pTASC: Trustable Autonomous Secure Communications

Patrícia R. Sousa
DCC-FCUP/CRACS-INESC TEC
Porto, Portugal
psousa@dcc.fc.up.pt

André Cirne
DCC-FCUP/CRACS-INESC TEC
Porto, Portugal
andre.cirne@fc.up.pt

João S. Resende
DCC-FCUP/CRACS-INESC TEC
Porto, Portugal
jresende@dcc.fc.up.pt

Rolando Martins
DCC-FCUP/CRACS-INESC TEC
Porto, Portugal
rmartins@dcc.fc.up.pt

Luís Antunes
DCC-FCUP/CRACS-INESC TEC
Porto, Portugal
lfa@dcc.fc.up.pt

ABSTRACT

The number of devices connected to the Internet has been increasing exponentially. There is a substantial amount of data being exchanged among numerous connected devices. The added convenience brought by these devices spans across multiple facets of everyday life, such as drivers reporting an accident through dash cams, patients monitoring their own health, and companies controlling the safety of their facilities. However, it is critical to increase safety and privacy across the data generated and propagated by these devices. Previous works have focused mainly on device management and relied on centralized solutions namely Public Key Infrastructure (PKI). This paper describes a novel mechanism that ensures secure autonomous communication between Internet of Things (IoT) devices, while using a completely decentralized solution that mitigates the classical single points-of-failure problem. This is accomplished by a new peer-to-peer protocol using Short Authentication Strings (SAS), in which verification is made through a Limited-Location Channel (LLC).

CCS CONCEPTS

• **Security and privacy → Network security**;

KEYWORDS

IoT, Location-Limited Channel, Network security, Embedded Systems, Security and Privacy

1 INTRODUCTION

As new scenarios and threads emerge, along with the evolution of cyberspace and the number of Internet-connected devices, security also has to evolve. Due to the explosion of the number of these interconnected devices, there are increasing requirements for security and privacy.

IoT is a technological concept where all the devices of our daily lives are connected to the Internet, such as clocks, appliances or even clothes, acting in an intelligent and sensorial manner [16]. When the devices are linked together and to the Internet, they can make use of additional resources towards achieving newer capacities.

These devices are limited in computational resources and, often, are connected to a network that does not have access to the Internet, or only have access to an intermittent ad-hoc mesh network. These restrictions create additional problems when scaling up the number of devices, such as the insufficient bandwidth to maintain these devices interconnected. In any circumstance, the necessity of securely sharing information [25] is one of the open problems in IoT and is the main focus of this work.

The exchange of data/information securely is not novel. There are some solutions based on PKI [2], however, they are not suitable for IoT. A PKI is a system that manages certificates used in asymmetric encryption. Using this infrastructure, any device should be able to verify the integrity and ownership of a public key. A problem normally associated with PKIs is that they rely on a centralized entity that can represent a single point of failure (SPOF).

Even with fault-tolerance, the infrastructure supporting the PKI can potentially fail. Additionally, a PKI can also become inaccessible due to the lack of network connection between a device and itself.

To avoid this, we can consider a decentralized solution [22]. Besides that, the existence of an infrastructure dedicated to a protocol is somehow incompatible with scenarios where the devices are connected to an ad-hoc network, as they can come and go over time, i.e., churn. So, to include all these scenarios, a suitable system needs be supported by the devices themselves, i.e, by implementing a peer to peer protocol.

ICDCN '19, January 4–7, 2019, Navi mumbai, India

In order to solve these restrictions, we have develop a new protocol, based on the Z and Real-time Transport Protocol (ZRTP) protocol [37], which is a key agreement protocol used on the setup of a secure call on VoIP. Thus, we also took into account the properties of Password Authenticated Key Exchange by Juggling (J-PAKE) [19]. We based our proposal on these protocols allowing the negotiation of secret keys peer-to-peer, without requiring the participation of any other device or a human interaction after the first communication/iteration, in order to provide secure peer-to-peer data exchange.

1.1 Contributions

This work makes the following contributions:

- Design and implementation of a novel approach for secure peer-to-peer data exchange focused on a home environment;
- Completely decentralized solution, avoiding the SPOF issue;
- Enhancement of cryptographic protocols: Our approach is based on some properties of J-PAKE and ZRTP to secure the end-to-end data exchange;
- Use of Limited-Location Channel (LLC) to exchange the SAS, and then, compare in both local clients automatically. The use of this concept removes the need for verbal and manual comparison with screens or keyboards. This concept is based on proximity of the ad-hoc network devices, that is, a secure pre-authentication channel can be established by visual or physical contact between the communication devices [5]. This secure pre-authentication channel enables devices to exchange their keys directly without the need for public key and CA certificates. The use of LLC is only needed for the first iteration, because the following iterations rely on key continuity and forward secrecy;
- Definition of a threat model with security analysis;
- Provisioning evaluation with runtime of the pTASC system as compared to a PKI system, where the Online Certificate status Protocol (OCSP) verification overhead exists. We present the setup of the pTASC system and the conditions of the experiments. In this paper, we do not focus on comparisons at the data exchange level (i.e., cipher times).

1.2 Premises

Due to the usage of LLC, this model is based on two assumptions: all participants are located in the same ad-hoc network; and all the participants trust each other *a priori*. Therefore, this type of model fits all scenarios that fall under these two assumptions (in a home environment, for example) and does not apply in any other scenario.

1.3 Outline

Section 2 presents the state of the art with some of systems related with our protocol. In Section 3, we present an overview of the proposed system (pTASC) and characteristics of the system.

This is followed by Section 4 that describes the implementation of pTASC. Section 5 describes a security analysis including a definition of the threat model with several attack scenarios, analyzing whether or not the attacks are mitigated using our approach. Also, we present some theorems and proofs of some of the attack scenarios.

In Section 6, we present results of the comparison of execution times between this system and a PKI system. Lastly, Section 7 presents the conclusions of this work.

2 STATE OF THE ART

This section describes technologies or systems related to our proposed solution.

2.1 ZRTP protocol

The ZRTP is a key agreement protocol used on VoIP. This protocol is not based on digital certificates, but on the Diffie-Hellman (DH) keys (also called shared secret keys). The DH method is a specific encryption algorithm for key exchange based on discrete logarithms. On the end of this key agreement, it is possible to generate, through shared secrets a master key, to create a Secure Real-time Transport Protocol (SRTP) Cryptographic Context and so, establish a SRTP stream [6][37]. The DH algorithm alone, does not provide protection against Man-in-the-Middle (MitM) attacks. In order to authenticate both peers in the key exchange on ZRTP, it is used a SAS, that is generated during the key negotiation from the DH shared secrets. In the case of a MitM attack, the devices will end up with different shared secrets and thus, different SAS. If the SAS is the same, the communication could be classified as secure, but, for this to happen, the SAS needs to be accepted on the two devices by a user. When a communication is classified as secure, the shared secrets are saved, to produce new secure sessions on the future, decreasing the computational effort and skipping the user intervention on the comparison of SAS [37].

The *ZRTPCPP* [13] is an implementation of ZRTP on C++ constructed as a library that adds ZRTP support to the *GNU ccRTP* stack. This library is able to run without other required software, such as cryptographic libraries or local databases. Documentation is clear and has a good support from the community, but the mode of operation is more complicated, because it is part of the *GNU ccRTP*.

The *GNU ccRTP* [14] is an implementation of RTP. This implementation has been created as an application-level protocol framework, giving liberty to the programmer for customize a wide range of settings and aspects of the framework behaviour.

The *ZRTPCPP* is restrictively a implementation of ZRTP and delegates all things to the *GNU ccRTP*, which involves communication, discovering peers and encryption. Currently, this ZRTP implementation is being used by a VoIP application (Silent Phone) [31] and an open source video conference program (Jitsi) [21]. At a higher level of abstraction, the *ZRTPCPP* [13] implementation is based on, the *ZrtpQueue* class, which is responsible for connecting the different components, including the *GNU ccRTP*.

The *ZrtpQueue* is responsible by the management of the *ZRTP sessions*, connecting the ZRTP with the user interface by implementing the *ZrtpUserCallback* and manage the *GNU ZRTP CORE*.

The *GNU ZRTP CORE* processes and implements the ZRTP. Its construction respects a Facade design pattern [8], where the Facade class is the *Zrtp* class. This initialize all parts of the subsystem, including the produce of the hash chain, the mechanism that enforces an attacker to choice a SAS string and that avoid a birthday attack [37][29]. The *Zrtp* class delegates the process of messages to a state machine, the *ZrtpStateClass*, which is also responsible to track the state of the protocol.

2.2 J-PAKE

The PAKE (Password-Authenticated Key Agreement) protocols are a recent addition to the cryptography literature. Currently, the most sophisticated algorithms do not perform key exchanges based on public key cryptography and allow low-entropy passwords to be used. In [23], the three main state-of-the-art PAKE protocols are discussed. In the same paper, there are also two protocols that are more efficient than the J-PAKE protocol, which were adopted as the defaults in OpenSSL. J-PAKE [19] [18] is based on a shared password, which does not require a PKI or a third party in order to establish a secure communication between two parties. It uses an elliptic curve DH for the key agreement and a Schnorr Non-Interactive Zero-Knowledge (NIZK) signatures [17] proof mechanism to authenticate two peers and establish a shared secret between them based on a passphrase. There are some services that still use J-PAKE, such as the Pale Moon Web-Browser, the lightweight API in Bouncycastle (1.48 and onwards), and the Thread (IoT wireless network protocol). This protocol was also supported by FireFox Sync, OpenSSH, and OpenSSL, but was removed after 2014. There are several known J-PAKE issues, already published by *Mohsen Toorani* [35]. This paper presents an analysis of J-PAKE, as used by Firefox Sync, and has identified vulnerabilities in it. J-PAKE is vulnerable to a password compromise impersonation attack and has other shortcomings with respect to replay and Unknown Key-Share (UKS) attacks. Also, according to the same paper, J-PAKE has also been included in OpenSSL and OpenSSH, but problems were reported during implementation [24].

2.3 Device Pairing Using Short Authentication Strings

Device Pairing Using Short Authentication Strings [11] is a two-device pairing mechanism based on the agreement and checking of a secret's authenticity using an SAS. This protocol consists of three phases: discovery, agreement, and authentication. When the pairing service starts, the server starts publishing the chosen instance name. The client will discover that name and the corresponding connection parameters [11]. After the server is discovered, the client and server use a TLS session which allows them to agree on a shared secret using a cryptographic protocol that produces an SAS. After this, there is an authentication phase, used to validate the pairing through an SAS. In this phase, the comparison of the SAS is made through a manual verification, i.e., the user has to verify that both devices display the same string. If, instead, the server and client support Quick Response (QR) codes, then the server displays a QR code with the encoding of the SAS, and the client is capable of scanning the value of the SAS and comparing it to the locally computed value.

2.4 Ad-hoc Authentication

There are some works in the literature that describe authentication models for ad hoc networks. *Nidal Aboudagga et al.* [1] has presented a document with a taxonomy and research issues in the authentication protocols for ad-hoc networks. We describe in more detail some similar to our approach. *Asokan, N and Ginzboorg, Philip* [4] presented a key agreement protocol in ad-hoc networks which is based on PAKE and a location-based key agreement to authenticate through a limited channel such as Bluetooth. However, such protocol does not focused on IoT and does not have key continuity feature that is a good application in this context, as key continuity allows devices to move away after the first pairing and continuously have secure communications on the following connections.

Additionally, there a vast literature related with LLC as [5] that presents new schemes for peer-to-peer authentication in ad-hoc wireless networks. Also, the authors describe

how to use demonstrative identification to perform pre-authentication over LLCs. *Amir Spahić* [33] presents an authentication mechanism (pre-authentication phase) which uses context information through LLC using Infrared. *Serge Vaudenay* [36] presents a concept that authenticates a short string, the SAS, through an extra insecure channel. This concept is similar to our proposal, however, this is based on the use of a narrow-band authentication channel. Another proposal [10] is based on a new LLC using biometrics. The protocol efficiently calculates a shared secret key from biometric data using quantization and cryptanalysis. The authors use grip pattern based biometrics as a location limited channel to achieve pre-authentication in a protocol that sets up a secure channel between two handheld devices.

3 PTASC OVERVIEW

pTASC is based on a DH key exchange to overcomes the complexity of PKI systems or any trusted third party, and also, to make a decentralized solution. However, DH alone cannot ensure authentication, and it has a MitM problem [30]. *Zimmermann P. et al.* [37] proposed a solution to this, by allowing the detection of MitM attacks through displaying a SAS for the users to read and verbally compare over the phone in Voice Over Internet Protocol (VoIP) communications. This solution (ZRTP) lacks the possibility of adapting this protocol for data. In a protocol adapted to data, it cannot be used the comparison through voice because there is no audio or secure channel established where we can exchange the SAS securely.

This protocol needs authentication to prevent a MitM attack during the exchange of SAS. To solve this issue, we use an adaptation of the LLC concept. The idea of this concept is to create an extra channel (secure and authenticated channel) to exchange the SAS securely, without being vulnerable to a MitM attack. Then, the SAS is compared locally on the both client sides. For the exchange, we use Infrared as a LLC. As Infrared is more limited in terms of range, we can ensure devices wanting to exchange information have to be in the close proximity of each other. This is a secure paradigm that is valid in two scenarios: all participants are located in the same room; and all the participants trust each other *a priori*.

This decision to use LLC is also due to the fact that devices in the IoT often do not have a screen to show a pairing information, or at least users cannot interact with this screen easily, so it may be necessary to have a keyboard or another interface. For this reason, pTASC automates all the necessary operations that otherwise would require manual intervention.

Figure 1 presents the communication scheme between two devices (A and B) through pTASC. The communications

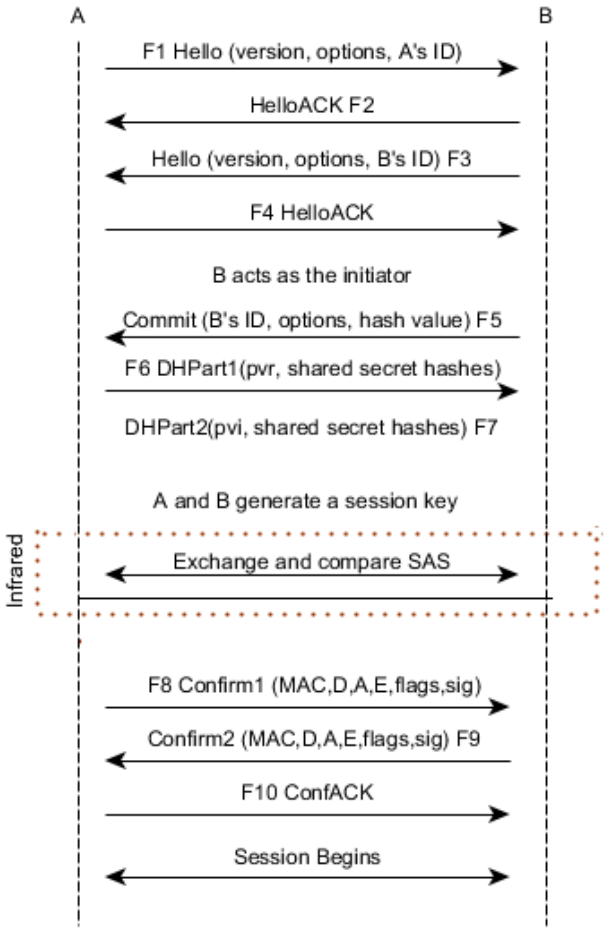

Figure 1: pTASC overview

scheme starts with both of the devices (A and B) exchanging HELLO and HELLOack messages (acknowledgement of receipt of the HELLO message) in steps F1 to F4. In these HELLO and HELLOack messages, the identification of both of the two devices A and B are revealed to each other. The identification is generated using a Pseudo Random Number Generator (PRNG) and this identification is validated in this phase. After this first exchange of messages, the key agreement exchange can begin with a Commit message in step F5 from the device B to the device A. The scheme shown assumes that the device B is the initiator. There are two approaches that can now be carried out in order to agree a key between the device B and the device A:

- DH mode: pTASC endpoints exchange a new shared secret through the DH exchange;
- Pre-shared mode: In this mode, the DH calculation is omitted by the endpoints, as it is assumed that there exists a known shared secret from a previous session.

However, *DHPart1* and *DHPart2* messages [1] are still exchanged to determine which shared keys should be used. Instead of DH values (hvi and pvr), the end points use nonces, along with the retained secret keys, to derive the key material [34].

After this phase, the Infrared component is used to privately exchange the SAS and the comparison is made locally on both sides. Then, it is possible to send a *Confirm* message from both devices indicating that both have accepted and verified the SAS. Finally, the protocol is ready to start sending data in this secure session.

pTASC takes advantage of some additional security and privacy properties, as the forward secrecy, in order to guarantee confidential communication in the future. This means that when the two devices A and B perform multiple connections at different times with each other, the protocol rotates the keys between each ones of the sessions so that the same key is not used in a following session. This way, all the communications keys are different. At this point, an identifier file caches symmetric key material used to compute secret session keys, and these values change with each session. If an attacker gets access to the local seed to derive the future and current keys, the attacker will not be capable of reproducing any of the previous exchanges of information. In other words, if a single communication is compromised (say the session key is "leaked"), this does not compromise the confidentiality of all of the communications prior to it. This way, pTASC is able to guarantee additional protection in communication because, even if the users do not bother with SAS, there is still fairly decent authentication against a MitM attack, based on a form of key continuity.

The negotiation of the keys generates a sequence of letters and numbers, whose size can be defined and which must be equal at both ends. Any attempt to capture and decrypt the voice will cause the sequences to be different (a sequence is a mathematical function derived from the initial key of both ends, so, any difference in the key will generate different values). The key continuity is often confused with forward secrecy. However, the idea of key continuity is that a key is used only once, but is used as seed for the following seed. That is, the old key is used to generate the new one, i.e., the complexity of the key becomes greater with each use. This key continuity solution is a good option for IoT devices because the number of devices are increasing and automatic provisioning of the session keys is necessary, so there is no need for provisioning individually the session key to each ones of the devices A and B, beforehand.

Based on this overview, we highlight two main characteristics of our system:

[1] Messages where the DH public values are exchanged

Usability
Some secure proposals do exist, but, in order to apply them, many procedures are necessary, and many of them require previous key changes. Note that, the exchange of keys by unsecured means (email for example) compromises all of the security of the process. The system proposed in this work is easy to use. In the case of J-PAKE, for example, its use in Firefox Sync is less easy to use, as it requires that the SAS be written in both devices to provision these two devices. Also, in ZRTP, the protocol needs to exchange the SAS verbally and then, compare it on the phone. With LLC, we can exchange the SAS securely and privately between the peers and then, compare it locally in both sides.

Decentralized
The existing limitations of the IoT (low power and processing) are well known, making PKI difficult to use in such an environment. The protocol presented in this paper differs from the PKI model because it is decentralized - it does not rely solely and exclusively on a CA, but rather encompasses a number of reliable, independent elements. Our approach is more suitable for low-resource devices without any need for PKI, key certification, trust models, certification authorities, and so on, which bring with them inherent complexity.

4 IMPLEMENTATION OF PTASC

We decided to construct the new protocol based on the *ZRTP* for end-to-end communication because of the support given by the development community and the existence of a complete ZRTP implementations with well known stable libraries as dependencies. For this reason, we choose the library *ZRTPCPP*.

The *ZRTPCPP* implementation is deeply connected to *GNU ccRTP* that is the software responsible for accept connections, differentiate messages from different protocols and pass them to the respective handler. The RTP protocol uses UDP protocol, so, at the application level, there are mechanisms to ensure the reliability and filter out of order packets. To separate these two packages, we create a package called *DTP*, that emulates the *GNU ccRTP* to facilitate the connection with the *ZrtpQueue*.

In comparison with *GNU ccRTP*, the *DTP* differs in the transport protocol, because it uses TCP instead of UDP. We choose TCP because the protocol that we are developing is directed to data, so, if a packet is lost, it needs to be retransmitted. As we use TCP, we do not need to implement reliability mechanism at the application level (as it happens on the RTP) because TCP already provides reliability mechanisms implemented [27].

The *DTP* package needs to be able to receive and manage many connections at the same time. Without this capability,

the protocol would be limited to communicate with only one device at a time, that would bring a negative impact on performance when a device needs to talk with several peers. This type of necessity is common to servers, where there are two main architecture types of servers: multiplex [20] servers and multithread [7] servers. The multiplex server [20] is an event-driven approach, where asynchronous I/O is used and a unique process is responsible for multiple connections. The server changes between the connections when exists an event to process. The multithread [7] approach basically associates each incoming connection with a separate thread, where synchronous blocking I/O is the natural way of dealing with I/O. Given these two options, we opted by the multiplex server, because of the low memory capacity of an IoT device, which is not compatible with a multithread server. As we needed to implement a multiplex server, it was necessary to find a way to implement the non-blocking I/O. We use the native function *select* to implement the necessary asynchronous mechanisms, allowing a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation.

The *DTP* package is organized in four classes: *DtpServer*, *DtpHandler*, *DtpConnection* and *Worker* (Figure 2).

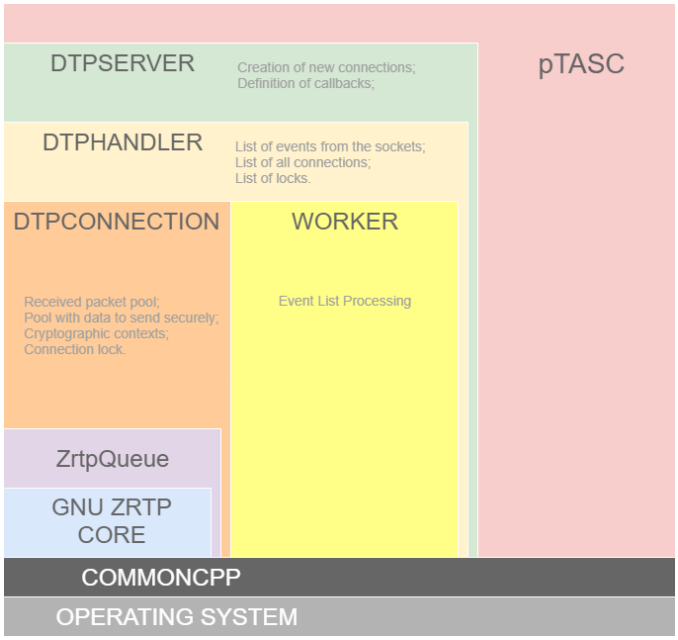

Figure 2: Relationship diagram of the new protocol implementation.

The *DtpServer* has the role of Facade class, allowing the programmer to define user callbacks, handlers for packets and connections with other peers. The *DtpHandler* manages the connections and workers. This class contains the list of all connections and locks to manage the multiplex architecture. Furthermore, this class has the *get_next_socket* function, that monitors the different connections using the *select*. The *Worker* class is the base for the multiplex server, it has his own thread and will process which of the events identified by the *DtpHandler*.

Each one of the connections has an instance of the *DtpConnection* associated. This instance keeps the context of the connection. This context has the different pools of packages, from the cryptographic context to the communication channels. This type of structure is equivalent to the *GNU ccRTP*. The *DtpConnection* is the class responsible for the connection with the *ZrtpQueue* and each *DtpConnection* has one instance of a *ZrtpQueue*.

To structure the packages of the *DTP*, we decided to use as starting point, a ZRTP packet. The ZRTP packet has field constant on all ZRTP packets, called *ZRTP magic number*, which simply serves to identify the protocol. We use this approach to structure the *DTP* packets. We kept the same header structure of a *ZRTP packet*, and changed *ZRTP magic number* value to differentiate protocols. With this type of structure, we can add more protocols sharing the same connection channel.

After the implementation of the *DTP* package and after trying to join the *GNU ZRTP CORE* with the *DTP*, we realized that, on the *GNU ZRTP CORE*, there were dependencies related to *GNU ccRTP*, on functions involved on threads and thread concurrent synchronization. These dependencies problems were possible to solve using *Commoncpp* [15], a small library which is a dependent of *GNU ccRTP*.

Beyond the *DTP* implementation, we need to create a system that uses the SRTP keys generated on the ZRTP to transmit information safely on the same channel. In similarity to the *GNU ccRTP*, we associate a crypto context to each communication channel and cipher the packets on the channel using AES-CBC. The use of this algorithm was due to the existence of an implementation on the *GNU ZRTP* CORE and so, do not increase redundant code.

Our protocol can be expanded by a programmer and integrated on an application. We allowed the developer to override the functions that process and receive data packages and all user callbacks that are hooked to the ZRTP protocol.

With all of the above components, we were able to build a new protocol to the IoT world that offers a reliable form to share information securely, on a peer-to-peer way, without the participation of any other device, besides the sender and the receiver.

5 SECURITY ANALYSIS

In this section we define a Threat Model and we define some claims and the respective theoretical proofs for the claims.

5.1 Threat Model

It is possible for a collision of the SAS to occur, enabling an MitM, i.e., an honest user can have a 4 characters size SAS key, and the attacker can get the same key and thus intercept the call through an MitM [32]. However, this can be prevented by increasing the SAS size. This is performed, as shown in the RFC of ZRTP [38], by increasing the cipher key size (the AES key), producing a new SAS with a greater size. The increase in size adds security regarding the communication cipher and also makes the collision of the SAS harder for an attacker.

Replay attack is an attack that obtains information from one communication and tries to set the information in a next round of the communication. For example, if A is exchanging a file with B, an attacker called Mallory can send a piece of the previous file exchanged to try to corrupt the protocol. To solve this problem, pTASC uses the properties key continuity and forward secrecy to generate a new session key and make obsolete packets and ciphered information exchanged previously. This makes the protocol secure against replay attacks because, if an attacker performs this type of attack, A and B will just ignore the information sent and continue the ongoing transfer.

Traceability problems, were first proposed by *João S. Resende et al.* [28], where the attacker leaks information from VoIP metadata during the communication with the pears. This problem is relevant in VoIP scenarios because there is no need to use this type of metadata when we trust in a third party server. In our scenario, we need to index the information in the local cache, because we do not have information for other sources (peer-to-peer solution). This way, an attacker can obtain metadata information but, for the best of our knowledge, there is no solution to mitigate this metadata leakage, while maintaining a scenario without a trusted third party.

A LLC mitigates the MitM by the attackers, however, it is possible for them to try to intercept the communication and exchange a key with one of the devices. As we use Infrared, it limits the range of distance of the attacker to exchange data with the "trustable" devices. This scenario assumes that all participants are located in the same room; and all the participants trust each other *a priori*.

5.2 Attack Scenarios

In this subsection, we measure the security of the proposed solution against different scenarios of attack. This way, we are able to classify the security of the protocol based on a set of theorems, adapting those used by *Afifi et al.* [3] to prove that an authentication protocol is secure. To complete the evaluation, we also add new theorems.

Claim 1. Our protocol is secure against de-synchronization attacks.

Proof. The key to avoiding de-synchronization attacks is represented in the figure 1, in which there is a unique identifier that is used to store information related to this client in a database. When the user sends the message F7, there is an update to the local cache, rotating the keys. This means that just when both parties have exchanged correctly the information, they update the information. If, for example, the disk corrupts the information or another type of problem occurs, both devices, when performing the pre-shared mode, will not be capable of negotiating a session key and will drop to the first stage, where a new DH key exchange is performed.

□

Claim 2. The proposed protocol is secure against tag impersonation attacks, based on the security provided by the combination of PRNG's and locally stored secret keys.

Proof. Each device has one unique identifier (as stated in the figure 1 in F1 and F3) that is used for communication with any other device. If an attacker obtains device A's ID, he may attempt to send A's ID to a device B, impersonating device A, but he will be unable to do it. As explained previously, we have an identifier file that caches symmetric key material used to compute secret session keys, and these values change with each session. So, when the attacker tries to impersonate device A, it is impossible for him to pass undetected to device B because, when he attempts to generate a new key with the victim, the secret keys are not the same as his, and device B drops the communication. In the first iteration, he will have an authenticated channel, so he cannot complete the attack, regardless.

□

Claim 3. Our protocol is secure against replay attacks.

Proof. Replay attack is an attack that obtains information from one communication and tries to set the information in a next round of the communication. For example, if A is exchanging a file with B, Mallory can send a piece of the previous file exchanged to try to corrupt the protocol. To solve this problem, pTASC uses the properties key continuity and forward secrecy to generate a new session key and make obsolete packets and ciphered information exchanged previously. This makes the protocol secure against replay attacks because, if an attacker performs this type of attack, A and B will just ignore the information sent and continue the ongoing transfer.

□

Claim 4. Resistant to Single Point of Failure.

Proof. In the case of PKI implementations, there is a third party, a CA, that checks the validity of the certificates. The CA establishes a link between public keys and identities people or organizations. Therefore, customers have to rely on a third party.

The certification authority represents a SPOF, as once one is compromised, all peers are compromised as well. For example, *Let's Encrypt* has issued 15,270 "PayPal" certificates [9] to sites used for phishing. A failure in this type of systems compromises several entities.

Regarding pTASC, while facing an attack it can only compromise at most one of the participants, not both, because it is a decentralized solution.

□

6 EVALUATION

In this section we will show the performance tests that we made of our protocol with a test case. For all measurements, 10 samples were always taken so that we can obtain a mean and a standard deviation of the values obtained in several connection attempts. The results were only at the level of provisioning, because here we are not interested in the data exchange cipher time, only the time until the moment of the SAS exchange.

This section also presents a comparison between the performance of our scenario versus a centralized solution such as PKI systems.

6.1 Setup

To test the operation of the protocol in the real world, we create an IoT environment as shown in Figure 3, where two Raspberry Pi 3 Model B, running *Raspbian Stretch* [12], are connected over an ad hoc network. Infrared sensors were used, namely an IR Transmitter (KY-005) and an IR Receiver (KY-022), to exchange the SAS key.

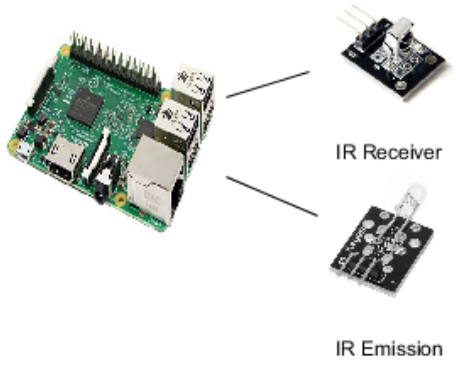

Figure 3: Ad hoc Network setup

6.2 Results

With the previous setup we made a test to evaluate the secure session establishment runtime.

The Infrared was implemented with *pigpio* [2]. As we have an IR Receiver and an IR Transmitter, we make two exchanges from the Alice to Bob and simultaneously, from Bob to Alice. We collected ten time samples of one of the communications, with an average of time to exchange of $0.19ms \pm 0.04$.

Finally, to establish the secure session, we measure the runtime since the start of the protocol until the creation of a secure channel, using the function *gettimeofday* from the native library *sys/time.h*. We collected ten time samples, with an average of time to connect of $1050ms \pm 58.5$.

In all, we have a mean runtime of $1050.2ms$ on the first connection.

Figure 4 shows the PKI scenario to be evaluated. Here, we have a client, represented by a local Raspberry Pi, and a server hosted in a remote Raspberry Pi provisioned with DigiCert certificates. The remote certificates support the OCSP stapling that removes the complexity of customers communicating with the CA. The TLS Server periodically questions the OCSP responder about the validity of its own certificate and caches the response. The OCSP responder returns an OCSP response, which is (directly or indirectly) signed by the CA that issued the certificates. The TLS client can treat this stapled OCSP response in the same way, i.e., the certificate should only be used if it has a valid timestamp and signature.

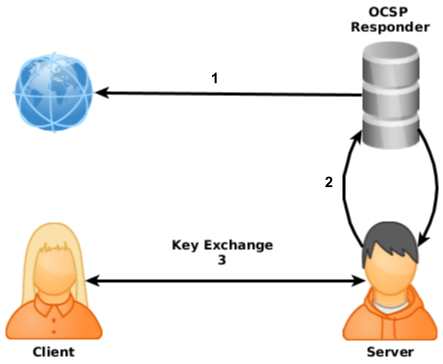

Figure 4: Scenario 2 - Provisioning with PKI

During the TLS handshake, the client announces support for OCSP stapling to the server. In turn, the server activates

[2] *pigpio* [26] is a library for the Raspberry which allows control of the General Purpose Input Outputs (GPIO) library and the time was measured with a SAS exchange

the Certificate Status flag if it has supports for it. This process is shown in step 1 in Figure 4. During step 2, an SSL connection is established between Alice and Bob. The goal is to measure the certificate's OCSP (state) check runtime on top of the runtime added by SSL connection handshake.

To measure the runtime of this scenario, we used *s_client* and *s_server* binaries from the OpenSSL library. We ran *s_server* service on the server machine, and made the *s_client* run on the local machine, so that we could obtain both the OCSP information and server/client handshake results. With this in place, we obtained an average runtime of $380ms \pm 11.6$ across the 10 iterations.

6.3 Discussion

The runtime of our system increases 64% comparatively to PKI systems. The use of an infrared channel behaving as LLC does not add much overhead to the overall runtime. This means that the time of $1050ms \pm 58.5$ is the approximate base time, and other methods for LLC may be added. That is, if the Infrared channel is changed by another method like NFC, for example, we have a time of $1050ms \pm 58.5 + \delta$, the δ being the execution time of the NFC.

This inefficient scenario is related with the change for TCP connections and the packet loss from the ad hoc network. Also, the use of PKI was tested based in high density certificate network. This means that the runtime is important and everything is optimized to perform in the smallest time possible, compared to our implementations that is deployed in a local scenario and does not uses optimized solutions.

7 CONCLUSIONS AND FUTURE WORK

Compared to the traditional environments of PKI systems (Citizen Card or HTTP servers), the IoT systems incur in limitations from battery and sometimes Internet connections. Our first approach was to analyze a home environment where we could secure the systems based on the home trust. Based on this, we could empower the IoT devices without the need of expensive computations to communicate with the outside world, but at the same time infer security by removing the connection with the Internet. This ad hoc network, without access to third party services improves security by mitigating the attack surface.

We have presented a novel solution for the provisioning and communication between IoT devices that makes use of a modified secure key exchange protocol, in which the audio channel was replaced with data negotiation over a LLC.

We provide an evaluation of the provisioning phase on both pTASC and PKI-based systems. The experiments were performed in a realistic environment so that we could present meaningful results that are close to real-world usage. When comparing pTASC with PKI-based systems, our system exhibits a runtime overhead of 64%. However, we have a decentralized solution, which guarantees some key properties, such as security and independence and solving the SPOF problem.

The system uses a LLC for SAS comparison. Although we have used Infrared, we can adapt other mechanisms like Bluetooth, NFC or other technologies of limited location. A combination of these methods is also possible.

In future work, we plan to deploy this architecture in a real environment to fully understand the impact on the IoT devices in a smart-city concept.

ACKNOWLEDGMENTS

The work of Patrícia R. Sousa and Luís Antunes was supported by Project "NanoSTIMA: Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring and Analytics/NORTE-01-0145-FEDER-000016", financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

João S. Resende is supported by the research grant PD/BD/128149/2016, provided by the Portuguese national funding agency for science, research and technology, Fundação para a Ciência e Tecnologia (FCT), within the scope of Operational Program Human Capital (POCH), supported by the European Social Fund and by national funds from MCTES.

The work of Rolando Martins was supported by a scholarship from the Fundação para a Ciência e Tecnologia (FCT), Portugal (scholarship number SFRH/BPD/115408/2016)

REFERENCES

[1] Nidal Aboudagga, Mohamed Tamer Refaei, Mohamed Eltoweissy, Luiz A. DaSilva, and Jean-Jacques Quisquater. 2005. Authentication Protocols for Ad Hoc Networks: Taxonomy and Research Issues. In *Proceedings of the 1st ACM International Workshop on Quality of Service &Amp; Security in Wireless and Mobile Networks (Q2SWinet '05)*. ACM, New York, NY, USA, 96–104. https://doi.org/10.1145/1089761.1089777

[2] Carlisle Adams and Steve Lloyd. 2002. *Understanding PKI: Concepts, Standards, and Deployment Considerations* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] Mohamed Hossam Afifi, Liang Zhou, Shantanu Chakrabartty, and Jian Ren. 2018. Dynamic Authentication Protocol Using Self-Powered Timers for Passive Internet of Things. *IEEE Internet of Things Journal* 5 (2018), 2927–2935.

[4] N Asokan and Philip Ginzboorg. 2000. Key agreement in ad hoc networks. *Computer communications* 23, 17 (2000), 1627–1637.

[5] Dirk Balfanz, Diana K. Smetters, Paul Stewart, and H. Chi Wong. 2002. Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In *NDSS*. Network and Distributed System Security Symposium, San Diego; CA; USA., 13.

[6] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. 2004. *The Secure Real-time Transport Protocol (SRTP).* RFC 3711. RFC Editor. http://www.rfc-editor.org/rfc/rfc3711.txt http://www.rfc-editor.org/rfc/rfc3711.txt [Online; Accessed 16-06-2018].

[7] Ernest S Bender and Chun-Kwan K Yee. 2007. Method and apparatus for managing thread execution in a multithread application. (May 8 2007). US Patent 7,216,346.

[8] Jan Bosch. 1998. Design Patterns as Language Constructs. *Journal of Object-Oriented Programming* 11 (1998), 18–32.

[9] Bill Brenner. 2017. Let's Encrypt issues certs to 'PayPal' phishing sites: how to protect yourself (2017)). (2017). http://bit.ly/2i7Z4bT http://bit.ly/2i7Z4bT, [Online; Accessed 19-05-2017].

[10] I.R. Buhan, J.M. Doumen, Pieter H. Hartel, and Raymond N.J. Veldhuis. 2006. *Feeling is Believing: a location limited channel based on grip pattern biometrics and cryptanalysis.* Number 06-29 in CTIT technical reports series. Centrum voor Telematica en Informatie Technologie, Centrum voor Telematica en Informatie Technologie. Imported from CTIT.

[11] Christian Huitema Daniel Kaiser. 2016. Device Pairing Using Short Authentication Strings. (2016). https://github.com/jitsi/zrtp4j https://tools.ietf.org/html/draft-ietf-dnssd-pairing-01, [Online; Accessed 21-04-2017].

[12] Raspbian Developers. 2018. Raspbian. (2018). https://www.raspbian.org/ https://www.raspbian.org/ [Online; Accessed 16-06-2018].

[13] Werner Dittmann. 2018. ZRTPCPP. (2018). https://github.com/wernerd/ZRTPCPP https://github.com/wernerd/ZRTPCPP [Online; Accessed 16-06-2018].

[14] Inc Free Software Foundation. 2018. GNU ccRTP. (2018). https://www.gnu.org/software/ccrtp/ https://www.gnu.org/software/ccrtp/ [Online; Accessed 16-06-2018].

[15] Inc Free Software Foundation. 2018. GNU [u]Common C++. (2018). https://www.gnu.org/software/commoncpp/ https://www.gnu.org/software/commoncpp/ [Online; Accessed 16-06-2018].

[16] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[17] F Hao. 2016. Schnorr NIZK Proof: Non-interactive Zero Knowledge Proof for Discrete Logarithm version 5. *Online]. Tersedia: https://tools.ietf. org/html/draft-hao-schnorr-05* (2016).

[18] Feng Hao. 2017. J-PAKE: Password-Authenticated Key Exchange by Juggling. (2017).

[19] Feng Hao and Peter YA Ryan. 2008. Password authenticated key exchange by juggling. In *International Workshop on Security Protocols*. Springer, 159–171.

[20] Kazuhiko Isoyama. 2007. Multiplex server system and server multiplexing method. (Sept. 20 2007). US Patent App. 11/723,499.

[21] jitsi. 2018. zrtp4j. (2018). https://github.com/jitsi/zrtp4j https://github.com/jitsi/zrtp4j [Online; Accessed 16-06-2018].

[22] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. 2001. Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks.. In *icnp*, Vol. 1. Citeseer, 251–260.

[23] Jean Lancrenon, Marjan Škrobot, and Qiang Tang. 2016. Two more efficient variants of the j-pake protocol. In *International Conference on Applied Cryptography and Network Security*. Springer, 58–76.

[24] S. Martini. 2018. Session Key Retrieval in J-PAKE Implementations of OpenSSL and OpenSSH. (2010). (2018). https://github.com/jitsi/zrtp4j http://seb.dbzteam.org/crypto/jpake-session-key-retrieval.pdf, [Online; Accessed 16-06-2018].

[25] Kim Thuat Nguyen, Maryline Laurent, and Nouha Oualha. 2015. Survey on secure communication protocols for the Internet of Things. *Ad Hoc Networks* 32 (2015), 17–31.

[26] pigpio. 2018. The pigpio library (2012). http://abyz.me.uk/rpi/pigpio/index.html. (2018). [Online; Accessed 31-07-2018].

[27] Jon Postel. 1981. *Transmission Control Protocol.* STD 7. RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt http://www.rfc-editor.org/rfc/rfc793.txt.

[28] João S Resende, Patrícia R Sousa, and Luís Antunes. 2018. Evaluating the Privacy Properties of Secure VoIP Metadata. In *International Conference on Trust and Privacy in Digital Business*. Springer, 57–68.

[29] Dominik Schürmann, Fabian Kabus, Gregor Hildermeier, and Lars Wolf. 2016. CVE-2016-6271. Available from MITRE, CVE-ID CVE-2016-6271.. (Feb. 2016). https://nvd.nist.gov/vuln/detail/CVE-2016-6271 https://nvd.nist.gov/vuln/detail/CVE-2016-6271 [Online; Accessed 16-06-2018].

[30] Dong Hwi Seo and P Sweeney. 1999. Simple authenticated key agreement algorithm. *Electronics Letters* 35, 13 (1999), 1073–1074.

[31] SilentCircle. 2018. ZRTPCPP. (2018). https://github.com/SilentCircle/ZRTPCPP https://github.com/SilentCircle/ZRTPCPP [Online; Accessed 16-06-2018].

[32] Dorgham Sisalem, John Floroiu, Jiri Kuthan, Ulrich Abend, and Henning Schulzrinne. 2009. *SIP security.* John Wiley & Sons.

[33] Amir Spahić, Michael Kreutzer, Martin Kähmer, and Sumith Chandratilleke. 2005. Pre-authentication using infrared. In *Privacy, Security and Trust within the Context of Pervasive Computing*. Springer, 105–112.

[34] Peter Thermos and Ari Takanen. 2007. *Securing VoIP Networks: Threats, Vulnerabilities, and Countermeasures.* Addison-Wesley Professional.

[35] Mohsen Toorani. 2014. Security analysis of J-PAKE. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*. IEEE, Funchal, Portugal, 1–6.

[36] Serge Vaudenay. 2005. Secure Communications over Insecure Channels Based on Short Authenticated Strings. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology (CRYPTO'05)*. Springer-Verlag, Berlin, Heidelberg, 309–326. https://doi.org/10.1007/11535218_19

[37] P. Zimmermann, A. Johnston, and J. Callas. 2011. *ZRTP: Media Path Key Agreement for Unicast Secure RTP.* RFC 6189. RFC Editor. http://www.rfc-editor.org/rfc/rfc6189.txt http://www.rfc-editor.org/rfc/rfc6189.txt [Online; Accessed 16-06-2018].

[38] Phil Zimmermann, Alan Johnston, and Jon Callas. 2011. ZRTP: Media path key agreement for unicast secure RTP. (2011).