Research Article

C-Lock: Local Network Resilient Port Knocking System Based on TOTP

Javier Junquera-Sánchez , **Carlos Cilleruelo** , **Luis de-Marcos** ,
and **José-Javier Martínez-Herráiz**

Department of Computer Science, Universidad de Alcalá, Spain

Correspondence should be addressed to Carlos Cilleruelo; carlos.cilleruelo@uah.es

Received 3 June 2021; Revised 20 October 2021; Accepted 26 November 2021; Published 31 January 2022

Academic Editor: Dapeng Wu

Copyright © 2022 Javier Junquera-Sánchez et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Port knocking is an access-control technique that consists of revealing a network protected resource only to those users that can prove they know a preshared port sequence. This proving process is done by connecting to the defined ports in the correct order; so, the list gets exposed to the adversaries with access to the connection's channel. We propose a newfangled technique for protecting this process, avoiding eavesdroppers to get a long-live valid sequence. Our method is based on TOTP codes and has been designed thinking on making it the most usable as possible. There has been designed two different approaches, but we demonstrate that the most simple of them is far enough robust, while it remains to be very usable. This technique is especially suitable for enhancing the resilience of network services against local network adversaries.

1. Introduction

In a worldwide connected network, where almost all servers have to be remotely administrated, access control has become a crucial task in data protection. As Telnet was in the past, nowadays, the most popular protocol for remote administration is SSH (Secure Shell) [1]. The hardening of SSH service has been widely addressed by the security experts, concluding that network segmentation (such as VPN or VLAN) and packet filtering in earlier network stages could be the most effective approaches for it. However, there are scenarios where this kind of measures are not feasible or are completely ineffective.

One of the most utilized security measures for hardening SSH services is port knocking. Port knocking is an inexpensive protection technique that consists of denying the connection to a targeted port until a preshared set of ports has not been “knocked” (i.e. contacted somehow) in a certain order (i.e., until these ports have not received a connection from the client, the protected port will not be opened).

Some approaches could seem similar, like changing the default port for other (e.g., change the SSH's port from 22

to 2222), but they are just “security through obscurity” measures that could be useful to avoid some automatic attacks, but ineffective against a simple network scan [2]. By contrast, port knocking is generally accepted to be effective both as a second authentication factor and filtering technique [3].

Even so, against certain adversaries, port knocking lacks effectiveness, and it becomes necessary to find alternatives and improvements. These adversaries are mainly

- (1) Adversaries with enough time for finding the correct knock sequence
- (2) Adversaries with eavesdropping capabilities

This is the case of some IoT (Internet of things) devices that may not have enough capabilities for creating an isolated network (just for themselves). But if they can filter incoming connections, then it can detect a port knocking sequence. In particular, it can be applied to medical environments where it is necessary to install network-connected medical devices, whose configuration cannot be modified (or cannot even be known) at the risk of loss of guarantee, and whose security is not reliable. As Ben-Gurion researches

showed [4], there is an increasing myriad of unpatched medical devices connected to medical networks which could be easily used for pivoting through network segments.

IoT devices can be mainly categorized into two types [5]: devices with high resources (e.g., Raspberry PI-based) or with low resources (e.g., based on Arduino or other micro-controllers). Often, as IoT devices that are implemented in low resource hardware, developers have to optimize their power consumption, and they are deployed without implementing any security control [6]. This leverages to a high-risk scenario for information security, where the most common threats are signal jamming attacks and replay attacks [6]. Signal jamming consists of that the adversary interferes with communication between two systems. This interference can be done via several technologies, like RFID (radio-frequency identification). But as IoT is widely implemented also using common mobile technologies (e.g., Android, and Raspberry PI), both attacks (i.e., signal jamming and replay attacks) can be performed using well-known techniques.

These well-known techniques are often inherited from traditional computer networks attacks. Some of them are mitigated simply by correlating data about the genuine devices to detect and evade intruders [7], because some IoT devices require low-cost specifications. But as the complexity of the system grows (even though it means the attack surface also grows), advance defense techniques can be implemented.

The present work is aimed at providing a mitigation to the risks that port knocking faces against local network adversaries (whose origin is mainly placed on replay attacks) and provides the design of a soundness port knocking model which also brings resilience against local network-related attacks. Even though the findings of this paper have been detected having in mind medical environments, our proposal is valid for many other scenarios (especially, any scenario where a local adversary could be present).

We have also put our efforts in designing the model following industry standards, based on the most usable and secure techniques, to increase the likelihood of its successful implementation in real environments.

The remainder of this document follows the next structure: In Section II, we study the different elements that compound our scheme and its context. In Section III, we analyze the different adversaries that a port knocking system could face. In Section IV and Section V, we describe our system and analyze its properties. Finally, in Section VI, we extract the conclusions of our analysis.

2. Background

2.1. Port Knocking. The most basic port knocking system, proposed in [8], consists in establishing a secret set of ports that must be interacted with before a protected port gets revealed. Figure 1 shows the process for establishing an SSH connection, with the ports 1234, 5678, and 9012 defined as the secret set.

One of the most popular implementations (in fact, the official Debian's port knocking package [9]) was developed

FIGURE 1: Port knocking process.

by Judd Vinet [10], but there are many references about how to implement this security technique. One very well documented could be read in [11].

The objective of port knocking is not just putting a facade to service for avoiding configure it correctly (which is highly discouraged). Even a correctly secured service could be the victim of a zero-day attack [12], and ensuring that only a certain number of clients can interact with it, and could be the only effective action against it.

It is necessary to understand that port knocking is not a system per se, but an authentication procedure. And consequently has to be analyzed as an authentication procedure, caring about all the issues these systems could face [13].

By contrast, the single package authentication approach [14] relies on a mechanism where the client, to be allowed to connect, has to prove his by identity sending a special package to the server.

2.2. TOTP. Another core element of our proposal is TOTP. TOTP is a HOTP-based algorithm (HMAC-based one-time password) [15] for generating time-based OTP codes. Using TOTP, two sides of communication can mutually authenticate themselves with a high-security degree, just using simple and ephemeral numeric codes.

HOTP codes are generated, broadly, applying an HMAC function [16] to a counter code. This counter (also known as moving factor) acts as a kind of ratchet [17] and usually means the time a security event has happened for ensuring mutual authentication (e.g., the times a remote controller has been used, for opening a car). In TOTP, the code is defined as the 30 second slots “elapsed since midnight UTC of January 1, 1970” [18].

TOTP is a de facto standard for providing two-factor authentication and is widely implemented as an online account access control approach (like Google, Facebook, and many others do). To prove his identity, after entering his password, a user has to provide a numeric code generated by an application or a smart-card where they have previously installed a special code.

This process is simply a TOTP code generation, based on a shared secret (usually codified in the QR code) and is used for authenticating that the user who provides the login password is the same person that was registered at the sign-up stage.

It can also be applied in the prevention of replay attacks [19], and even though it should be sent over secure channels (i.e., encrypted like IPsec or SSL [20], the key must be strong enough for resisting unprotected transmissions without putting the generation system on risk.

2.3. Related Work. There have been several researches focused on improving port knocking systems. In [21], the authors analyze what they call NAT-knocking problem, which happens when both the legit client and the attacker are behind the same NAT. In this case, the attacker could impersonate the client because, after the knocking process, the server just identifies the genuine client by his IP address. In addition, they refer to problems related to the overload that port knocking could cause to the server (DOS-Knocking attack), but this issue had been previously addressed using 2FA [22].

Moreover, sending multiple factors for hardening the port knocking authentication process has been widely studied by [23]. They also detect the difficulty of associating the identity of the client who authenticates (who does the knocking process) with who connects (the one who later establishes the connection with the protected port).

Finally, the time synchronization between client and server has been proposed as a way of randomizing the knocking sequence. In [24], they use the time as part of the seed for an RNG, which determines the knock sequence every time it receives a connection.

2.4. Adversary Modeling. The port knocking's authentication process is more similar to a monologue than to a conversation. This fact is very relevant both for defining our adversary model and for the later analysis.

From now on, we will have to present those four actors:

- (i) Protected service: the service that must not be exposed to anyone who does not know the correct knock sequence
- (ii) Server: the host where the protected service is located. It will also be the oracle for checking the port combination
- (iii) Client: the genuine user who knows the shared secret necessary for solving the port knocking sequence and accessing the protected service
- (iv) Adversaries: those actors (semihonest or malicious) who should never access the protected service

Unless otherwise indicated, when we talk about authentication, we are referring to the port knocking process (i.e., not the secret service or other authentication processes).

2.5. Assumptions. The server will always be online and accessible at least to every user in a local network.

- (i) The cryptographic elements implemented (i.e., cypher suites and TOTP) are robust
- (ii) The firewall system is secure

- (iii) There exists a preshared secret between the client and the server, has been generated with a secure PRNG, and has been securely shared
- (iv) The protected service has its security measures correctly implemented (i.e., secure authentication and attack detection)

Concerning our model, it must comply with the following specifications:

- (i) After a correct knocking sequence, the protected service will only be available for a short period for new connections, and only from the same source
- (ii) Firewall will allow established connections (i.e., if after the port knocking, a correct authentication against the protected service is done, and this connection must not be denied after the time)

2.6. Adversary Goals. As the adversary never should know how to solve the protected service's security challenges (e.g., the SSH credentials), his goal should be to be able to access the protected service anywhere he wants (e.g., for brute forcing).

This can be done well by acquiring the capability to generate the knocking sequence anytime (e.g., retrieving the secret key) or forcing the client to repeat it arbitrarily.

2.7. Capabilities. In the worst case, our adversary will be a Dolev-Yao's adversary [25], which, in practice, means that the only limit to his capabilities is those related to the cryptographic techniques used.

- (i) Knows the algorithm
- (ii) Runs in polynomial time
- (iii) Can interact anytime with the server and with the network (i.e., has eavesdropping and tampering capabilities)
- (iv) Does not know the preshared secret

Depending on his attitude, we could distinguish three possible adversaries:

- (1) Honest-but-curious: an eavesdropper who can retrieve the knock sequence played by the client
- (2) Malicious online (active in authentication time): can tamper the connection during the knock sequence or even impersonate the client's IP
- (3) Malicious online (active after port knocking time): can force the client to replay the knock sequence (e.g., interrupting an established connection, so the client has to connect again)

Malicious offline adversaries are not taken into account because, as argued in security, they would not suppose a threat.

2.8. Proposed Model. Without ever ignoring security, usability is a fundamental pillar of this model, as its effectiveness and its acceptance depend on it [26]. These objectives will be achieved following security guidelines and using industry state of art technologies. Additionally, user-side functionality has been designed; so, the user will not need anything else than well-known tools (i.e., would not need to install new software to use the solution).

2.9. Setup. To guarantee the correct construction of the TOTP codes, both client and server have to be time synchronized. The best way is using an NTP server, but this functionality has to be provided by third parties (i.e., a public time server) to fulfill our compatibility principle (e.g., the user has to be able to configure its TOTP generator using Google Authenticator) [27].

After the server administrator configures which service should be protected (as in a regular port knocking system), the shared secret is established in the server. This secret must be generated using a secure PRNG [28] and must be at least 160 bits long. According to TOTP's RFC requirements: "The keys MAY be stored in a tamper-resistant device and SHOULD be protected against unauthorized access and usage" [18].

On the client's side, the shared secret will not necessarily be stored in the device that will do the authentication (the client itself). In fact, in most cases, it will be into the device of an operator (e.g., its mobile phone), who will just introduce the TOTP code in the client when the authentication is required.

The sharing method must be human-friendly and easy to read by an external device. The proposed procedure for achieving this purpose is using both a base 32 representation and a QR code. This method (illustrated in Figure 2) is the most common way of setting up 2FA in web services.

After the previous step, the system is ready. The protected service is hidden behind the firewall, and the server starts to wait for the first port knocking sequence.

2.10. Port Sequence Generation. The first time slot's lifetime will lasts the time gap between the epoch timestamp t_0 [29] and the next timestamp t such that $t \equiv 0 \pmod{30}$. After that moment, a new port set is generated each 30 seconds.

Figure 1 shows the algorithm designed for the port generation. The initialization process is very simple, just consists in generating the TOTP code. The substantial part of the algorithm construction has been how to generate the port set with this code. And here is where we have found the most serious conflict between usability and security.

The main benefit of this strategy is that the number of ports for the knocking could be increased (to increase the knocking effort) without increasing the generation cost (as it stills depending on the same seed), but requires special software in the client's side for processing the TOTP.

On the other hand, the second strategy just requires the TOTP code for generating the knocking sequence. Each port would be mapped to a pair of digits from the TOTP code and added to a mask. The proposed mask has not been arbitrarily chosen, but $0 \times C000$ (49152) marks the start of the

FIGURE 2: Sharing secret.

dynamic ports range [30], and $0 \times C030$ is the first number that ends with two zeros. These ports have been selected due to the following:

- (1) No service should be listening on them (this way, we avoid disruptions)
- (2) In consequence, it is very difficult to receive a SYN request (i.e., something that could be interpreted as a knock)
- (3) Using $0 \times C030$ instead of $0 \times C000$ allows the user to replace the two trailing zeros with the TOTP's corresponding value, instead of having to calculate the sum

With this alternative strategy, no additional software would be needed in the client. The knock sequence could be easily derived by a human from the TOTP code, and it could even be reproduced using a web browser.

2.11. Authentication Process. Once calculated the knock sequence, the client has to reproduce it. The server will interpret every SYN message as a knock; so, the client has to follow the algorithm, connecting in the correct order to each sequence's port.

We have selected SYN to characterize a knock for compatibility because almost any software with network capabilities will be able to generate it (as commented previously, in the worst case, it could be reproduced with a web browser).

A complete TCP handshake has been discarded as it would require an active service for the interaction (what could suppose overload). Also, it is easier to allow (as far as secure firewall configuration is concerned) the port knocking service to listen to SYN messages, than allowing complete input connections (especially in those port ranges).

After a first correct knock from an IP address, the server saves an internal state for this source (as shown in Figure 3). If the next knock is also correct (following the sequence), the state is changed, otherwise, is deleted. When a user has correctly reproduced the complete sequence, the server adds a rule in the firewall for letting him connect to the protected service for 30 seconds.

FIGURE 3: Sequence evaluation.

2.12. Example Scenario. Due to the versatility and functionality of c-lock, this system can be applied to multiple network architectures and scenarios. One example scenario will be to apply c-lock in a health environment, like a hospital.

As a critical system, hospitals are in great need of cybersecurity [31]. Also, hospitals need to bring security to medical devices, and they cannot or modify these medical devices. This situation generates a problem, how to offer security to devices that you cannot modify or operate [32]?

One possible solution to this problem is to involve firewalls in order to access and communicate with these devices. In the designing and implementation process of a network architecture, we can involve the use of firewalls in front of all medical devices. There are numerous firewalls, open source, and not that are Linux-based, and c-lock could be easily deployed in them.

Furthermore, we will be involving 2FA without affecting their daily use or default installation. However, it is implicit that good network segmentation is crucial for the security of this system. Firewalls running c-lock should be the only way of communication with these devices.

3. Discussion

The proposed method was designed for systems with packet filter capacities (i.e., in contrast with those IoT systems with low complexity-cost specifications, like integrated circuits). As the main purpose of the method is to protect access to administration interfaces (e.g., SSH), it should not exceed any computational constraint.

Implementing this method is also costly free in computational terms for this type of system, i.e., should not imply any overhead in contrast with a classical port-knocking process, as it should also be executed once per SSH session, and it is based on trivial operations. However, it would be necessary to modify the operational procedures to add the port knocking step before starting the administration sessions.

3.1. Security. The model has been analyzed under the principles discussed in [33]. We have chosen an unbounded scenario, where an infinite number of sessions could be played by the adversary.

As previously told (see section Adversary modeling), the analysis of a port knocking system has some peculiarities. In particular, using a symbolic approach in a model where there is just one player sending messages may not show relevant results [34]. Given that, and even though we have picked elements from other verification schemes, the

strength of the system will be measured using a cryptographic approach.

Assuming that TOTP generation is secure, the simplest way for an adversary to determine the key which (would let him generate correct port sequences) will be the exhaustive search [35]. This approach, according to [36], would take more than 10 years for our key size and algorithms. So then, it should not exist a successful offline attacker.

We have evaluated our model as a computational complexity problem where the average effort for breaking it and should take more than 30 seconds (the time slot's length) to be carried out. This analysis has also led us to determine if is secure to implement the most usable port generation strategy (strategy B showed in Algorithm 1).

There are $1e6$ different combinations in both strategies. It is trivial to check this fact in approach B, where the port numbers are directly extracted from the TOTP code (and it has 6 digits). Approach A combinations depend on HOTP. HOTP is a deterministic function which, in the space of the TOTP code, and for more than 2 different moving factors, makes the port generation function injective (i.e., the probability that 2 different TOTP codes generate the same set with three or more ports is negligible) (see Lemma 1)). So then, strategy A generates also $1e6$ different combinations.

Lemma 1. (Strategy A is injective). $HOTP: \mathbb{R} \times \mathbb{Z} \longrightarrow \mathbb{Z}1e6$

$$\begin{aligned}
 F: \mathbb{Z} \times \mathbb{R} \times \mathbb{R} &\longrightarrow \mathbb{B}/F(x, y, z) \Leftrightarrow \mathbf{hotp}(x, y) = \mathbf{hotp}(x, z), \\
 \forall a, b \in \mathbb{Z}_{1e6} | a \neq b | \forall x &\Rightarrow Qx = P(F(x, a, b)) = 1/1e6, \\
 \forall y | y > 2 &\Rightarrow Q_0 Q_1 \cdots Q_y = (1/1e6)^y \sim 0.
 \end{aligned}
 \tag{1}$$

Therefore, the adversary's effort will depend on how each strategy should be executed.

3.2. Adversary Effort. We have to define some variables to calculate the effort that an adversary should do to succeed and how much time would it cost. Each round (i.e., each complete knocking sequence followed by checking the status of the protected port) runs over these parameters:

- (i) The time for computing the HOTP code is negligible because it could even be precomputed
- (ii) Each port sequence has n ports, and each port is codified with two digits of the TOTP code. So, there would exist $1e(2n) = 10^{2n}$ possible codes
- (iii) The time for executing each knock lasts K time (with K semiconstant, defined by the time for sending a SYN message)
- (iv) After the last correct knock, the protected service gets exposed after y time

Consequently, for both strategies, each round will last $n \cdot K + y$ seconds. In strategy A, n and y could be configured in the implementation, while in strategy B, just y can be set. As a result of that, the effort for testing all

```

Initialization
1:  $tSlot \leftarrow epochT \text{ imestamp()} // 30$                                 { //means integer division }
2:  $totp \leftarrow hotp(tSlot, sk)$                                 { e.g. 123456 }
Strategy A
1: for  $n \leftarrow 0$  to 3 do
2: begin
3:  $aux \leftarrow hotp(n, totp)$ 
4:  $port[n] \leftarrow (aux \% 65534) + 1$                                 {  $0 < port < 65535$  }
5: end
Strategy B
1:  $mask \leftarrow 0xC030$                                 {  $0xC030 = 49200$  }
2:  $p_{aux}^{[0]} \leftarrow (totp // 1e4)$                                 { e.g. 12 }
3:  $p_{aux}^{[1]} \leftarrow (totp // 1e2) - (p_{aux}^{[0]} * 1e2)$                                 { e.g. 34 }
4:  $p_{aux}^{[2]} \leftarrow (totp) - (p_{aux}^{[1]} * 1e2) - (p_{aux}^{[0]} * 1e4)$                                 { e.g. 56 }
5: for  $n \leftarrow 0$  to 3 do
6: begin
7:  $port[n] \leftarrow mask + p_{aux}[n]$                                 {  $port[0] = 0xC030 + 12 = 49212$  }
8: end

```

ALGORITHM 1: Port generation algorithm.

combinations would be $10^{2n} \cdot (K + y)$, and the average effort for finding a successful random port set will be $10e(2n) \cdot (K + y)$ seconds.

With an equal fixed port set length (e.g., 3 ports in the two strategies), the requirement for an adversary to not be able to break the system (i.e., not generating a correct sequence in less than 30 seconds) is that $y > (30^2 - 10^6 \cdot K) / 10^6$.

In the practice, this means that in a highly unlikely scenario where a SYN message is sent instantly, y should be greater than $9e(-3)$. Moreover, when the adversary would find a correct sequence (i.e., after almost spent 16 hours, setting y to one second), he would only be able to attack the protected service during less than 30 seconds.

And all that without regarding the noise it would make.

3.3. Other Attacks. As it is a passive system (like the previous works), it is not vulnerable to port scans (it just listens and modifies its internal states), depending on his attitude.

The only profit an honest-but-curious adversary could take is eavesdropping the knock sequence and makes a replay attack. Repeating the sequence (as long as it is done in the same timeslot) would let him access to the protected service, but only during 30 seconds. The legit client just sends the knock sequence once, and its connection to the protected service stays for a while; so, this adversary will have few chances of success.

From a malicious adversary perspective, there could be two different types of attacks related to achieving his goals. The malicious adversary could be active during the authentication process or after it.

During the authentication process, an adversary could tamper the communication through a man-in-the-middle attack, so that he could use the client as a code generator. It would work as follows:

- (1) The adversary kills the established connection, to force the client to authenticate again

- (2) The client sends the port sequence, and the adversary uses it to authenticate himself
- (3) When the client tries to reach the protected service, the adversary answers with a fake rejection
- (4) Then the client, believing there could have been an error, would send again the sequence or wait until the next timeslot to repeat the process

If the TOTP code was generated by the client (instead of introduced by an operator), and the authentication was done by an automatical process, the adversary could be able to repeat this process indefinitely. With an operator-controlled authentication, this attack would be easier to detect, because after some failures the operator would notice that something is happening.

The other active attack (after the authentication process) would be similar to the passive approach. Instead of replaying the sequence, after successful authentication, the adversary could impersonate the client's IP address for accessing the protected service. This approach would only let him access to the protected service for 30 seconds unless he launches other attacks (oriented to exploit the protected service, not the port knocking authentication).

In none of these attacks, the adversary would be able to forge correct port sequences.

3.4. Contribution. In contrast with the classic port knocking model, which could be defeated after just one client's authentication (i.e., an eavesdropper just needs to listen one connection to retrieve the knocking sequence), our model is not vulnerable to eavesdropping. If an adversary obtains a correct set of ports, it will only be able to use it in the remaining time until the next timeslot (i.e. less than 30 seconds).

In our scenario, the NAT knocking attack [21] would not be feasible, because all participants would be in the same network segment. The server just calculates the knocking

TABLE 1: Time complexity vs. adversary effort.

n	Sequence calculation (s)	Avg. adv. effort (s)	Avg. adv. effort [$y = 1$ s] (s)
1	$3.0000e-04$	0.22	10.00
2	$2.0503e-05$	2.23	100.01
3	$1.8358e-05$	22.36	1000.12
4	$2.0980e-05$	223.60	10001.24
5	$2.4557e-05$	2236.06	100012.49
6	$2.7894e-05$	22360.67	1000124.99
7	$3.1232e-05$	223606.79	10001249.92
8	$3.4809e-05$	2236067.97	100012499.21
9	$3.8385e-05$	22360679.77	1000124992.18

sequence once at the start of the timeslot; so, DOS-knocking attack [21] would also be useless.

Our approach avoids the overload that [24] could generate in the system. The complexity of our model, with a fixed knock sequence length, is $O(1)$ for computing the knock sequence, and if it is computed every time the time-slot changes (i.e., instead of when the client tries to connect), it would not cause any significant overhead. Table 1 shows a server-side time analysis for calculating n ports and the average effort (in seconds) it would suppose for an adversary guess in the sequence with just a latency of 0.25 ms (i.e., the minimum latency required for critical IoT applications [37]), and with the y value set in one second (as discussed in the Adversary Effort subsection).

Besides, our model (in special, configured with the strategy B) is more versatile when being used by different types of clients and systems. Thirdly, generating the port sequence using TOTP, instead of a custom method, is more secure and verifiable ([38]).

4. Conclusions

The model proposed provides the necessary elements to introduce port knocking systems in local networks in a secure manner. With our system, an adversary will not be able to take profit from an eavesdropped authentication process. Moreover, he would not be able to craft a correct port knocking sequence.

Therefore, we have achieved our goals: the model is secure, efficient, and very usable. Even though the port generation could be more complex (as occurs with port generation strategy A), we have proven that it will not increase significantly the security. Additionally, it would involve unnecessary complexity in the client, while with a simple method (i.e., port generation strategy B), we can assemble a long-lived security system.

The system should be constructed using strategy B. Furthermore, if strengthen the system's resilience was required (something that because of the results might not happen), it would be enough with incrementing the time between the last correct knock and protected service opening.

4.1. Future Work. An adversary could take advantage of an incorrect implementation. If it is necessary, an automatically triggered authentication should be limited to the number of retries after an error. If the retries number overcomes a threshold, supervision should be required.

All the elements out of our system scope should be configured following the best practices of security. Finally, the model has been designed to be used for just one client (more specifically, with just one preshared key).

It would be great to verify the system using automatical protocol provers (e.g., Tamarin Prover ([39]; [40])), addressing also the secret key sharing process and the introduction of the TOTP code in the client.

Nevertheless, the most attractive research path could be its extension for a multiclient scenario. As the TOTP standard foresees the generation of up to 10 digits codes, it could be studied its usage. Studying if TOTP-based port switching could be implemented in a SPA approach might also yield interesting results.

Data Availability

Data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 Research and Innovation Program under grant agreement No. 826284 (ProTego).

References

- [1] J. Geerling, *Ansible for DevOps: Server and Configuration Management for Humans* [Google-Books-ID: oLuVjgEACAA], 2015, LeanPub.
- [2] D. Stuttard, "Security & obscurity," *Network Security*, vol. 2005, no. 7, pp. 10–12, 2005.

- [3] S. Jeanquier, *An analysis of port knocking and single packet authorization*, [M.S. thesis], Information Security Group Royal Holloway College, University of London, 2006.
- [4] T. Mahler, N. Nissim, E. Shalom et al., "Know your enemy: Characteristics of cyber-attacks on medical imaging devices," in *Presented at the RSNA Conference*, vol. 6, Chicago, IL, United States, 2017.
- [5] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight cryptography algorithms for resource-constrained IoT devices: a review, comparison and research opportunities," *IEEE Access*, vol. 9, pp. 28177–28193, 2021.
- [6] F. Chantzis, I. Stais, P. Calderon, E. Deirmentzoglou, and B. Woods, *Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things*, No Starch Press, Inc, San Francisco, 2021.
- [7] E. E. Tsiropoulou, J. S. Baras, S. Papavassiliou, and G. Qu, "On the mitigation of interference imposed by intruders in passive RFID networks," in *Decision and Game Theory for Security*, Q. Zhu, T. Alpcan, E. Panaousis, M. Tambe, and W. Casey, Eds., Springer International Publishing, Cham, 2016.
- [8] M. Krzywinski, "Port knocking: network authentication across closed ports," *SysAdmin Magazine*, vol. 12, pp. 12–17, 2003.
- [9] "Debian – el sistema operativo universal," 2020, <https://www.debian.org/index.es.html>.
- [10] J. Vinet, "Knockd - a port-knocking server," 2020, <https://zeroflux.org/projects/knock/>.
- [11] M. Doyle, *Implementing a Port Knocking System in c (Honors Thesis)*, The University of Arkansas, 2011.
- [12] M. Rash, *Linux firewall ls*, No Starch Press, 2007, <https://nostarch.com/firewalls.htm>.
- [13] M. Zviran and Z. Erlich, "Identification and authentication: technology and Implementation issues," *Communications of the Association for Information Systems*, vol. 17, no. 1, 2006.
- [14] M. Rash, "Protecting SSH servers with single packet authorization," *Linux Journal*, vol. 2007, no. 157, p. 6, 2007.
- [15] F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, "HOTP: An HMAC-based one-time password algorithm," (RFC No. 4226) [Library Catalog: <https://tools.ietf.org/>]. 2020, <https://tools.ietf.org/html/rfc4226>.
- [16] H. Krawczyk, R. Canetti, and M. Bellare, "HMAC: keyed-hashing for message authentication (RFC No. 2104)," 2020, <https://tools.ietf.org/html/rfc2104>.
- [17] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *Advances in cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds., Springer International Publishing, Cham, 2017.
- [18] J. Rydell, M. Pei, and S. Machani, "TOTP: time-based one-time password algorithm (RFC no. 6238)," 2020, <https://tools.ietf.org/html/rfc6238>.
- [19] A. S. M. Abukeshipa and T. S. M. Barhoom, *Implementation and Comparison of OTP Techniques (TOTP, HOTP, CROTP) to Prevent Replay Attack in RADIUS Protocol*, [M.S. thesis], Islamic University of Gaza, 2014.
- [20] M. L. T. Uymatiao and W. E. S. Yu, "Time-based OTP authentication via secure tunnel (TOAST): a mobile TOTP scheme using TLS seed exchange and encrypted offline keystore," in *2014 4th IEEE International Conference on Information Science and Technology*, Shenzhen, China, 2014.
- [21] A. I. Manzanares, J. T. Márquez, J. M. Estevez-Tapiador, and J. C. H. Castro, "Attacks on port knocking authentication mechanism," in *Computational science and its applications – ICCSA 2005*, O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganá, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, Eds., Springer, Berlin, Heidelberg, 2005.
- [22] D. Worth, *CÖK - cryptographic one-time knocking*, BlackHat, 2004, Retrieved May 2, 2020, from <https://blackhat.com/presentations/bh-usa-04/bh-us-04-worth-up.pdf>.
- [23] R. de Graaf, J. Aycock, and M. Jacobson, "Improved port knocking with strong authentication," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, Tucson, AZ, USA, 2005.
- [24] T. Popeea, V. Olteanu, L. Gheorghe, and R. Rughiniş, "Extension of a port knocking client-server architecture with NTP synchronization," in *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*, Iasi, Romania, 2011.
- [25] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.
- [26] C. Braz and J.-M. Robert, "Security and Usability: The Case of the User Authentication Methods," in *Proceedings of the 18th conference on l'interaction homme- machine*, Montreal, Canada, 2006.
- [27] "Google Authenticator," 2020, <https://play.google.com/store/apps/details?id=com>.
- [28] J. I. Schiller and S. Crocker, "Randomness requirements for security (RFC no. 4086)," 2020, <https://tools.ietf.org/html/rfc4086>.
- [29] "Epoch (computing) [Page Version ID: 950077310]. (2020, April 10). In Wikipedia. Page Version ID: 950077310," 2020, [https://en.wikipedia.org/w/index.php?title=Epoch_\(computing\)&oldid=950077310](https://en.wikipedia.org/w/index.php?title=Epoch_(computing)&oldid=950077310).
- [30] J. Touch, "Recommendations on using assigned transport port numbers (RFC no. 7605)," 2020, <https://tools.ietf.org/html/rfc7605>.
- [31] L. Coventry and D. Branley, "Cybersecurity in healthcare: a narrative review of trends, threats and ways forward," *Maturitas*, vol. 113, pp. 48–52, 2018.
- [32] P. A. Williams and A. J. Woodward, "Cybersecurity vulnerabilities in medical devices: a complex environment and multifaceted problem," *Medical Devices*, vol. 8, p. 305, 2015.
- [33] S. Matsuo, K. Miyazaki, A. Otsuka, and D. Basin, "How to evaluate the security of real-life cryptographic protocols?," in *Financial Cryptography and Data Security*, R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, Eds., Springer, Berlin, Heidelberg, 2010.
- [34] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in cryptology – EUROCRYPT 2000*, B. Preneel, Ed., Springer, Berlin, Heidelberg, 2000.
- [35] M. J. Wiener, "Exhaustive key search," in *Encyclopedia of Cryptography and Security*, H. C. A. Tilborg, Ed., Springer US, Boston, MA, 2005.
- [36] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," in *Public Key Cryptography*, H. Imai, Y. Zheng, H. Imai, and Y. Zheng, Eds., Springer, Berlin, Heidelberg, 2000.
- [37] P. Schulz, M. Matthe, H. Klessig et al., "Latency critical IoT applications in 5g: perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, 2017.

- [38] W. M. S. Alves, T. L. Prado, A. M. Batista, and F. A. S. Ferrari, "The dangerous path towards your own cryptography method," 2018, <https://arxiv.org/abs/1812.05440>.
- [39] B. Schmidt, *Formal analysis of key exchange protocols and physical protocols [Ph.D. thesis]*, ETH Zurich, 2012.
- [40] "Tamarin prover," 2020, <https://tamarin-prover.github.io>.