Offline Trusted Device and Proxy Architecture based on a new TLS Switching technique

Denis Migdal
Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC
14000 Caen, France
denis.migdal@ensicaen.fr

Christian Johansen
Institute for Technology Systems
University of Oslo
Oslo, Norway
cristi@ifi.uio.no

Audun Jøsang
Department of Informatics
University of Oslo
Oslo, Norway
josang@ifi.uio.no

***Abstract*—Infection of client devices poses a significant threat to secure user authentication. Combining vulnerable client devices with special security devices, as often the case in e-banking, can increase significantly the security. However, these often incur usability hurdles. This paper describes a new architecture where an untrusted proxy on the client device communicates both with server applications, and a trusted application running on a trusted device. The proxy switches between two TLS channels, one from the client application, and another from the trusted device. The result is a highly usable and flexible architecture with strong security assurances which, moreover, is transparent to the client or server applications, thus allowing it to be deployed in existing systems. We have implemented a PoC (available open source) and demonstrated it using the OffPAD device. Various applications of our architecture can be imagined, some of which we present in the end of the paper, applicable to web services and IoT systems.**

***Keywords*-Authentication and authorization; Hardware security; Security architectures; Usable security; TLS switching; Offline Trusted Device Proxy Architecture; Trusted device; Secure Proxy; IoT; Web authentication;**

I. Introduction

Server applications (server apps) typically require user authentication based on user credentials, i.e., only registered, authorized and authenticated users are granted access to services. However, client devices are often infected with malware,[1] e.g., false certificate installation could facilitate phishing attacks [29] or a keylogger [41] could be installed to intercept user input s.a. user credentials, and allow false authentication to server apps. Existing solutions for protecting online transactions with strong security typically involve an external trusted device, e.g., for the generation of OTPs (One-Time Password) in e-banking.

The first author was partially supported by the project OffPAD with number E!8324 part of the Eurostars program funded by the EUREKA.

The second author was partially supported by the project DiversIoT funded by the Norwegian Research Council.

The third author was partially supported by the project Oslo Analytics funded by the IKTPLUSS program of the Norwegian Research Council.

[0]This paper is accompanied by the online technical report [38].

[1]According to PandaLabs' 2015 estimates, a third (32.13%) of the world's PCs are infected with some sort of malware, of which the majority (60.30%) are Trojans.

A common architecture is to put a secure intermediary between the client applications (client apps) and the server apps, e.g., the SSL proxy concept [21] or the Bitdefender BOX[2] in Internet of Things. We call this a *trusted proxy*. Such a trusted proxy is transparent to both the client apps and the server apps, and is assumed secure. All data passes through the trusted proxy, allowing it to read, log, modify, or suppress transmitted data.

Note that the trusted proxy technically can perform actions without the user's approval or knowledge, but it is assumed (trusted) not to do so. Usually the trusted proxy is deployed on a trusted device as a black box, and therefore, the users cannot ensure the real behaviour of the trusted proxy. Thus users have to trust the provider of the trusted device. There is also no absolute guarantee that a trusted device will not fall pray to MitM (Man-in-the-Middle) attacks on the communication from the client device. Moreover, trusted proxies are unable to perform exhaustive verifications due to the usually limited capabilities (s.a. speed of network interfaces) of the trusted device on which they are deployed. Thus the use of trusted devices should be limited in practice.

An architecture that would limit the use of trusted devices is to let client apps communicate directly with server apps, and only query a trusted application (trusted app) for sensitive information. The communications are offline as defined in [37], "follow[ing] controlled formats, during short and restricted time periods, not involving wireless broadband capabilities". We call such a trusted device an *offline trusted device (OTD)*. Since communications from the trusted app to server apps goes through the (possibly infected) client device, any sensitive information would need to be protected. This concept of using a separate trusted device for sensitive computations is used by HTTP XDAA [25], an extension of the standard HTTP DAA [20] where the credentials and the challenge-response computations are handled by a separate trusted device.

However, in such an architecture, the trusted app does not remain transparent to neither the client apps nor the server

[2]http://www.bitdefender.com/box/

apps. This requires adaptations of existing client apps, or server apps, in order to use the trusted app. Such adaptations (writing, installing, and maintaining specific code), are costly and are detrimental to the adoption of the architecture.

The trusted device may be transparent to server apps, meaning that the trusted device handles, in place of client apps, sensitive security computations that must not be exposed to a possibly infected client device. However, client apps, must be especially programmed to use the trusted device, usually in form of plugins. The trusted device is then just an extra security feature, e.g., MP-Auth [32] encrypts a password with the public key of the server app, for which it uses a trusted device, but does not require one, i.e., client apps knowing the password are able to respond to the server app without interacting with the trusted device.

Alternatively, the trusted app may be known to the server app and used in a protocol designed to require a trusted device. Banking applications are such examples [1], where the trusted device holds a secret, for example a private key specific to the user and the trusted device. In this case the server app needs to be configured when provisioning trusted devices to users.

The idea of obtaining usable authentication using an offline and private trusted device has been described and motivated in [24], which argued for managing user credentials on such a device. Here we present an architecture and a prototype implementation that combines the above described traditional architectures, i.e., using a proxy on the client device and a peripheral OTD.

A. Contributions

- We present a new architecture called Offline Trusted Device and Proxy (OTDP) architecture in Section II.
- The OTDP architecture is based on a novel use of TLS, which we call *TLS Switching* and describe in Section III, involving two interchangeable channels for communication between the server app respectively the trusted app and the Proxy. We compare in Section III-C our TLS Switching technique with related works, i.e., with *TLS handshake proxying* [44], *TLS splitting* [31], and *TruWalletM* [11].
- The OTDP architecture can be applied for various authentication scenarios, some given in Section IV, for the Web domain in Section IV-A and for the IoT domain in Section IV-B.
- We also discuss in Section V *security aspects* of our proposal, as well as *usability aspects* in Section VI.

Our work focuses on user-side security, usability, ergonomics, and *transparency* of the used software and hardware for both the client apps and the server apps.

II. The Offline Trusted Device and Proxy architecture

We consider a traditional client-server architecture where it is assumed that the client device is infected with malware. Server apps on the other hand are assumed to be secure.

Existing protocols, as well as existing trusted devices, can be used on top of the OTDP architecture, enabling the use of a trusted device without the previously discussed drawbacks of existing architectures. For applications with high security requirements, such as e-banking, eHospitals, or various IoT applications in Smart Infrastructures, it is possible to use special hardware-enabled security solutions [28]. This can be a transportable device pluggable into the client device, like the OffPAD [37], or a secure part of the client device s.a., a secure element, a TPM (Trusted Platform Module) [5], [12], or a Trusted Execution Environment[3] like Intel's SGX [2], [35], [43].

A. The OTDP architecture

We propose the Offline Trusted Device and Proxy (OTDP) architecture which combines a *Proxy* that is sitting on the untrusted client device, and communicates with a *trusted app* running on an offline trusted device, holding sensitive information like user credentials. This can be seen as a combination of the positive aspects from the two classical architectures mentioned in the introduction.

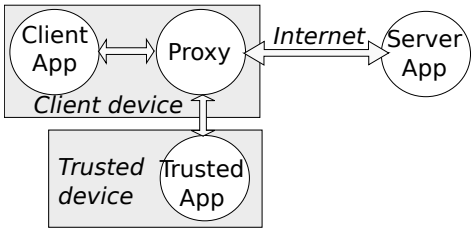

Figure 1. Offline Trusted Device and Proxy (OTDP) Architecture.

Client apps communicate with server apps through the Proxy. The implementation of the Proxy, based on our TLS Switching, makes sure our OTDP architecture is transparent to both the client apps and the server apps, while still allowing to read, log, modify, and suppress exchanged data. The trusted device is able to communicate with the server app only through the proxy which controls the communication format and the time when the trusted device is used. Thus the trusted device is assumed to be an OTD. The Proxy is also assumed to be open-source, so the user can read and modify the sources. Hence the Proxy is a white box. Thus spying on the user or doing actions without the user's approval is harder for the provider of the trusted device. More important, there is no false feeling of security since the Proxy is also considered untrusted as it runs on the client device.

[3]http://www.globalplatform.org/specificationsdevice.asp

Sensitive information is retrieved, computed, stored, or processed by the trusted app upon its initiative, or upon requests from the Proxy which, without the knowledge of the sensitive information, cannot correctly answer queries from the server apps.

To protect the sensitive information stored and maintained by the OTD, we develop the concept of *TLS Switching*, presented in Sec. III, which does not require specific algorithms or protocols on neither client apps nor server apps. We only need the Proxy installed on the client device.

B. Design decisions and implementation of the Proxy

The Proxy was designed to be extensible through the use of filters and commands, making it easier, for example, to add support for a new communication protocol.

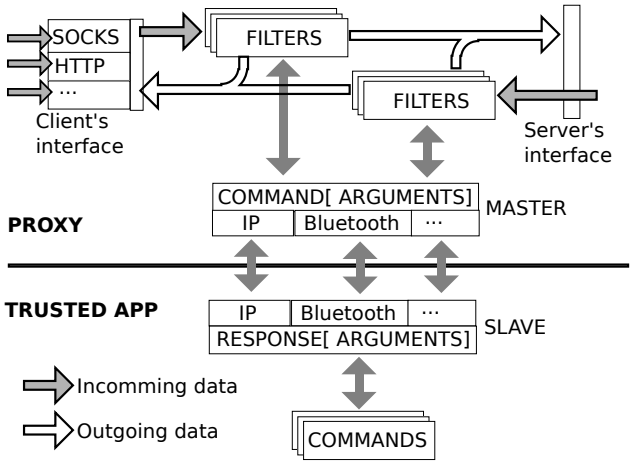

Figure 2. OTDP Proxy software architecture

The following design requirements on the Proxy were followed in the implementation (link to source code is provided in footnote on page 5).

- The communication interface towards client apps (called *Client's interface* in Figure 2) offers several proxy protocols such as SOCKS [30] or HTTP Proxy [19]. SOCKS v5 supports TCP and UDP streams. HTTP Proxy only supports HTTP streams. For client apps non-compliant with proxy protocols, tools can be used to "socksify" them.[4] Static redirections can easily be offered as well.
- The client interface allows trivial insertion, substitution and deletion of proxy protocols, making the proxy architecture easy to extend.
- The communication interface towards server apps (*Server's interface* in Figure 2) is configurable to use a network proxy if the client device sits behind one.
- The communication interface towards the trusted app has two-layers. The first layer encapsulates the communication medium, and the second abstracts the format of the messages.
  - The first layer allows trivial insertions, deletions and substitutions of communication mediums such as USB, WiFi, Bluetooth, or NFC.
  - The second layer uses a slave/master communication: the master is the Proxy who sends a command and the slave (the trusted app) must immediately respond. The slave/master communication makes our OTDP architecture compliant with slave/master mediums, such as ISO7816-compliant smartcards, like MasterCard's Display Card[5].
- Incoming data from client apps or server apps are processed by a set of *Filters* in Figure 2. Thus, by adding or removing a filter, features can be added or removed from the Proxy. If needed during the processing of messages, a filter can request information or services from the trusted app via a command, such as asking the trusted app to send sensitive information to the server app.
- The trusted app listens to a set of *Commands*. Similarly, by adding or removing a command, features can be added or removed from the implementation of the trusted app or adapt to specific hardware.
- The implementation of the Commands follows a command design-pattern, whereas Filters follow the chain of responsibility design-pattern [22, Chap. 5].
- Commands and responses are human-readable to facilitate debugging operations. In consequence, a human may monitor and understand the communication between the Proxy and the trusted app with tools such as Wireshark[6], or to simulate the Proxy (or the trusted app) to test the trusted app (respectively the Proxy) with tools such as netcat or openssl.[7] Messages start with an ASCII command-name (or response-status) followed by optional arguments. Response-status uses the HTTP status-codes [19].
- Theses communications are configurable via an XML configuration file.

Ideally, the trusted device should be integrated into an object already carried by the user, e.g., as the OffPAD [37] which is a phone back-cover OTD connected through microUSB to a smart phone. In case multiple users are using the same device, or the same user uses it in different contexts, the secure external device also provides specific user profile management.

[4]http://linux.die.net/man/1/socksify

[5]http://newsroom.mastercard.com/press-releases/mastercard-introduces-next-generation-display-card-technology-a-first-for-singapore/

[6]https://www.wireshark.org/

[7]nc(1) / s_client(1) – Linux man page, http://linux.die.net/man/1

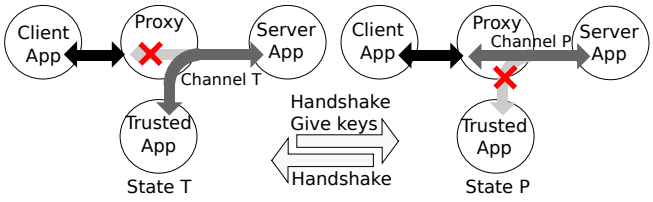

Figure 3. TLS Switching visualization: Proxy changing between two TLS channels T and P.

III. TLS Switching

A. Motivation

OTDP requires two TLS channels of communication with the server app: one from the Proxy (channel P) and one from the trusted app through the Proxy (channel T). Since the Proxy resides on the untrusted client device it must be unable to read or write on channel T. Since we want OTDP to be compliant with traditional client-server architectures, the channels should be seen as one on the server app (see Figure 3). To the authors' knowledge, no existing protocol enables to do so, and as we explain in Section III-C, close related technologies fail to meet our goals.

B. Description

Our use of TLS (called TLS Switching) works in two modes (or states of operation): state T (trusted app) and state P (Proxy). In both states the TLS handshake and record protocols are used [15].

In state T the channel T is used to exchange sensitive information with the server app. State T is reached when the trusted app performs a renegotiation [16] to renew the TLS session keys and initialization vectors (IVs). Due to TLS's forward and backward secrecy [26], the Proxy cannot deduce keys and IVs from the ones it knows. Thus the Proxy is not able to read nor write records on the channel T during the state T.

In state P the channel P is used to transmit, in a more efficient way, information between the server app and the client app. State P is reached when the trusted app renegotiates the TLS session keys and IVs, and passes them to the Proxy, enabling it to read and write records though a standard TLS connection.

Transparency: TLS Switching operates as follows (shown in Figure 4). A secure connection pre-exists between the trusted app and the Proxy (0), e.g., through microUSB in the case of the OffPAD. The client app tries to establish a TLS connection with the server app, but is connected to the Proxy (1). Optionally, the Proxy might send information to the trusted app (s.a. the server's domain name, or the TLS extensions used by the client app) (2a). The Proxy then establishes a TCP connection with the server app (2b) over which the trusted app establishes a TLS connection (3). The TLS connection is then in state T, ready to exchange

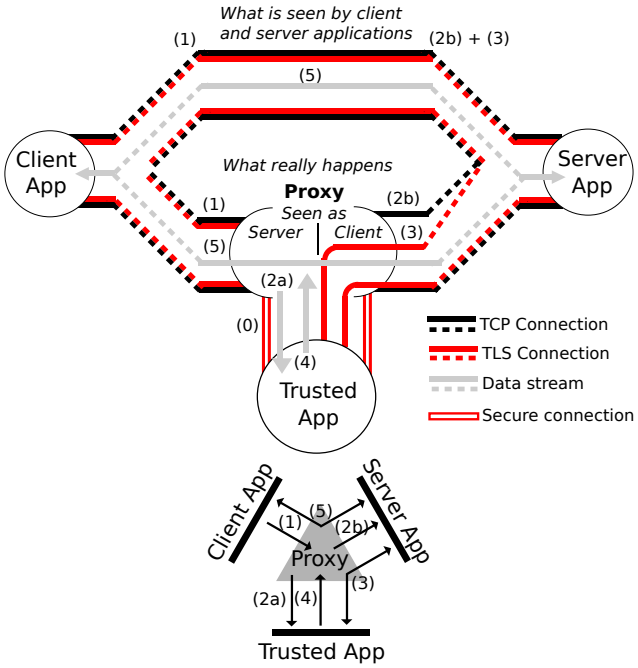

Figure 4. Establishment of TLS Switching connections.

messages between the trusted app and the server app, or to go into state P (4) to allow the client app to communicate with the server app (5).

TLS Switching requires:

- the server app to accept TLS renegotiations [16], which is rather common;
- the Proxy to support the cipher suite and TLS extensions [8] used in state P *(by our implementation)*;
- the trusted app to verify TLS certificates *(by design)*;
- the trusted app and the Proxy must be able to exchange the current cipher suite, symmetric keys and IVs in a secure way *(we assume this by construction)*.

C. Related work

TLS Handshake Proxying [44] is a technique used with *edge servers* that cache content to reduce the connection latency between a client and a server, i.e., the client establishes a TLS connection to a local edge instead of the distant server. However, the edge is not trusted to share the server's private key. In consequence, during the local TLS handshake the edge will forward to the server the message involving the private key, i.e., only one long-distance communication.

This TLS Handshake proxying technique may be used in our architecture, but only when in state P, as our proxy cannot generate TLS records during state T. Nevertheless, this requires to modify parts of TLS libraries, and makes the trusted app and the Proxy interdependent.

SSL Splitting [31] uses a proxy to split a non-encrypted SSL connection into two streams, one for the content and one for the content's integrity. The client establishes a

non-encrypted SSL connection with the server through the proxy which forwards all messages. The server answers with the Message Authentication Code (MAC) of the content and a content identifier which the proxy replaces with the cached content before sending the SSL record to the client. For encrypted connections [31] proposes a work-around by sending the symmetric encryptions keys and IVs to the proxy so that it can read SSL records, and write records knowing the MAC. Thus, the proxy is trusted for confidentiality, but not for integrity.

In our case, the Proxy needs to be able to build complete TLS records when in state P, but is not trusted to read or write when in state T, i.e., when the channel T is used.

TruWalletM [11] presents a similar approach to OTDP and TLS Switching. Instead of TLS renegotiation, TLS resumption is used implying, roughly speaking, to close and open TCP connections at each state change. This is well suited for communications using TCP connections lasting only the time the server app responds to a client app query, s.a. in HTTP1.0, AJAX. However, this is less suited when the TCP connection aims to gather many queries and responses, s.a. in HTTP2.0, or when the communication is a stream, s.a. in WebSockets, often closed when the underlying TCP connection is closed.

Moreover, the trusted device is required to establish each connection, which might lead to privacy concerns as the trusted device would be then able to log user's browsing history. This breaks one of our main motivations for OTDP. Also, TruWalletM does not consider the possibility to receive sensitive information from the server app, or to propose additional trusted device-specific features to compliant server apps, s.a. authentication of the trusted device by the server app, which we do.

TruWalletM also requires a Trusted Execution Environment (TEE) whereas our proposed architecture is compliant not only with TEE but also with generic trusted devices, even those having limited UI allowing non-confidential sensitive information inputs and notifications on the client device. Moreover our Proxy enables easy implementation of additional features, not necessarily requiring a Proxy and TLS Switching, s.a. OTPs or password auto-typing for local authentications on the client device.

D. TLS Switching implementation

TLS is implemented by various libraries such as GnuTLS[8] or OpenSSL[9] in C, and JSSE[10] in Java. We implemented a proof of concept of the TLS Switching and OTDP architecture in Java v1.7 with OpenJDK's TLS engine (see source code[11]).

[8]http://www.gnutls.org/

[9]https://www.openssl.org/

[10]http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

[11]https://github.com/denis-migdal/OTDP-and-TLS-Switching-PoC

The Proxy and the trusted app exchange messages through TLS records over TCP. In state T, the Proxy simply forwards messages from the trusted app to the server app, and from the server app to the trusted app. In state P, the Proxy should only process application data messages, and forward the rest to the trusted app.

We use two new types of clear messages, enabling the Proxy and the trusted app to exchange information when changing state:

- *'t' messages* sent by the Proxy to the trusted app in order to go into *state T*. Theses messages are *followed by a renegotiation* performed by the trusted app.
- *'p' messages* sent by the trusted app to the Proxy in order to go into *state P*. Theses messages are *preceded by a renegotiation* performed by the trusted app.

The *p* messages contain the current TLS parameters required to partially build a TLS engine on the Proxy: the cipher suite identifier, the TLS protocol version, and the symmetric keys. This allows the Proxy to build and read TLS records without exposing the trusted app private keys.

To return into state T, as the internal state of the Proxy TLS engine is modified after reading and writing records, the trusted app's TLS engine has to be updated through *t* messages before performing a renegotiation. This only requires two sequence numbers, incremented for each record sent or received under the current keys.

The *p* messages do not have to contain sequence numbers as they can be deduced by the Proxy. Moreover, *p* and *t* messages do not have to contain initialization vectors. Indeed, RC4 cipher suites are prohibited in TLS 1.2 [40] thus cipher suites only offer block ciphers (AES with CBC or 3DES with CBC) or NULL cipher. Yet, in block ciphered messages, initialization vectors are randomly chosen for each record and are included in the record's message.

1) Implementation discussions:

Our implementation allows the Proxy and the trusted app to use different TLS libraries, thus making them more independent. In particular, we modify classes on the fly to extract symmetric keys when renegotiating.

With our implementation we tried to avoid modifying and rebuilding TLS libraries, as this would need to be repeated at each library update. Also, we avoid re-implementing a whole TLS layer, which could lead to several security threats due to implementation errors or slow reactivity.

To avoid performing TLS renegotiations when not required, the Proxy performs the initial handshake in place of the trusted app. Thus, TLS connections start in state P and do not require TLS Switching as long as no sensitive information are exchanged, or expected to be exchanged. Then, to go into state T for the first time, the Proxy sends a *p* (with additional content) instead of a *t* message to the trusted app.

As an alternative, instead of exchanging ciphers information, the Proxy and the trusted app could cooperate to

perform a renegotiation. One could generate and read clear renegotiation messages and the other encrypt and decrypt them with the current keys. Such a feature would cost at least one renegotiation to go into state T, and two (asking to encrypt/decrypt a renegotiation message and receiving the result) to go into state P. We tested such an alternative implementation which proved to be more costly than our current implementation.

IV. Applications of the OTDP and TLS Switching

Two applications of OTDP have been demonstrated at [37] using the OffPAD device [46], namely: cognitive (i.e., by the human user) server authentication based on the petname system [18], and user authentication based on the extended challenge-response protocol XDAA [25] involving the trusted device. Other possible applications of the OTDP architecture and the TLS Switching are described below (and detailed in the technical report [38]).

A. Application in web systems

1) Server app authentication: OTDP strengthens server apps' authentication by verifying their certificates on the trusted device, thus preventing attacks by dubious certificates injected in client apps. Moreover, using a petname system and protocol (e.g., as implemented on the OffPAD) we can achieve cognitive server app authentication, i.e., where the user is actively aware of the identity of the server apps. Whitelists and blacklists on the trusted device, and similarity checks are used as an attempt to counter phishing attacks.

2) User authentication: The trusted device OffPAD allows user authentication through fingerprint biometrics. This can be used to unlock user credentials in various situations. We have demonstrated the use of user credentials stored on the trusted device for the XDAA authentication protocol.

3) Identifying messages: The Proxy is used to identify messages from client or server app matching a specific template. The Proxy, ignorant of the sensitive information, is unable to process such messages, and thus forwards them to the trusted app. The trusted app, might then force the connection into state T and send, after user confirmation, a reply to the server app. Such a reply is built from sensitive information stored (or inputed by the user) on the trusted device. Instead of asking them from the user, sensitive information can also be generated by the trusted app, s.a. strong passwords during registrations.

Identifying client apps' messages before sending them to the server apps could be coupled with whitelists and blacklists stored in the trusted device to block unwanted messages. Examples include blocking HTTP requests to phishing websites, or even HTTP requests to advertising servers.

We use the identification of server apps' messages in the HTTP XDAA. However, some client apps might block until users provide the sensitive information directly in a form

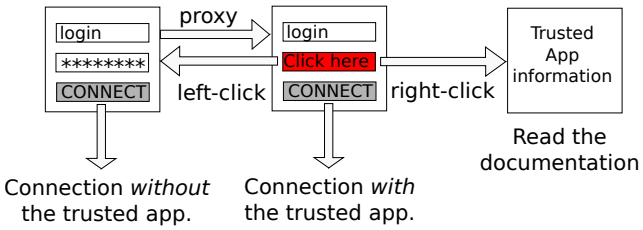

Figure 5. Modifying a login form to inform and enforce user behaviour.

inside the client app. In such cases, users have to fill the required fields with false sensitive information to submit the form without exposing sensitive information on the client device. The usage of client apps' messages identification would then be more appropriate.

4) Trusted app authentications: The standard TLS handshake protocol offers client and server authentications. Server apps may authenticate the trusted app using a handshake with client authentication. In this case the trusted device stores a private key or computes one from user information, such as passwords or biometric data. The public key may be certified by the trusted device's provider, given during registration on the server app, or trusted on the first usage. The trusted app may propose other authentication methods through TLS extensions [8].

The trusted app must ensure the origin and destination of sensitive information by, e.g., authenticating server apps. For this, the trusted app checks the server apps' certificate and searches the requested IP and/or domain name in its blacklist and its whitelist. In order to prevent phishing attacks, the user can be warned by the trusted app when requesting for a domain-name syntactically too close to one from the whitelist. DNSSEC [3], [7] may also be used by the trusted app to prevent attacks against the DNS protocol [4].

5) Modifying web pages: The Proxy is used to improve ergonomics by adding warnings or information about the OTDP usage and features, and to enforce desired user behaviour, e.g., see Figure 5. We can prevent the user from filling in password fields on the untrusted client device. When the form is submitted, the Proxy uses *message identification* (as previously described) then asks the trusted app to complete the answer on the user's behalf.

B. Applications in IoT systems

1) Privacy watchdog in Smart Grid metering: Smart Meters are plagued by privacy considerations [17], [33], because the high frequency of real-time energy consumption that they report to the energy provider results in big data that can reveal information like behaviour patterns in the home, what devices are used (coffee machine vs. tea cooker) and when, or even what brand of devices (thus life style and wealth). Solutions often involve putting an intermediary between the meter and the energy provider. The OTDP can be used, with the smart meter as the client app behind the

proxy implemented by Smart Meter manufacturers with the functionality of enforcing various privacy policies that a user can configure. Security critical functionalities can be done on a specific trusted device of the user, e.g., provided by some trusted entity like a governmental body that oversees data protection issues. The trusted device can be in the form of a javacard or secure element inserted in the smart meter.

2) Secure authentication in eHospitals: In hospitals equipped various IT infrastructure (like NFC door openers, smart medicine storage, RFID tagged medical devices, in-door positioning located personnel, smart operation rooms, smart UIs, etc) it is important to have usable authentication that does not interfere with medical duties [6]. The OffPAD phone jacket device [37] can be used by hospital personnel (also given to patients) to put around their personal mobile phones. The OTDP enables to transform the personal phone into a highly versatile authentication device, by having on the mobile phone installed the proxy as a phone app, communicating with various client apps each doing different forms of medical IT functionality, e.g., alarms, search medical devices, open doors, continuous authentication, etc.

3) Secure applications in Smart Transportation: The trusted device can be implemented by a secure hardware like a TPM chip or a processor with functionalities like Intel SGX. These have limited connectivity, i.e., only with the computing unit, therefore can be regarded as offline. In Smart Car/Truck the Proxy and client can be part of the car's software, whereas the OTD can be part of the owner's key or a similar pluggable device. Communication and authentication protocols can be imagined based on OTDP for securing various sensitive communications. Examples include: communication with the smart road infrastructure could be secured with OTDP, or authentication and access to in-car services like road congestion alarm and guidance, or access to car's on-board information while in service.

V. Security considerations

We assume that the client device, and by extension the Proxy, are compromised, allowing attackers to arbitrarily listen and modify I/O interfaces (s.a. network, screen, keyboard). The trusted device, trusted app, server app, and the TLS protocol are assumed to be secure and trustworthy.

Sensitive information might originate from user inputs (s.a. passwords or fingerprints), be stored, or generated, on the trusted device. They must never be exposed on the untrusted client device. However, non-confidential sensitive information might originate from untrusted sources (s.a. the client device) but must be confirmed by the user through the trusted device and not be exposed afterward.

Thus sensitive information must be exchanged exclusively between the trusted app and the server app, i.e., only during state T. Indeed in state P, the Proxy being able to read, modify, and write records, exchanged information would be exposed. In state T, the client device ignores the current session keys, protected by the forward and backward secrecy of the TLS handshake. Moreover, a Proxy cannot enforce state P as it cannot read nor write records in order to perform a renegotiation. In order to avoid phishing attacks, handshake server authentication is asked by, and performed by, the trusted app with TLS certificate verification (see Sec IV-A1) guaranteeing the origin (or destination) of sensitive information during state T.

Although the client device is unable to read exchanged data between the trusted app and the server app, the client device is able to know when sensitive information are exchanged, as well as their length, more or less the cipher block size (or padding length) used.

However, the client device is still able to arbitrarily modify received, and sent, information during state P. Thus the integrity and origin (or destination) of the information printed, or inputed, on the client device cannot be trusted. Although message verifications and general computations are possible, the trusted device is not intended to replace the client device [28].

More problematic, the client device might send unwanted messages during state P, s.a. asking the server app to execute actions once the user is connected, or connecting the user on a different account to make her perform actions on the attacker account (e.g. crediting money on the attacker account). We argue that in such case, the server app is vulnerable to lunchtime attacks [13]. To prevent such attacks, server apps should require user authentication and confirmation for any sensitive action.

OTDP being transparent to server apps, standard server apps have no concept of state T, or state P, thus send sensitive information independently of the current state, and would trust information received both during state T as well as during state P. In a transparent architecture, the trusted app thus has to predict and anticipate when sensitive information might be sent by the server app. Such predictions are not possible in a generic way, and must be performed during state T, as the client device is untrusted.

For an improved security, server apps could be programmed to take into account the current state of the connection while remaining compliant with standard client apps. During a renegotiation initiated by either the trusted app or the server app, the trusted app might offer "State T" and "State P" in the CLIENT HELLO's TLS extensions list, and go into the state requested by the server app's answer (a missing TLS extension meaning that the server app does not support the concept of state T and state P). If the trusted app wants to force the state T, it can offer the "State T" TLS extension alone.

Server apps might as well ask for client authentication during handshake to authenticate the trusted app (Sec. IV-A4). The trusted app might respond with different certificates depending on the requested state, or send an empty certificate for state P. However this could lead to incom-

patibilities depending on the server app's implementation of TLS. Instead, we recommend the usage of an unique TLS certificate independently of the requested state with an indication of the requested state in the CLIENT HELLO's TLS extensions list. The certificate may be certified by the trusted device's provider, trusted at first use, or given during registration on the server app.

Since we assume the communication between the Proxy and the trusted app to be secure from a network attacker, (e.g., through microUSB or NFC), and the client device is considered untrusted, OTDP architecture does not add attack vectors compared to the traditional architectures. Moreover, the communications from server apps, and from client apps, are as in traditional architectures because of the transparency that the Proxy provides. However, new attacks can come from poor design of new protocols that OTDP enables. Therefore, we would encourage the use of formal verification tools and methods when designing security protocols, s.a. ProVerif [10], CryptoVerif [9], Tamarin [42], or Murphi [39] (each being based on different theoretical foundations).

VI. USABILITY CONSIDERATIONS

A strong method of authentication is not always enough if users are not encouraged to use it. Moreover, when interacting with client apps running on a client device which can be corrupted, users should be prevented from revealing their credentials on the client apps. However, if the system lacks ergonomics and is too constraining, users might reject, and bypass, it [14].

The Proxy being located on the client device, which the user controls, and the trusted app mainly responding to queries from the Proxy, the users are more aware of the normal behaviour of the Proxy and, to a certain extent, of that of the trusted app. Actions, s.a. connecting the user to a server app, should not be performed without user's knowledge, for consent as well as for security reasons. Users should thus be able to give consent in full knowledge of the cause, thus limiting required user trust toward the provider of the trusted device.

A. User authentication to the trusted device

The trusted device must verify the user's identity before granting access to its services and allowing the Proxy to interact with the trusted app. This prevents an attacker from using the trusted app if the trusted device is stolen or lost. In our case, the OffPAD device uses biometric fingerprint for user authentication.

Still, if the user is authenticated and the trusted device is lost, then the user should be disconnected from it. This is done automatically by the trusted app when it detects that it cannot communicate with the Proxy, which is possible when the client device on which the Proxy resides is a different physical system than the trusted device. It is less probable that the attacker can obtain both devices, except when they are joined, e.g. like in the case of the OffPAD where the trusted device is a phone-jacket, and the client device is the phone. Alternatively, we could have the user's identity verified regularly through continuous authentication techniques based on biometrics [27], [45].

B. User interaction

Users are warned when the trusted app waits for user inputs or when messages are blocked, by generating notifications. To prevent users from disabling them, notifications must be used with care. At the same time, the Proxy displays explanations or security messages on the client device. This improves the ergonomics, but as the client device might be corrupted, these explanations must be used with care. For a web browser, a notification page is shown to the user, requesting her to check the trusted device. When the user has performed the requested actions on the trusted app, the page is automatically refreshed (using AJAX).

The Proxy also modifies the responses from the server apps to insert fields or informative messages. As shown in Figure 5, the Proxy inserts scripts in HTML pages to disable password inputs. In this way the user is prevented from filling, out of habit, the password input [47]. Alternatively, the user has the option to disable such interferences so to use the input field normally when needed. The Proxy could also be used to let the user input information (or commands) for the trusted app on the client device, which, is more ergonomic, especially if the trusted device has limited user inputs (e.g. no keyboard). The provided information must not be confidential, and must be verified and confirmed by the user on the trusted device.

C. User behaviour

Credentials, s.a. passwords, should not be exposed to users as they might write them on a post-it, or share them with colleagues or with social engineering attackers [36]. Credentials could be generated by the trusted app at registration, and never shown to users. Since these do not need to be remembered, credentials can be safer, with more entropy, while being unique for each server app [23].

Users typically do not change their credentials at reasonable intervals as they should do. Stored credentials and other sensitive information could have an expiration date which when reached would warn the user. With the user's approval, the trusted app could replace outdated information and update them on the server apps. Furthermore, this is not a generic way of doing this since each server app uses its own protocol to update credentials [34].

The user should be warned when the trusted app logs her into a server app; the trusted app should at least ask for user confirmation. Since several identities can be suitable for a given authentication request, the trusted app can ask the user to choose the identity to use among the suitable identities and consider this choice as a confirmation.

VII. Conclusion and Future Work

We have presented a novel architecture that combines the benefits of both having an OTD for helping with secure computations, thus providing stronger security, as well as an untrusted Proxy running on a client device to provide more usability, communicating with the trusted app when needed. This new OTDP architecture is enabled by a novel use of TLS channels, which we call *TLS Switching* because the Proxy and the trusted app collaborate to make two secure channels which they switch between, depending on whether information needs to be transmitted from the client app or from the trusted app. This architecture and the TLS Switching have been implemented in an extensible and open-source PoC. The architecture has also been demonstrated using the OffPAD as the trusted device, connected through microUSB to a Android phone [37]. We have considered (though not formally) the security aspects specific to this new architecture, as well as the usability that we aimed for. Another goal of the OTDP architecture was to be transparent to both client apps and server apps, which is not respected by the related work that we surveyed.

Future work could conduct more detailed user experiments about the usability of the OTDP architecture, whereas the security could be verified using formal methods. More widely used authentication methods than HTTP DAA (s.a. OAuth) may be implemented and tested, although we see no evident obstacle to their usage with OTDP and TLS Switching. Fully transparent client's interface (i.e. not requiring proxy configuration on client apps) may be implemented using TUN/TAP and iptables. OTDP architecture may as well be generalized for the specific needs of a company architecture and a centralized credentials management.

Acknowledgment

We thank reviewers from CCS'16 and IFIP-SEC'17 for helping improve this draft, and OffPAD project members (esp. Léonard Dallot) for their help during the developments.

References

[1] Manal Adham, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis. How to attack two-factor authentication Internet banking. In *Financial Cryptography and Data Security*, volume 7859 of *LNCS*, pages 322–328. Springer, 2013.

[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In 2^{nd} *Int. Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2013.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, RFC Editor, March 2005.

[4] Suranjith Ariyapperuma and Chris J Mitchell. Security vulnerabilities in DNS and DNSSEC. In *2nd International Conference on Availability, Reliability and Security (ARES)*, pages 335–342. IEEE, 2007.

[5] Will Arthur and David Challener. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, 2015.

[6] Jakob E. Bardram, Alex Mihailidis, and Dadong Wan, editors. *Pervasive Computing in Healthcare*. CRC Press, 2007.

[7] Jason Bau and John C. Mitchell. A security evaluation of DNSSEC with NSEC3. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2010.

[8] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 4366, RFC Editor, April 2006.

[9] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[10] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[11] Sven Bugiel, Alexandra Dmitrienko, Kari Kostiainen, Ahmad-Reza Sadeghi, and Marcel Winandy. TruWalletM: Secure web authentication on mobile platforms. In *Int. Conf. on Trusted Systems*, pages 219–236. Springer, 2010.

[12] Sergiu Bursuc, Christian Johansen, and Shiwei Xu. Automated Verification of Dynamic Root of Trust Protocols. In 6^{th} *Int. Conf. on Principles of Security and Trust (POST 2017)*, volume 10204 of *LNCS*, pages 95–116. Springer, 2017.

[13] Mauro Conti, Giulio Lovisotto, Ivan Martinovic, and Gene Tsudik. FADEWICH: Fast Deauthentication over the Wireless Channel. In 37^{th} *IEEE International Conference on Distributed Computing (ICDCS 2017)*, pages 1–13. IEEE, 2017.

[14] Fred D. Davis. User acceptance of information technology: System characteristics, user perceptions and behavioral impacts. *International Journal of Man-Machine Studies*, 38(3):475–487, March 1993.

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, 2008.

[16] S. Dispensa E. Rescorla, M. Ray and N. Oskov. Trasnport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746, RFC Editor, February 2010.

[17] Günther Eibl and Dominik Engel. Influence of data granularity on smart meter privacy. *IEEE Transactions on Smart Grid*, 6(2):930–939, 2015.

[18] Md. Sadek Ferdous and Audun Jøsang. Entity Authentication & Trust Validation in PKI using Petname Systems. In *Theory and Practice of Cryptography Solutions for Secure Information Systems*, pages 302–334. IGI Global, 2013.

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068, RFC Editor, January 1997.

[20] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. An Extension to HTTP : Digest Access Authentication. RFC 2069, January 1997.

[21] Michael Freed and Elango Gannesan. Secure sockets layer proxy architecture, December 12 2006. US Patent 7,149,892.

[22] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[23] Shirley Gaw and Edward W Felten. Password management strategies for online accounts. In *2^{nd} Symposium on Usable Privacy and Security*, pages 44–55. ACM, 2006.

[24] Audun Jøsang, Christophe Rosenberger, Laurent Miralabé, Henning Klevjer, Kent A. Varmedal, Jérôme Daveau, Knut Eilif Husa, and Petter Taugbøl. Local user-centric identity management. *Journal of Trust Management*, 2(1):1–28, 2015.

[25] Henning Klevjer, Kent Are Varmedal, and Audun Jøsang. Extended HTTP Digest Access Authentication. In *Policies and Research in Identity Management (IDMAN)*, volume 396 of *IFIP AICT*, pages 83–96. Springer, 2013.

[26] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *33rd Annual Advances in Cryptology (CRYPTO 2013)*, volume 8042 of *LNCS*, pages 429–448. Springer, 2013.

[27] Geraldine Kwang, Roland H. C. Yap, Terence Sim, and Rajiv Ramnath. An usability study of continuous biometrics authentication. In *3rd International Conference on Advances in Biometrics*, pages 828–837. Springer, 2009.

[28] Ben Laurie and Abe Singer. Choose the red pill and the blue pill: a position paper. In *Workshop on New Security Paradigms*, pages 127–133. ACM, 2009.

[29] Neal Leavitt. Internet security under attack: Undermining of digital certificates. *Computer*, 44(12):17–20, 2011.

[30] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928, 1996.

[31] Chris Lesniewski-Laas and M Frans Kaashoek. SSL splitting: Securely serving data from untrusted caches. *Computer Networks*, 48(5):763–779, 2005.

[32] Mohammad Mannan and Paul van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security*, volume 4886 of *LNCS*, pages 88–103. Springer, 2007.

[33] Félix Gómez Mármol, Christoph Sorge, Osman Ugus, and Gregorio Martínez Pérez. Do not snoop my habits: preserving privacy in the smart grid. *IEEE Communications Magazine*, 50(5):166–172, 2012.

[34] Peter Mayer, Hermann Berket, and Melanie Volkamer. Enabling automatic password change in password managers through crowdsourcing. *11th Int. Conf. on Passwords*.

[35] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *2^{nd} Int. Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2013.

[36] B Dawn Medlin, Joseph A Cazier, and Daniel P Foulk. Analyzing the vulnerability of us hospitals to social engineering attacks: how many of your employees would share their password? *International Journal of Information Security and Privacy (IJISP)*, 2(3):71–83, 2008.

[37] Denis Migdal, Christian Johansen, and Audun Jøsang. DEMO: OffPAD – Offline Personal Authenticating Device with Applications in Hospitals and e-Banking. In *23^{rd} ACM Conference on Computer and Communication Security*, pages 1847–1849. ACM, 2016.

[38] Denis Migdal, Christian Johansen, and Audun Jøsang. Usable authentication with an offline trusted device proxy architecture (long version). Technical Report 453, University of Oslo (IFI), August 2016.

[39] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murphi. In *IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society, 1997.

[40] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465, RFC Editor, February 2015.

[41] Seref Sagiroglu and Gurol Canbek. Keyloggers. *IEEE Technology and Society Magazine*, 28(3):10–17, 2009.

[42] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *IEEE Computer Security Foundations Symposium*, pages 78–94. IEEE, 2012.

[43] Kristoffer Severinsen, Christian Johansen, and Sergiu Bursuc. Securing the End-points of the Signal Protocol using Intel SGX based Containers. In *5^{th} Workshop on Hot Issues in Security Principles and Trust*, pages 40–47. Stuttgart Univ. Technical Report, 2017. Available at https://sec.uni-stuttgart.de/_media/events/hotspot2017/proceedings.pdf#page=40.

[44] Douglas Stebila and Nick Sullivan. An analysis of TLS Handshake proxying. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1 of *TRUSTCOM '15*, pages 279–286. IEEE Computer Society, 2015.

[45] Issa Traore and Ahmed Awad E. Ahmed. *Continuous Authentication Using Biometrics: Data, Models, and Metrics*. IGI Global, 1st edition, 2011.

[46] Kent Are Varmedal, Henning Klevjer, Joakim Hovlandsvåg, Audun Jøsang, Johann Vincent, and Laurent Miralabé. The OffPAD: Requirements and usage. In *Network and System Security*, volume 7873 of *LNCS*, pages 80–93. Springer, 2013.

[47] Bas Verplanken and Wendy Wood. Interventions to break and create consumer habits. *Journal of Public Policy & Marketing*, 25(1):90–103, 2006.