and with privacy-preserving reputation systems for cloud-based participatory sensing applications. The envisioned system will be based on [27], [28], [29] and will effectively maintain the privacy and anonymity of users [30], [31]. Finally, we plan to explore the possibility of the use such an authentication technique in e-Health applications [32], [33], [34] where users' data are considered sacrosanct [35], [36].

References

[1] A. T. B. Jin, D. N. C. Ling, and A. Goh, "Biohashing: two factor authentication featuring fingerprint data and tokenised random number," *Pattern recognition*, vol. 37, no. 11, pp. 2245–2255, 2004.

[2] T. Bhattasali and K. Saeed, "Two factor remote authentication in healthcare," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 380–386, Sept 2014.

[3] M. H. Eldefrawy, K. Alghathbar, and M. K. Khan, "Otp-based two-factor authentication using mobile phones," in *2011 Eighth International Conference on Information Technology: New Generations*, pp. 327–331, April 2011.

[4] A. Michalas and M. Bakopoulos, "Secgod google docs: Now i feel safer!," in *2012 International Conference for Internet Technology And Secured Transactions*, pp. 589–595, Dec 2012.

[5] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hbsch, and I. Paraskakis, "Paasword: A holistic data privacy and security by design framework for cloud services," in *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, pp. 206–213, 2015.

[6] N. Paladi and A. Michalas, ""One of our hosts in another country": Challenges of data geolocation in cloud storage," in *Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), 2014 4th International Conference on*, pp. 1–6, May 2014.

[7] A. Michalas, "Sharing in the rain: Secure and efficient data sharing for the cloud," in *2016 International Conference for Internet Technology And Secured Transactions*, pp. 589–595, Dec 2016.

[8] A. Michalas and K. Y. Yigzaw, "Locless: Do you really care your cloud files are?," in *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pp. 618–623, Dec 2015.

[9] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[10] A. Michalas, N. Komninos, N. R. Prasad, and V. A. Oleshchuk, "New client puzzle approach for dos resistance in ad hoc networks," in *Information Theory and Information Security (ICITIS), 2010 IEEE International Conference*, pp. 568–573, IEEE, 2010.

[11] A. Michalas, N. Komninos, and N. R. Prasad, "Mitigate dos and ddos attack in mobile ad hoc networks," *International Journal of Digital Crime and Forensics (IJDCF)*, vol. 3, no. 1, pp. 14–36, 2011.

[12] A. Michalas, N. Komninos, and N. Prasad, "Multiplayer game for ddos attacks resilience in ad hoc networks," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, pp. 1–5, Feb 2011.

[13] A. Michalas, N. Komninos, and N. R. Prasad, "Cryptographic puzzles and game theory against dos and ddos attacks in networks," *International Journal of Computer Research*, vol. 19, no. 1, p. 79, 2012.

[14] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[15] "Process hrtime." https://nodejs.org/api/process.html#process_process_hrtime_time.

[16] "jmeter." http://jmeter.apache.org.

[17] "Openssl." https://www.openssl.org/.

[18] "Let's encrypt." https://letsencrypt.org/.

[19] "Ssl lab." https://www.ssllabs.com/.

[20] "Htbridge." https://www.htbridge.com/.

[21] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[22] A. Shaw, "Data breach: from notification to prevention using pci dss," *Colum. JL & Soc. Probs.*, vol. 43, p. 517, 2009.

[23] C. for Disease Control, Prevention, *et al.*, "Hipaa privacy rule and public health. guidance from cdc and the us department of health and human services," *MMWR: Morbidity and mortality weekly report*, vol. 52, no. Suppl. 1, pp. 1–17, 2003.

[24] N. Paladi, A. Michalas, and C. Gehrmann, "Domain based storage protection with secure access control for the cloud," in *Proceedings of the 2014 International Workshop on Security in Cloud Computing*, ASIACCS '14, (New York, NY, USA), ACM, 2014.

[25] N. Paladi, C. Gehrmann, and A. Michalas, "Providing user security guarantees in public infrastructure clouds," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[26] Y. Verginadis, A. Michalas, P. Gouvas, G. Schiefer, G. Hübsch, and I. Paraskakis, "Paasword: A holistic data privacy and security by design framework for cloud services," pp. 1–16, 2017.

[27] T. Dimitriou and A. Michalas, "Multi-party trust computation in decentralized environments," in *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, May 2012.

[28] T. Dimitriou and A. Michalas, "Multi-party trust computation in decentralized environments in the presence of malicious adversaries," *Ad Hoc Networks*, vol. 15, pp. 53–66, Apr. 2014.

[29] A. Michalas, V. A. Oleshchuk, N. Komninos, and N. R. Prasad, "Privacy-preserving scheme for mobile ad hoc networks," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pp. 752–757, June 2011.

[30] A. Michalas and N. Komninos, "The lord of the sense: A privacy preserving reputation system for participatory sensing applications," in *Computers and Communication (ISCC), 2014 IEEE Symposium*, pp. 1–6, IEEE, 2014.

[31] A. Michalas, M. Bakopoulos, N. Komninos, and N. R. Prasad, "Secure amp; trusted communication in emergency situations," in *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*, pp. 1–5, May 2012.

[32] A. Michalas and R. Dowsley, "Towards trusted ehealth services in the cloud," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pp. 618–623, Dec 2015.

[33] K. Y. Yigzaw, A. Michalas, and J. G. Bellika, "Secure and scalable deduplication of horizontally partitioned health data for privacy-preserving distributed statistical computation," *BMC Medical Informatics and Decision Making*, vol. 17, no. 1, p. 1, 2017.

[34] K. Yigzaw, A. Michalas, and J. Bellika, "Secure and scalable statistical computation of questionnaire data in r," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2016.

[35] A. Michalas, N. Paladi, and C. Gehrmann, "Security aspects of e-health systems migration to the cloud," in *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on*, pp. 212–218, IEEE, 2014.

[36] R. Dowsley, A. Michalas, and M. Nagel, "A report on design and implementation of protected searchable data in iaas," tech. rep., Swedish Institute of Computer Science (SICS), 2016.