Trusted Execution Environment-Based Authentication Gauge (*TEEBAG*)

Ranjbar A. Balisane
Department of Computer Science
University of Oxford
Robert Hooke Building, Parks Road, Oxford, UK
ranjbar.balisane@cs.ox.ac.uk

Andrew Martin
Department of Computer Science
University of Oxford
Robert Hooke Building, Parks Road, Oxford, UK
andrew.martin@cs.ox.ac.uk

ABSTRACT

We present a new approach to authentication using Trusted Execution Environments (TEEs), by changing the location of authentication from a remote device (e.g. remote authentication server) to user device(s) that are TEE enabled. The authentication takes place locally on the user device and only the outcome is sent back to the remote device. Our approach uses existing features and capabilities of TEEs to enhance the security of user authentication. We reverse the way traditional authentication schemes work: instead of the user presenting their authentication *data* to a remote device, we request the remote device to send the stored authentication *template*(s) to the local device. Almost paradoxically, this enhances security of authentication *data* by supplying it only to a trusted device, and so enabling users to authenticate the intended remote entity. This addresses issues related with bad SSL certificates on local devices, DNS poisoning, and counteracts certain threats posed by the presence of malware. We present a protocol to implement such authentication system discussing its strengths and limitations, before identifying available technologies to implement the architecture.

CCS Concepts

•**Security and privacy → Trusted computing; Hardware-based security protocols;**

Keywords

Authentication; Trusted Computing; Trusted Executing Environment; TEE; SGX; TrustZone.

1. INTRODUCTION

In traditional remote authentication architecture, authentication occurs on a remote device against a remotely held *template* (we summarise this as remote-remote authentication). Using a remote service, the user provides their authentication *data* (password, fingerprint, one-time password, keystroke biometrics, etc.) to the remote device. The authentication takes place on the remote device by comparing the user provided authentication *data* with the authentication *template*(s) stored on the remote device. Based on the outcome of the comparison, a decision is made whether to grant or deny user access.

One of the problems with this authentication architecture is that the user is unable to authenticate the remote device. Therefore, users are vulnerable to phishing attacks carried out by attackers masquerading as a legitimate service provider (in order to obtain their authentication *data*).

Another problem arises from the user transmitting their authentication *data* from their device to a remote device. An attacker that has control of the user's local network, can manipulate local security measures making the transmitted *data* vulnerable to capture in its raw format. The absence of provenance information regarding the authentication *data* provided poses further issues. Malicious code can replay captured authentication *data*, or attempt to brute force the remote device in order to discover the authentication *data* with minimal cost to an attacker.

There have been a number of attempts to address these issues and will be discussed in the related work section, which are mainly built on top of the current architecture.

In this paper, we propose a new paradigm for authentication, changing the authentication location from a remote device to the local device (remote-local authentication). We provide an architecture and protocol where these attacks conceivably become ineffective using features and capabilities provided by Trusted Computing.

In our approach, instead of asking the user to send their authentication *data* to a remote device, we ask the remote device to send a previously stored authentication *template*(s) to a Trusted Execution Environment (TEE) available on a local device. The user then provides their authentication *data* to the TEE on their local device. The TEE carries out the authentication by comparing the two inputs and informs both parties of the outcome; we call this an authentication gauge. The decision could either be a binary or a confidence value depending on the matching algorithm specified and used.

The *template*(s) provided by the remote device will not be leaving the TEE, and it will therefore not be revealed to the user (nor to an impostor). The user-provided authentication *data* will be compared locally within the TEE, also not leaving the device. The *template* is encrypted so that it can only be accessed within a TEE.

TEE is a trusted computing platform, a hardware backed

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

NSPW '16, September 26 - 29, 2016, Granby, CO, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4813-3/16/09...$15.00
DOI: http://dx.doi.org/10.1145/3011883.3011892

secure environment, able to "strongly identify themselves" and "strongly identify their current configuration and running software" using strong cryptography [24].

It is worth noting, that the term TEE thus far, has been associated with ARM TrustZone, a secure area within mobile phones using ARM processors that meets the requirements for a trusted computing platform. According to ARM[1], TrustZone is "used on billions of chips" in a "diverse range of end markets, including smartphones, tablets, personal computers, wearables and enterprise systems". The new Intel Software Guard Extensions (SGX) also enables applications on PC platforms to meet the requirements of a trusted computing platform and will therefore be referred to in this paper as TEE enabler.

2. RELATED WORK

Bellovin and Merritt [4] present a protocol, Encrypted Key Exchange (EKE), with the aim of protecting weak passwords against brute force attacks by not sending the password to a remote location/entity. In the EKE a password is used by both parties to encrypt their own public key before sending it to the other party, or for encrypting parts of Diffie-Helman message exchanges. In order to establish a session key. The EKE protocol relies on the fact that encrypted data using a password is randomised, such as public keys. Therefore, an attacker cannot obtain a verifiable plain-text even if they were to brute force EKE encrypted data (encrypted using a weak password found in basic dictionaries).

One of the issues with this protocol is that the passwords have to be stored in plain text on the authentication server, or that the salt has to be sent in clear text to the user otherwise the initial encryption keys will not match [5]. Also, if an attacker is able to obtain the salt, they then gain access to the salted hash in the database, they will be able to discover the password with ease.

The EKE architecture is similar to our proposed architecture in that the authentication *template* is stored in a remote location. However, it differs from our approach since in the EKE, neither the authentication *template*, nor the authentication *data* is exchanged for the comparison to take place. Furthermore, the EKE and other password-authenticated key agreement protocols, are designed to work with passwords only, whereby the proposed architecture is meant to work with all existing authentication methods such as, fingerprints, keystroke dynamics, passwords, etc. This is rather important for service providers that do not wish their users to authenticate from places that they are not present, through the use of unsharable biometrics *data*.

Other approaches to enhancing authentication security rely on asymmetric cryptography, such as Fast Identity Online (FIDO) Alliance [23, 12], CBAT [7], PICO [29] and Apple Touch ID [1]. For these, asymmetric key pair is generated for the user account storing the private key locally, while sending the public key to a remote location. A random nonce is cryptographically signed using the stored private key and then sent to the remote location each time a user wishes to authenticate.

In these systems, the private key is unlocked to sign the random nonce using one or more authentication mechanism such as a password, fingerprint, etc. Legacy systems are used to enrol new devices and bind the user account to the device by associating and storing a private key on the local device. This approach removes the need for confidentiality of the transmitted authentication *data* as it is being transmitted. When anti-replay attack measures are implemented, an attacker obtaining the transmitted public key or the authentication *data*, will not be able to use it to authenticate themselves.

Generally, these approaches are used with devices that are already associated with a particular account, or a legacy authentication system is used to enrol a new device. For users who have accounts through a traditional system, enrolling a device with such architecture still relies on legacy systems, making legacy systems of authentication the least common mechanism for authentication.

Again, this authentication architecture (local-local authentication) differs from the proposed architecture here (remote-local authentication). It is a local-local architecture as the master authentication *template* is stored locally and the actual authentication (the matching) takes place locally, too. These approaches also permanently associate a user account with a particular device(s). In our approach, the *template* is stored remotely, and the remote device will have to send the *template* to the local device. The authentication takes place locally and only the outcome of the authentication leaves the TEE. Furthermore, our design allows the user to use any device without the need to enrol a user account associating it with a particular device.

There have been a number of attempts to address the provenance issue with authentication *data*. One approach is to use a physical security token and two factor authentication [21]. Physical security tokens are usually expensive to implement and the user is burdened with carrying the token with them at all times [17]. Considering an average user has around 25 accounts [15], having such a system for all accounts is impractical and costly.

Another approach is to use a challenge response mechanism, such as the Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) [21]. The most commonly used CAPTCHA are text-based, image-based and sound-based schemes [36]. They are meant to obscure messages from malicious code but remain identifiable to humans. However, some researches have shown that achieving this is difficult in practice. First, the messages can be too obscure for an average user to identify them [36]. Secondly, sophisticated malicious code might be able to do just as good a job (if not better) than an average user [6]. Since our approach uses a Trusted Path between the user and the TEE, it removes the need for CAPTCHA in order to prove that a physical entity has interacted with the input peripheral, and the authentication *data* was not generated or sent by malicious code.

3. ADVERSARY MODEL

The main aim of our architecture is to keep user authentication *data* secure. Though our approach addresses a large number of adversaries, it was designed to address the specific challenges discussed below. Physical attacks on the various chips supporting TEE are out of scope of this paper.

3.1 Phishing Attacks

The proposed approach can resist an adversary aiming to obtain authentication *data* through spoofing attack or DNS

[1]ARM - TrustZone, http://www.arm.com/products /processors/technologies/trustzone, (10/05/2016).

poisoning [10]. These attacks are often mitigated by using site certificates with an indication on the browser address bar indicating to the user that a particular site has a legitimate certificate and that they are who they claim to be. However, research has shown that users routinely ignore warnings presented by browsers regarding certificates [28, 30], as well as failing to notice visual indicators regarding certificates displayed within the browser URL bar. In mobile apps, this information is often not presented to the user.

3.2 Malicious Root Certificate

Attacks can be mounted by an adversary that is able to install an illegitimate root certificate on the user's device. The adversary is then able to generate fake certificates to decrypt captured communication from the device, including the user authentication *data* [19]. Our approach is designed to resist this attack.

This type of attack is commonly coupled with spoofing [11] or DNS poisoning [9] attacks to mislead the user into believing that the data leaving their device is encrypted and that they are visiting a legitimate site. Such adversaries include state-level actors which force their citizens to install an invalid root certificate authority or companies that own devices used by their users, and these certificates are pre-installed by the administrator controlling the network.

3.3 Brute Force Attack

Our proposed approach can resist an adversary who is using malicious code to carry out brute force attacks against a server seeking to obtain the authentication *template* for a user [33]. There are well known, freely-available tools such as Hydra, Medusa, Wfuzz, John The Ripper, etc. which brute force remote devices in order to discover the authentication *template*. Being freely available on the Internet, these tools have almost no cost for an adversary.

3.4 Malware on Local Device

Many approaches to authentication can be defeated by an adversary who is able to place malicious code on the user device aiming to steal their authentication *data* [18]. Our approach described here intends to defeat this attack, also.

4. APPLICATIONS

This protocol has a number of applications in contexts where a traditional authentication system might put users' authentication *data* at high risk.

An example includes users' transmitting their authentication *data* through a free open Wi-Fi hot-spot in airports and other similar places. Open Wi-Fi hot-spots can be malicious, set-up with the intention of stealing the authentication *data* of those connecting to the Internet through them. In such circumstances, it would be particularly useful for users to be able to authenticate the service provider first. This architecture will enable users to do so even in the absence of SSL certificate on the service provider's website. The identity of the service provider's site will be established by proving to the user that they have the correct authentication *template*, which matches the user authentication *data*. Also, since the authentication *data* will not leave the device, such attacks can be mitigated.

There are tools freely available on the Internet enabling attackers with a low powered computer and two wireless cards to hijack an auto-connect probe from a wireless interface. In this attack they use a 'SSL strip proxy' forcing the device to use HTTP instead of HTTPS and capturing the raw data with a packet dumping program, before relaying the data back to the intended remote service provider.

This protocol can assist service providers in keeping user authentication *data* secure, even when the user is authenticating from a known 'bad' location. A bad location can be a country or a company, intrusively monitoring and decrypting their users' communication in order to obtain their authentication *data*. A user travelling to a country which implements such intrusive digital communication monitoring can be assured that even if their session is hijacked while they are in the foreign country, their authentication *data* will remain secure.

This protocol can be used instead of using a CAPTCHA challenge response to differentiate between malware and a human to mitigate brute force attacks, since it uses a Trusted Path to the user. This architecture ensures that a physical entity is interacting with the peripheral (such as a keyboard) and guarantees that the response is not coming from malicious code. This will minimise the burden on the user whilst providing stronger guarantees. Through the Trusted Path, the user can be sure that their authentication *data* will remain secure even if a keylogger or malware is present on a device.

In addition to the intended uses highlighted above, this protocol can also be used to establish a TLS session key using RSA when a remote destination, such as a website, does not have a SSL certificate. Instead of having a random number generated by the user's device sent to the remote device encrypted with the remote device's public key to be used as a session key. Using our approach, the remote entity can generate a session key, encrypt it with the TEE's public key and send it to the TEE, to establish a TLS session key.

5. TRUSTED EXECUTION ENVIRONMENT PROPERTIES

The properties of TEE have been explored in a number of academic publications [24, 25, 26, 32]. Furthermore, several organisations outline these properties in documents such as the US National Institute of Standards and Technology (NIST) draft guidelines on hardware-rooted security in mobile devices (SP 800-164) [8], and the GlobalPlatform TEE specifications [16].

The most common properties described across the surveyed literature are summerised below, and listed in the same order as by Vasudevan et al [32].

Isolated Execution: An application must be able to run in complete isolation from other code inside or outside of the TEE. This is to protect the integrity and confidentiality of the application, and its data at run-time. The TEE isolated execution must not depend on the rich OS (normal OS) security.

Secure Storage: Secure storage ensures the integrity and confidentiality of the application and its data at rest. The secure storage key must be protected by hardware security components.

Remote Attestation: The TEE must be able to send strong attestation to remote parties regarding the system's identity and the identity of current configuration and running software. It must be able to attest the entire trusted computing base (TCB) of a given application. The attestation is made using a private key that is stored using hardware backed secure storage and accessible by a small TCB.

Secure Provisioning: TEE must be able to provide secure provisioning for data to be sent to a specific application within a specific device, providing protection for the confidentiality and integrity of the data.

Trusted Path: The TEE must be able to establish a Trusted Path to the input peripherals to ensure the authenticity of input data, and its confidentiality, when required, from other code running on the system. In the smartphone arena, GlobalPlatform is aims to standardise TEE properties in general and Trusted Path in particular[16]. Furthermore, the W3 Web Security Context have aimed to take a similar approach on the web [34].

6. SYSTEM DESIGN

The architecture consists of three entities, the client device, a TEE environment within the client device, and the service provider device (server device).

A: client device; its general computing environment.

B: server device; holds 'master copy' of (generally hashed) credentials - aka authentication *template*.

TEE: trusted execution environment present in the client's (Alice's) device.

Typically, A wants to authenticate to use a service provided by B. B sends the authentication *template* to A's TEE; the TEE has a Trusted Path to the user; receives input; checks it against the template; and returns (authenticated) answer to B.

This system does not output any authentication *data* or *template* by design and it only receives such data. The authentication application can only output requests, public keys, and confirmations.

It is designed to work with users already enrolled that have provided a remote device (e.g. service provider) with their authentication *template*(s) securely. The architecture can be used for any authentication method and mechanism. The main idea for the proposed architecture relies on the existing security features and properties of a TEE.

The isolated execution protects the authentication application from other code that is present within the TEE or in the rich world. Since applications are isolated during execution and at rest, other applications present will not be able to view or even be aware of transactions carried out by the authentication application within the TEE.

Using the remote attestation capabilities the TEE will attest to Bob (the service provider) of the identity of the TEE and the authentication application within the TEE and their current configuration to provide assurances of their integrity.

The use of trusted input by the TEE creates a Trusted Path for Bob, since Bob knows only a physical entity could have provided the authentication *data* and it was encrypted using an OTP generated by Bob.

The following simple message exchange highlights how the system works:

1. TEE generates a random public key/private key pair, E_A and D_A, and signs the public key using asymmetric cryptosystem using the TEE private key, yielding $\{E_A\}\ D_{TEE}$ and sends it to B.

2. B verifies the D_{TEE} signature, then generates a nonce OTP_B, and encrypts the authentication *template* P_B and OTP_B using E_A producing $E_A\ (P_B, OTP_B)$, then sends it to the TEE.

3. The TEE decrypts the message $D_A\ (P_B, OTP_B)$. Then it will ask A to provide P_A using a trusted input.

4. A provides P_A using trusted input.

5. TEE compares P_B and P_A and informs the user of the outcome. TEE will also encrypt the outcome using OTP_B yielding OTP_B(Outcome), as well as information regarding trusted input before sending it back to B.

6. B decrypts the message $OTPB^{-1}(OTPB(\text{Outcome}) = \text{Outcome})$.

Table 1: Notation

A	Alice, user device.
B	Bob, server device.
TEE	Trusted Execution Environment.
P	Authentication data/template, shared secret.
OTP	Random nonce, Onetime-Password.
$E_K(X)$	Asymmetric (public-key) encryption of X with (public) key E_K.
$D_K(X)$	Asymmetric (private-key) decryption of X with (private) key D_K.
$\{X\}D_K$	Asymmetric (private-key) signing of X with (private) key D_K.
OTP(info)	Symmetric (secret-key) encryption of 'info' with key OTP.
OTP^{-1}(info)	Symmetric (secret-key) decryption of 'info' with key OTP.

A simple diagram depicting the message flows for the authentication operation is shown in Figure 1.

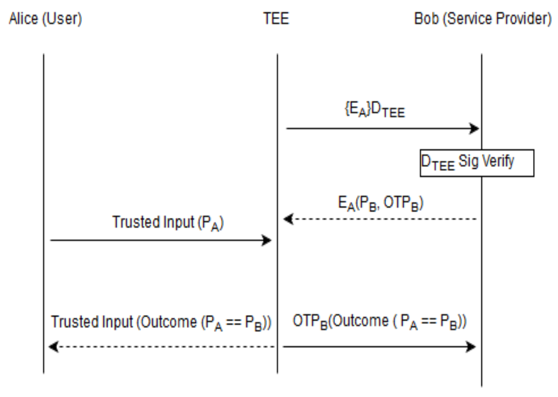

Figure 1: Abstract protocol message flow.

A typical TEEBAG architecture and authentication message flow is shown in Figure 2.

It is worth noting that we are using an OTP only known to a TEE and B to encrypt the outcome sent back to B. This will serve as a nonce to prove the freshness of the session, and it will prevent an adversary from knowing the outcome of the authentication even if a bad certificate is present and

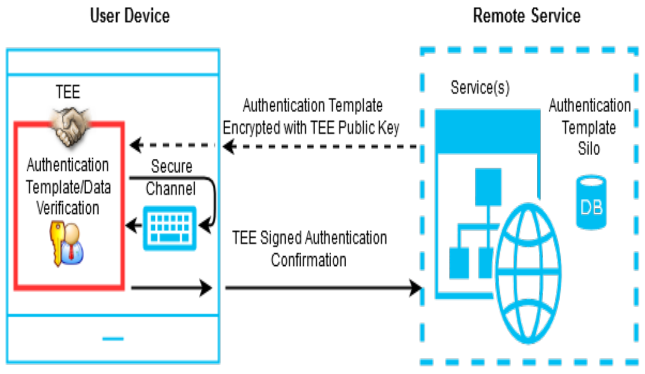
User Device
Remote Service
TEE
Authentication Template/Data Verification
Secure Channel
Authentication Template Encrypted with TEE Public Key
TEE Signed Authentication Confirmation
Service(s)
Authentication Template Silo
DB

Figure 2: Abstract authentication message flow.

DNS poisoning is in progress. However, if we were to use a bad public key for B with its private key known to the adversary, they will not be able to see or know the authentication *template/data* but they will know the outcome of the authentication. A real life protocol is, of course, not as simple as the one presented above: for example, it is important for the service provider to have the integrity and identity of the trusted applications verified before sending it the authentication *template*. It is crucial that the service provider knows the public key came from the intended application within a TEE. A technical report detailing a full message sequence diagram for the architecture can be found online [3].

6.1 Available Hardware Primitives

The aim of this section is to demonstrate the feasibility of implementing the proposed architecture without limiting it to a particular technology or point in time.

All the main chip manufacturers support TEE implementation: Intel SGX, ARM TrustZone (TZ), and AMD Platform Security Processor (PSP).

ARM TZ splits the system into two different worlds: secure and normal [35]. TEE is implemented in the secure world of TZ. Meanwhile, SGX creates 'enclaves', a form of user-level TEE [2]. PSP is similar to TZ, since it uses an integrated coprocessor next to the AMD64 cores, and it has an ARM processor with TZ security extension as a dedicated micro-controller.

Based on this, our proposed system has the potential of being implemented on most consumer electronic devices, such as PCs, laptops or smartphones.

One good candidate currently in the market to implement such system would be the Samsung Knox. It is an Android OS based solution, designed to enhance security through secure containerisation technology [27]. The technology is implemented on various types of Samsung handsets currently available which have ARM TZ support in the ARM Cortex-A processors series. Samsung Knox makes use of TZ through Trustonic Technologies that resides in the secure world of TZ.

Samsung Knox provides the security features required for the proposed architecture, including remote attestation, key generation, isolated execution, and Trusted Path. It makes use of the Trusted User Interface specification by Global Platform and is implemented by the trusted OS, Kinibi, residing in ARM TZ [27].

6.2 Challenges

There are a number of challenges that need to be considered when the proposed architecture is implemented.

These are not necessarily introduced by this architecture however they are partially orthogonal to this work and are being addressed by other researchers.

The main challenge is to enable the user to identify or authenticate the trusted application within the device they are using to differentiate it from an impostor. This is a well-known, difficult challenge [20, 22] with early work addressing some of the issues described in the early 199s [13, 14], however it is largely orthogonal to the issues described here. For example, it is possible to have a coloured LED (similarly to how an active webcam LED is lit) indicating when an application is executed in the TEE and presented to the user through a Trusted Path [37]. This will enable the user to identify when a trusted application is executed within a TEE and it is using a Trusted Path to interact with the peripheral. However, this does not indicate that the running trusted application is the intended application.

To solve the issue of authenticating the application on a device owned by the user, the user can enrol the trusted application by providing a visual cue. The first time a user opens a trusted application, they are prompted to draw something unique on the screen or upload a unique picture; each time the application is opened in the future, the user is presented with the same drawing or image. Since the image is stored within the TEE and only displayed using a Trusted Path when the application is executed, malicious code will not be able to view the image and later spoof the user. This will address the challenge for sole users of the device used for authentication.

However, this issue is more challenging when considering shared devices. One solution would be for the user to select a random image and store it on the remote device [21]. Whenever the user wishes to authenticate, the user selected image, along with a number of other images or part of that image is presented to the user. This way, the user can be sure that the request came from the intended trusted application which their identity must have been validated by the service provider. However, this solution has the apparent flaw that an attacker attempting to brute force an account might be able to identify the user's image after a number of failed attempts. Therefore great attention must be paid to ensuring that the image is not easily replicated or obtained by an attacker. This approach is similar to how graphical passwords work [31]. It is worth noting that this approach will also address fake clone attacks. A well-resourced adversary capable of creating a fake look-alike device will not be able to deceive the remote device by claiming it is a legitimate TEE in order to receive the graphical password. This will enable the user to identify fake cloned devices. Nonetheless, this approach will put extra burden on the user and introduce extra overhead due to the extra data that needs to be transferred. One solution would be to have this as a non-mandatory option for users.

It is worth noting that we have addressed issues arising from malware through the use of a Trusted Path. The remote party is assured that a physical entity has interacted with an authenticating interface to provide the authentication *data*. However, the physical entity does not necessarily have to be human or the intended person, but this does raise the cost of brute force attacks and limit their capability.

Similar to EKE in our architecture, the salt of a *template* has to be sent to the user device (the TEE) by the service provider. Nevertheless, the salt or extra data does not have to be sent in clear text. It is encrypted using the TEE public key and it can only be decrypted, as well as used, within the TEE.

Finally, capturing the authentication *template* in transit is possible, but it will always be encrypted using asymmetric cryptography. An attacker in control of the local network will not be able to weaken the encryption method used for the authentication *template* or *data*. Using asymmetric cryptography, it should not be feasible to decrypt the *template* even if captured. An active man-in-the-middle using a legitimate TEE will not be able to view the authentication *template*. The template is only sent after the attestation of the TEE identity and its running configuration has been provided. Therefore the template can only be decrypted and used within a TEE, and it cannot be exported or displayed to the outside. The user will also be using a trusted input, meaning the authentication *data* provided by the user will be obtained directly by the TEE, defeating man-in-the-middle/malware.

7. CONCLUSIONS

Our main contributions presented in this paper are: a novel architecture that addresses a number of difficult issues and a protocol designed to implement the architecture. We have also identified currently accessible technologies to implement such architecture.

The proposed architecture can enhance the security of authentication *data* by addressing issues related to user authentication of remote servers, rendering bad SSL certificates and DNS poisoning useless, and addressing issues related to the presence of malware attempting to steal authentication *data*. Preliminary formal analysis was carried out on the protocol design and no security weaknesses were found, but more work on fully formalising the adversary models has to be done.

Although the proposed system addresses some difficult security challenges, it is not however a silver bullet and there are a number of issues that need be considered when implementing this architecture. Even without these challenges being fully addressed, the architecture is still valuable.

Future work seeks to implement the proposed approach and analyse it.

8. REFERENCES

[1] Apple. iOS Security. Technical Report May, Apple Inc, 2016.

[2] A. Atamli-Reineh and A. Martin. Securing Application with Software Partitioning: A Case Study Using SGX. In B. Thuraisingham and V. Wang, XiaoFengand Yegneswaran, editors, *Proceeding of the 11th International Conference on Security and Privacy in Communication Networks*, SecureComm '15, pages 605–621, Dallas, TX, USA, Oct 2015. Springer International Publishing.

[3] R. Balisane. The introduction to TEEBAG. Technical Report 10/16, Oxford Research Archive, 2016.

[4] S. M. Bellovin and M. Merritt. Encrypted Key Exchange : Password-Based Protocols Secure Against Dictionary Attacks. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, SP '92, pages 72–84, Oakland, CA, USA, May 1992. IEEE.

[5] S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-based Protocol Secure Against Dictionary Attacks and Password File Compromise. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 244–250, Fairfax, Virginia, USA, Nov 1993. ACM.

[6] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell. The End is Nigh: Generic Solving of Text-based CAPTCHAs. In *Proceedings of the 8th USENIX Workshop on Offensive Technologies*, WOOT '14, San Diego, CA, USA, Aug 2014. USENIX Association.

[7] C. P. Cahill, J. Martin, M. W. Pagano, V. Phegade, and A. Rajan. Client-based Authentication Technology: User-centric Authentication Using Secure Containers. In *Proceedings of the 7th ACM Workshop on Digital Identity Management*, DIM '11, pages 83–92, Chicago, Illinois, USA, Oct 2011. ACM.

[8] L. Chen, J. Franklin, and A. Regenscheid. Guidelines on Hardware-Rooted Security in Mobile Devices (Draft). Technical Report SP 800-164, National Institute of Standards and Technology (NIST), Oct 2012.

[9] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In G. Danezis and P. Golle, editors, *Proceedings of the 6th International Workshop on Privacy Enhancing Technologies Revised Selected Papers*, PET '06, pages 20–35. Springer Berlin Heidelberg, Cambridge, UK, Jun 2006.

[10] H. Crawford, K. Renaud, and T. Storer. A framework for continuous, transparent mobile device authentication. *Computers & Security*, 39, Part B:127–136, Nov 2013.

[11] T. Dinev. Why Spoofing is Serious Internet Fraud. *Communications of the ACM*, 49(10):76–82, Oct 2006.

[12] J. Ehrensvärd and J. Kemp. FIDO U2F HID Protocol Specification. https://fidoalliance.org/specs/fido-u2f-v1.0-nfc-bt-amendment-20150514/fido-u2f-hid-protocol.html, 2015. Accessed: 2016-03-22.

[13] J. Epstein, J. McHugh, R. Pascale, H. Orman, G. Benson, C. Martin, A. Marmor-Squires, B. Danner, and M. Branstad. A prototype B3 trusted X Window System. In *Proceeding of the 7th Annual Computer Security Applications Conference*, CSAC '91, pages 44–55, Dec 1991.

[14] J. Epstein and R. Pascale. User Interface for a High Assurance Windowing System. In *Proceeding of the 9th Annual Computer Security Applications Conference*, CSAC '93, pages 256–264. IEEE, Dec 1993.

[15] D. Florencio and C. Herley. A Large-scale Study of Web Password Habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, Banff, Alberta, Canada, May 2007. ACM.

[16] K. Gillick. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide.

https://www.globalplatform.org/mediaguidetee.asp, 2012. Accessed: 2015-05-11.

[17] E. Grosse and M. Upadhyay. Authentication at Scale. *IEEE Security and Privacy*, 11:15–22, Jan 2013.

[18] T. Holz, M. Engelberth, and F. Freiling. Learning More about the Underground Economy: A Case-Study of Keyloggers and Dropzones. In M. Backes and P. Ning, editors, *Proceedings of the 14th European Symposium on Research in Computer Security*, ESORICS '09, pages 1–18, Saint-Malo, France, Sep 2009. Springer Berlin Heidelberg.

[19] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing Forged SSL Certificates in the Wild. In *Proceedings of the IEEE Symposium on Security and Privacy*, SP '14, pages 83–97. IEEE, May 2014.

[20] M. Lange and S. Liebergeld. Crossover: Secure and Usable User Interface for Mobile Devices with Multiple Isolated OS Personalities. In *Proceeding of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 249–257, New Orleans, Louisiana, USA, Dec 2013. ACM.

[21] B. Laurie and A. Singer. Choose the Red Pill and the Blue Pill: A Position Paper. In *Proceedings of the Workshop on New Security Paradigms*, NSPW '08, pages 127–133, Lake Tahoe, California, USA, Sep 2008. ACM.

[22] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-k. Chu, and T. Li. Building Trusted Path on Untrusted Device Drivers for Mobile Devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, pages 1–7, Beijing, China, Jun 2014. ACM.

[23] R. Lindemann, D. Baghdasaryan, and E. Tiffany. FIDO UAF Protocol Specification v1.0. https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-protocol-v1.0-ps-20141208.html, 2014. Accessed: 2015-01-28.

[24] A. Martin. The ten-page introduction to Trusted Computing. Technical Report RR-08-11, Oxford University Computing Laboratory, 2008.

[25] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Eurosys '08, pages 315–328, Glasgow, Scotland UK, Apr 2008. ACM.

[26] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping Trust in Commodity Computers. In *Proceedings of the IEEE Symposium on Security and Privacy*, SP '10, pages 414–429. IEEE, May 2010.

[27] Samsung. White Paper: An Overview of Samsung KNOX. Technical report, Enterprise Mobility Solutions Samsung Electronics Co., Ltd., Jun 2013.

[28] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The Emperor's New Security Indicators. In *Proceeding of the IEEE Symposium on Security and Privacy*, SP '07, pages 51–65. IEEE, May 2007.

[29] F. Stajano. Pico: No More Passwords! In B. Christianson, B. Crispo, J. Malcolm, and F. Stajano, editors, *Proceedings of the 19th International Workshop on Security Protocols XIX*, volume 7114, pages 49–81, Cambridge, UK, Mar 2011. Springer Berlin Heidelberg.

[30] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th USENIX Security Symposium*, pages 399–416, Montreal, Canada, Aug 2009. USENIX Association.

[31] X. Suo, Y. Zhu, and G. S. Owen. Graphical Passwords: A Survey. In *Proceeding of the 21st Annual Computer Security Applications Conference*, ACSAC'05, pages 10 pp.–472, Tucson, AZ, USA, Dec 2005. IEEE.

[32] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune. Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me? In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST '12, pages 159–178, Vienna, Austria, Jun 2012. Springer Berlin Heidelberg.

[33] J. Vykopal. A Flow-Level Taxonomy and Prevalence of Brute Force Attacks. In *Proceedings of the First International Conference on Advances in Computing and Communications (Part II)*, ACC '11, pages 666–675, Kochi, India, Jul 2011. Springer Berlin Heidelberg.

[34] W3C. W3C Web Security Context Wiki - Shared Secret Trusted Path. https://www.w3.org/2006/WSC/wiki/RobustSharedSecret, 2016. Accessed: 2016-06-15.

[35] J. Winter. Experimenting with ARM TrustZone Or: How I met friendly piece of trusted hardware. In *Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications*, TrustCom '12, pages 1161–1166, Liverpool, United Kingdom, Jun 2012. IEEE.

[36] J. Yan and A. S. El Ahmad. Usability of CAPTCHAs or Usability Issues in CAPTCHA Design. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, pages 44–52, Pittsburgh, Pennsylvania, USA, Jul 2008. ACM.

[37] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune. Building Verifiable Trusted Path on Commodity x86 Computers. In *Proceedings of the IEEE Symposium on Security and Privacy*, SP '12, pages 616–630, San Francisco, CA, USA, May 2012. IEEE.