FRACTAL: Single-Channel Multi-Factor Transaction Authentication through a Compromised Terminal

Savio Sciancalepore[1], Simone Raponi[2], Daniele Caldarola, and Roberto Di Pietro[2]

[1] Eindhoven University of Technology (TU/e), Eindhoven, Netherlands.
s.sciancalepore@tue.nl
[2] Hamad Bin Khalifa University (HBKU) - College of Science and Engineering (CSE) Division of Information and Computing Technology (ICT), Doha, Qatar
{sraponi, rdipietro}@hbku.edu.qa

Abstract. Multi-Factor Authentication (MFA) schemes currently used for verifying the authenticity of Internet banking transactions rely either on dedicated devices (namely, tokens) or on out-of-band channels—typically, the mobile cellular network. However, when both the dedicated devices and the additional channel are not available and the Primary Authentication Terminal (PAT) is compromised, MFA schemes cannot reliably guarantee transaction authenticity. The afore-mentioned situation is typical, e.g., offshore or on-board of aircraft, when only few untrusted terminals have Internet connection.
In this paper, we present FRACTAL, a new scheme providing single-channel transaction MFA through general-purpose additional authentication terminals. Moreover, the proposed solution is also resilient against a potentially-compromised PAT. FRACTAL easily scales up as per the number of multiple authentication factors, and it is extensible beyond the banking scenario, e.g., to unattended and constrained scenarios, by integrating also Internet of Things (IoT) devices as additional authentication terminals. Other than enjoying a formal verification of its security properties via *ProVerif*, FRACTAL is also supported by an extensive experimental performance assessment. Our real-world Proof-of-Concept scenarios, implemented using *Spring* micro-services, show that FRACTAL can complete a transaction in about 2 seconds, independently from the remote server location. The flexibility of use, the guaranteed security, and the striking performance, characterize FRACTAL as a solution with an expected high potential impact in the authentication field, for both Industry and Academia.

Keywords: Internet Transactions, Network Security, Cryptographic Protocols.

1 Introduction

The capillary diffusion of Internet services in the last decade has certified the shift of banking services from in-person to online [1], and this trend has been

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. For the definitive Version of Record: proceedings of The 24th International Conference on Information and Communications Security, 5-8 Spetember 2022, Canterbury, UK.

even magnified by the COVID19 pandemic. Nowadays, all banks offer web platforms and mobile smartphone applications allowing users with an active bank subscription to manage their funds online [2].

Despite the evident advantages, the afore-mentioned shift carries a plethora of security issues. For instance, powerful remote attackers could steal legitimate users' credentials to fully impersonate them on the web, e.g., by initiating unauthorized monetary transactions, leading to huge economic losses for both individuals and companies [3]. To offer enhanced security to their customers, many bank service providers currently implement user Multi-Factor Authentication (MFA) solutions [4]. Specifically, user MFA schemes require the client to provide multiple pieces of evidence to demonstrate to be the same physical person associated with the end-user account. Typically, user MFA occurs via either the delivery of a one-time Personal Identification Number (PIN) to a device registered by the end-user with the bank, such as a mobile cellular number, or proving to be in possession of dedicated smart card readers, released by the bank to the user [5]. Moreover, recent standards such as FIDO and FIDO2/WebAuthN provided multiple standardized mechanisms and out-of-the-box APIs to perform user authentication on several platforms [6].

In this context, mutual transaction MFA protocols focus on ensuring the authenticity of the Internet transactions. Such schemes have been widely investigated, both in the literature and in the industry/banking domain (see Sec. 6 and the survey in [4]). Nonetheless, currently-deployed solutions strongly rely on the availability of either dedicated devices (such as reader/token generators), used specifically for authentication purposes, or additional channels to the Internet connection, e.g., the cellular network, used to deliver the one-time password. When they are not available, their security relies on the security of the primary terminal adopted by the user to interact with the remote service. If such a terminal is controlled by the adversary, currently-available user MFA schemes cannot guarantee transactions mutual authentication, as they cannot reliably verify remote server's authenticity. The unavailability of the additional channel is a typical situation, e.g., on a cruise ship offshore, or during an aircraft trip, to name a few. In these scenarios, users and remote servers should establish *Strong Mutual Authentication* without a dedicated channel to the secondary devices, allowing an untrusted terminal to route messages to/from general-purpose additional authentication terminals. To the best of our knowledge, such a challenging scenario has not been addressed, yet.

Contribution. In this paper, we present *FRACTAL*, an efficient solution to enforce single-channel transaction MFA even when the main terminal is compromised. *FRACTAL* is a flexible scheme, where several user devices (e.g., general-purpose or Internet of Things (IoT) ones) can be used to demonstrate the authenticity of an online transaction, although the primary authentication terminal used to trigger the transaction could be compromised. Moreover, *FRACTAL* requires very limited effort by the user, which is required only to identify its own transaction. We discuss the security features of *FRACTAL*, and we prove its security via the verification tool *ProVerif*. Moreover, we implemented a func-

tioning prototype of *FRACTAL* proving that, using *FRACTAL*, it is possible to successfully perform an online transaction in 2 seconds on average, while adding multiple devices only slightly affects its performance.

We believe that *FRACTAL* may be useful in several application scenarios—as witnessed by the *FRACTAL* supporting patent [7], that inspired this paper.

Roadmap. This paper is organized as follows. Sec. 2 introduces the scenario and the adversary model, Sec. 3 illustrates *FRACTAL*, Sec. 4 discusses the security of *FRACTAL*, Sec. 5 includes the performance evaluation of *FRACTAL*, Sec. 6 reviews the related work, and, finally, Sec. 7 tightens the conclusions.

2 Scenario and Adversary Model

2.1 Scenario

We assume an end-user, namely $\mathcal{A}$, want to access via a regular Internet connection her savings account at the bank $\mathcal{B}$. $\mathcal{A}$ and $\mathcal{B}$ are equipped with a private/public key pair, and their connection is secured via the well-known Transport Layer Security (TLS) protocol, e.g., leveraging public-key certificates. We assume that $\mathcal{A}$ uses a Primary Authentication Terminal (PAT) to interact with the bank $\mathcal{B}$. The Primary Authentication Terminal (PAT) can be either a fixed workstation or a laptop. We do not assume the presence of any particular additional interface on the PAT (e.g., biometrics).

We assume that the server of $\mathcal{B}$, i.e., the *remote server*, requires MFA to authorize any operation. To this aim, $\mathcal{A}$ has to register with $\mathcal{B}$ multiple Additional Authentication Terminals. The Additional Authentication Terminals (AATs) can be any general-purpose device in possession of $\mathcal{A}$, that she can leverage to demonstrate her identity at the authentication time. Note that $\mathcal{A}$ registers the AATs at the join with $\mathcal{B}$, and they can be added/modified through secure channels. During the registration phase, the bank B stores securely its public key and public key certificate on each AAT, to avoid any possible tampering.

Finally, we assume that the AATs could not be connected to the Internet, thus being not able to communicate directly with $\mathcal{B}$. This is a frequent situation, occurring when the end-user is in a remote location. For instance, when $\mathcal{A}$ is on a cruise ship, usually only the PAT is connected to the Internet, while any other AAT would require additional subscriptions. Another use-case would be the use of a shared on-demand terminal, that is owned by the user but rented on request.

2.2 Adversarial Model

The adversary assumed in this work, namely *ADV*, is in full control of the PAT. We neglect the specific tool used by the attacker to compromise the PAT, as the PAT could be either colluding with the adversary or deployed by *ADV* on purpose. Overall, this assumption empowers *ADV*, enabling him to carry out both passive and active attacks from the PAT. As a passive attacker, *ADV* is a global eavesdropper, able to detect any packet transmitted and received by

the PAT. As an active attacker, *ADV* features active attacking capabilities, in line with the well-known Dolev-Yao attacker model [8]. Thus, *ADV* can inject its own messages on the channel, either by replaying eavesdropped messages or by forging new messages, impersonate either the PAT or the server, as well as perform Man In The Middle (MITM) attacks against every involved party. In addition, by having complete control of the PAT, *ADV* can tamper with the copy of the key and the public key certificate of the server stored on the PAT, e.g., by replacing them with one of his choices. Moreover, in our paper, we assume that at least one AAT is not compromised by the adversary.

One of the possible goals of the attacker is to steal money from $\mathcal{A}$, held in her account on $\mathcal{B}$. To this aim, relying on the compromised PAT, *ADV* can launch either synchronous or asynchronous MITM attacks.

In the former scenario, *ADV* waits for the end-user to perform a transaction. At that time, *ADV* launches a MITM attack and redirects any request performed by the client to a malicious server, interacting with the remote server on behalf of *A*. In the latter case, *ADV* launches the attack without waiting for actions performed by the end-user.

3 Protocol Description

3.1 Basic Protocol Flow

Fig. 1 describes the initial steps required by *FRACTAL*. Note that the operations described in this section are always executed, and they do not depend on the particular scenario where the protocol is operated.

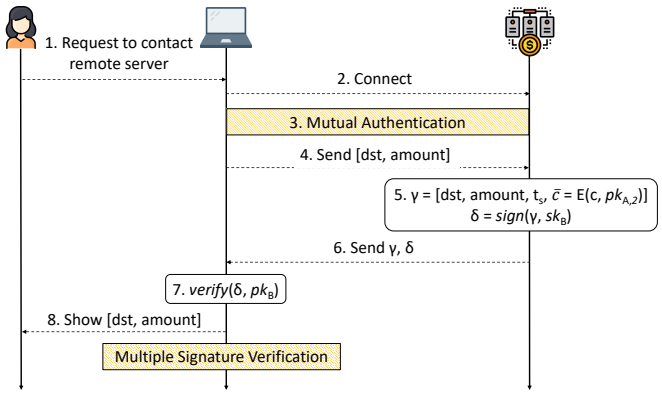

Fig. 1. Initial protocol flow of *FRACTAL*.

1. The end-user first instructs the PAT to initiate a connection with the remote server. Typically, this operation occurs through the typing of a Uniform Resource Locator (URL) in the browser.

2. The PAT sends a connection request to the public IP address of the remote server, on behalf of the end-user. Note that, if the PAT is compromised, the request can be redirected first to the IP address of the attacker, to be then re-routed to the server—e.g., the attacker carries out a MITM.
3. The PAT and the remote server carry out mutual authentication, to verify each other identities. Any mutual authentication protocol can be used here (the prototype described in Sec. 5.1 uses TLS with X.509 certificates).
4. Through the PAT, the end-user specifies the details of the desired banking transaction. This interaction typically happens through the keyboard, and involves the specification of the transaction recipient dst, and the amount to be transferred, $amount$.
5. Assume $\mathcal{A}$ registered a single AAT with the bank (N=1). Let $sk_{\mathcal{A},2}$ and $pk_{\mathcal{A},2}$ be the private/public key pair of the AAT. On the reception of the transaction details from the PAT, the remote server generates a one-time code c. The code c is encrypted using the public key of the registered AAT $pk_{\mathcal{A},2}$, generating an encrypted code $\tilde{c}$, as in Eq. 1.

$$\tilde{c} = E\left(c, pk_{\mathcal{A},2}\right), \tag{1}$$

   where the operator $E\left(m, K\right)$ refers to the public-key encryption of the plaintext m using the public key K. When N AATs are registered, N encrypted codes are generated according to Eq. 1. Then, the remote server creates a public-key signature of the transaction, namely δ, as in Eq. 2.

$$\delta = sign\left(\left[dst, amount, t_s, \tilde{c}\right], sk_{\mathcal{B}}\right), \tag{2}$$

   where t_s refers to the expiration time of the transactions, and $sign$ is a generic public-key signature algorithm.
6. The information about the recipient of the transaction, the amount, the timestamp, and the signature δ are delivered back to the PAT.
7. At reception time, the PAT first verifies the authenticity of the signature δ, as per Eq. 3, by using the public key of the remote server $pk_{\mathcal{B}}$.

$$verify\left(\left[dst, amount, t_s, \tilde{c}\right], pk_{\mathcal{B}}\right) \stackrel{?}{=} \gamma, \tag{3}$$

   where $verify\left(\cdot\right)$ is a signature verification algorithm. Note that the PAT can verify only the authenticity of δ. Instead, it cannot verify autonomously the content of $\tilde{c}$, but it has to rely on the assistance of the AAT(s).
8. As a first verification step, the recipient and the amount involved in the transaction are showed to the end-user. Thus, the end-user can immediately realize if the intended recipient and amount match the ones showed by the PAT. However, when the PAT is compromised, this verification step is not enough to ensure transaction authenticity. Therefore, additional validation steps are performed with the assistance of the AAT(s).

The above-discussed steps are common to all the scenarios assumed in our work. The following operations, instead, depend on the scenario and capabilities of the involved terminals. We hereby consider two reference scenarios, where the AATs are equipped either with a Bluetooth channel (3.2), or with a camera (3.3), being these the most diffused interfaces in general-purposes devices.

3.2 Scenario #1

In this scenario, we assume the PAT is connected to the AATs via Bluetooth. In line with Sec. 2, we assume that the AAT(s) do not have Internet connection. Fig. 2 shows the additional interactions required by *FRACTAL*, described below.

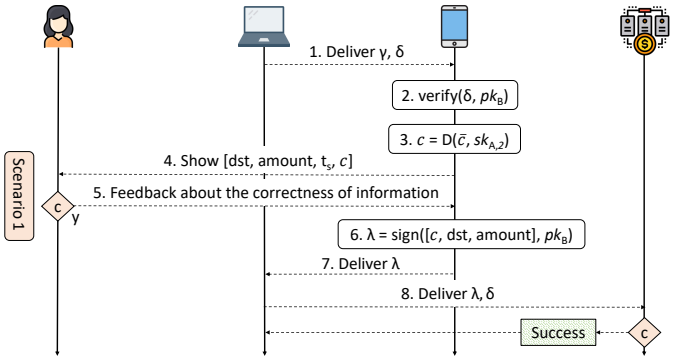

Fig. 2. Additional interactions required by *FRACTAL* in the Scenario #2.

1. Assuming the pairing between the PAT and the AAT has been already performed. The PAT delivers to the AAT all the information received from the remote server, including the transaction recipient, the amount, the validity time, the encrypted code, and the signature δ.
2. First, the AAT verifies the signature δ (2b), via the check in Eq. 3.
3. Then, using its private key $sk_{\mathcal{A},2}$, the AAT extracts c, as per Eq. 4.

$$c = D\left(\tilde{c}, sk_{\mathcal{A},2}\right), \tag{4}$$

4. The available information, i.e., the transaction recipient, amount, and validity time are visually shown to the user, e.g., via a screen, so that she can verify their consistency. If the end-user verifies the correctness of the information and the AAT supports an input method, the end-user can specify her final approval, by pressing a dedicated button. Otherwise, the end-user rejects the transaction.
5. If $\mathcal{A}$ validates the transaction, the AAT signs all the information using the public key of the remote server, generating a signature λ. The value λ is then delivered to the PAT.
6. The PAT forwards to the remote server λ and δ.
7. At reception time, the remote server checks the correctness of λ, using its private key. If the code just received matches the locally-stored one for the particular transaction, identified by its signature δ. If the correspondence is verified, the transaction is executed. Otherwise, the transaction is aborted.

3.3 Scenario #2

In this scenario, we assume the AAT(s) does not feature any means to connect with a remote PAT. Despite this is a quite restrictive assumption (usually, at least one connection mode is available), it helps modelling a scenario where the PAT and the AAT are theoretically incompatible. The only assumption on the AAT is that it includes a camera and an application that can process QR codes. Fig. 3 shows the additional interactions required by *FRACTAL*, described below.

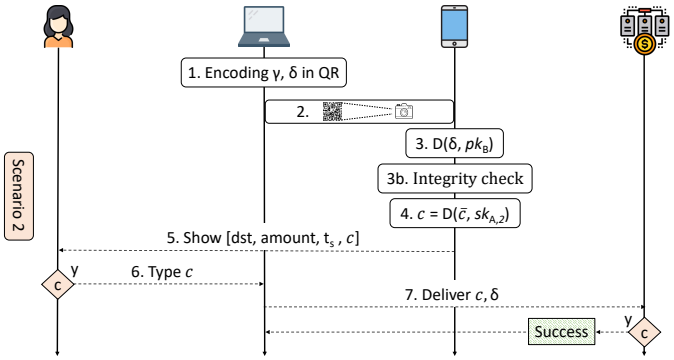

Fig. 3. Additional Interactions required by *FRACTAL* in the Scenario #2.

1. The PAT creates a QR code, encoding the information received from the remote server, i.e., the transaction recipient, the amount, the time validity, $\tilde{c}$, and δ, and it shows the QR code on the screen.
2. $\mathcal{A}$ uses the camera on the AAT to acquire the QR code.
3. The processing of the QR code includes two main steps. First, the AAT validates δ (3b), through Eq. 3. Then, the AAT extracts c, as per Eq. 4 (2b).
4. The available information, i.e., the transaction recipient, amount, and validity time are visually shown to the user, e.g., via a screen, so that she can verify their consistency.
5. If the end-user verifies the correctness of the information, the end-user can proceed with the transaction by entering the one-time code c to the PAT. Otherwise, the end-user rejects the transaction.
6. In case $\mathcal{A}$ typed the code c, the PAT forwards to the remote server c and δ.
7. At reception time, the remote server checks if c matches the locally-stored one for the particular transaction, identified by its signature δ. If it matches, the transaction is executed. Otherwise, the transaction is aborted.

4 Security Considerations

4.1 Security Features

Overall, *FRACTAL* provides the following security features.
Protection Against Replay Attacks. FRACTAL is robust against replay attacks thanks to the support of nonces and expiration timestamps. Indeed, the signature of the transaction (δ) delivered to the PAT and the AATs includes a one-time code, c, and a timestamp, indicating the transaction expiration time. Thus, any replay after the expiration time is immediately identified and rejected. At the same time, if the adversary replays the transaction before the expiration time, being the code c one-time, the transaction is rejected, as well.
Protection Against MITM Attacks. Thanks to standard mutual authentication, *FRACTAL* protects against external MITM attacks, launched by adversaries not in control of the legitimate entities. An additional feature of *FRACTAL* is the capability to reject MITM attacks even when the PAT is compromised. Indeed, through *FRACTAL*, the end-user can always detect the mismatch between the provided information and the (supposedly authenticated) one received by the remote server. We prove such property in the following Theorem 1.

Theorem 1. *Assume that the attacker ADV compromises the PAT. Then, under the security assumptions in Sec. 2.2, ADV is prevented from misleading the user and the remote server about the authenticity of a forged transaction.*

Proof. Assume that *ADV* selects dst' and $amount'$ as the forged transaction recipient and amount. Controlling the PAT, *ADV* can obtain from the remote server an authentic signature $\delta' = sign\left[dst'||amount'||t_s||\tilde{c'}, sk_{\mathcal{B}}\right]$, where $\tilde{c'} = E\left[c', pk_{A,2}\right]$. Now, the attacker has three options. The first one is to send to the AAT the legitimate values dst and $amount$, together with the forged signature δ'. In this case, the signature verification on the AAT, reported in Eq. 3, fails. Thus, the attack would be rejected.
The second option consists in delivering to the AAT dst', $amount'$, and δ'. In this case, being δ' generated from dst' and $amount'$, the check in Eq. 3 would be successful. However, the end-user can notice that $dst \neq dst'$ and $amount \neq amount'$, thus detecting the attack and rejecting the transaction.
The third option is to report to the AAT the legitimate transaction values, i.e., $amount$, dst, and δ, but to report to the remote server the code for the legitimate transaction and the signature of the forged transaction (i.e., c, δ'). Being δ generated from both dst and $amount'$, the check in Eq. 3 would be successful. Thus, the AAT would generate the code c and it would report c to the PAT. Then, the PAT could report to the remote server δ', mimicking an approval of the end-user on the forged transaction. However, being the relationship between the one-time-code and the transaction unique, the remote server could easily verify that both $\delta \neq \delta'$ and $c \neq c'$, denying the execution of the transaction.

Note that the considerations above apply also with *asynchronous* attacks, as the AAT would pop up an unsolicited notification to the end-user. Thus, the

end-user could easily realize the ongoing attack. That is, having the human in the loop is instrumental to the security of *FRACTAL*, reverting the saying that the human is the weakest link in the security chain.

In summary, the PAT has no control over the messages it routes from the remote server to the AATs. Therefore, *ADV* cannot modify such messages in a way to achieve its objectives. We formally verify this property in 4.2 via *ProVerif*.

4.2 Formal Security Analysis via ProVerif

We formally verified the security properties of *FRACTAL* using the automatic tool ProVerif [9], in line with many recent scientific contributions [10], [11], and we also released the source code at [12], to allow interested readers to reproduce our results and verify our claims. The logic of ProVerif is rooted on two main assumptions. First, the cryptographic primitives used within the security protocol are inherently robust. Second, the attacker is consistent with the widely-accepted Dolev-Yao model, having the capability to read, inject, delete, and modify all the messages exchanged on the communication channel. Based on the above assumptions and user-specified security objectives, ProVerif enables the formal analysis of secrecy and authentication properties [9].

We implemented *FRACTAL* in ProVerif to verify that, even when the PAT is controlled by *ADV*, the remote server could always discriminate forged and legitimate transactions. We implemented the flow of *FRACTAL* in *Scenario #1*, and we modeled the end-user through a simple process, verifying that the values possessed by the AAT are the same typed to the PAT. We also assumed that the mutual authentication step represented in Fig. 1 has been already executed, and that the result of such a process is a session key $TLSKey$, shared between the PAT and the remote server. Finally, to model the tampering of the PAT, we leaked the session key $TLSKey$ to the adversary, enabling PAT impersonation.

With reference to our security properties, ProVerif provides the output *not attacker(elem[]) is true* when the attacker does not know the value of *elem*, while the output *not attacker(elem[]) is false* is provided if the attacker knows the value of *elem*. Moreover, the output *inj-event(last_event ()) ==> inj-event(previous_event ()) is true* means that the function *last_event* is executed only when another function, namely *previous_event* is executed. Thus, as per the logic of the ProVerif tool, we defined two main events:

- *begin_EndUser(x_1,x_2)*, indicating that the End-User initiates a transaction specifying the values x_1 and x_2;
- *end_RS(x_1,x_2)*, indicating that the remote server completes a transaction with the values x_1 and x_2.

Fig. 4 shows the excerpt of the output of *ProVerif*, when executed locally (recall that the source code is available at [12]).

The first query verifies that the session key $TLSKey$ is known to *ADV*. As mentioned above, this condition models the tampering of the PAT. The second query verifies that the event *end_RS(x_1,x_2)*, occurring when the remote server

```
Verification summary:
Query not attacker(TLSKey[]) is false.
Query inj − event(end_RS(x1,x2)) ==> inj − event(begin_EndUser(x1,x2)) is
true.
```

Fig. 4. Excerpt of the output provided by the *ProVerif* tool.

completes a transaction with the values x_1 and x_2, happens if and only if the event *begin_EndUser(x_1,x_2)* has previously occurred. In turn, this means that the server completes a transaction with the end-user only when the end-user really initiated that transaction. Overall, the positive outcome of this query verifies that, thanks to *FRACTAL*, the remote server can always verify the end-user, even when the PAT is compromised.

5 Implementation and Performance Assessment

5.1 Implementation Details

We implemented *FRACTAL* in Java, using Spring [13]. Spring is an open-source application framework, containing a set of core features that can be used by any Java application. It also includes several extensions, that allow to build web applications on top of the Java Enterprise Edition (EE) platform. To allow type inference, conciseness, and inter-operation with mobile applications, we used the Kotlin programming language [14]. We rely on the non-relational database MongoDB to store transactions and the related details [15]. To efficiently manage cryptographic keys and X.509 certificates, we use *keytool* [16], while we used the *ZXing* library for barcode image processing [17]. Finally, we implemented the app running on the AATs using the AndroidStudio IDE [18].

We implemented the PAT and the *remote server* as standalone JAVA web applications on a dedicated machine, i.e., an Intel Core i7-3632QM, equipped with a CPU running at 2.20 GHz, 8.00 GB of RAM, and the Windows 10 Operating System (OS). Using a dedicated web application for the PAT, the user can login and insert the transaction details on a web page, and then the web app manages the interactions with the remote server. This approach allows separating the entities involved in the system, while introducing a negligible interaction delay. For the AAT, we used a *Xiaomi Mi A3* smartphone, running the Android 10 OS. Finally, our protype uses the *SHA-256* hashing algorithm and the *RSA-2048* public key signature scheme. As a reference example, in Figures 8 and 9 (included in Annex 7), we report the screen shown to the end-user on the AAT and the PAT to validate the transaction, respectively. The web application of the PAT requires $1,342$ MB of RAM, while the *apk* of the app is $3,259$ KB for the *Scenario #1* and $4,142$ KB for the *Scenario #2*.

5.2 Experimental Performance Assessment

For our experimental evaluation, we adopted a methodology inspired by the recent contribution in [19]. Specifically, the PAT, the AATs, and the remote server of our first scenario have been physically implemented in the same machine, thus being directly-connected to each other. However, we modeled a realistic deployment of the remote server in a random point of the World at the communication level, by introducing additional delays in the interaction between the PAT and the remote server. These additional latencies have been modeled by considering real end-to-end communication delays incurred between two real endpoints connected to the Internet. In detail, we identified ten (10) geographically distributed hosts publicly accessible through the Internet via their IP addresses, listed in Tab. 1. Then, we launched a set of $10,000$ *ICMP Echo Requests* to IP addresses

Table 1. Details of the hosts used for the modeling of a remote server.

ID	IP Address	Nation	Location
$S1$	39.32.0.1	Pakistan	Islamabad
$S2$	8.8.8.8	USA	Mountain View
$S3$	76.74.224.13	Canada	Vancouver
$S4$	61.69.229.154	Australia	Sydney
$S5$	193.70.52.72	France	Paris
$S6$	167.71.129.73	England	London
$S7$	80.116.252.221	Italy	Rome
$S8$	202.46.34.59	China	Shenzhen
$S9$	125.30.18.121	Japan	Tokyo
$S10$	139.59.140.10	Germany	Frankfurt

in Tab. 1 from an endpoint located in Doha, Qatar. Finally, we measured the Round Trip Time (RTT) values of the *ICMP Echo Requests*, as the difference between the time when the request is sent and the time when the corresponding *ICMP Echo Reply* is received. Then, the RTTs have been statistically modeled through an empirical Cumulative Distribution Function (CDF), shown in Fig. 5, and these curves have been used to model the time required to contact a specific remote server. Specifically, for each experiment, we located the remote server in one of the identified hosts, and we modeled the corresponding communication delay by extracting a random sample from the empirical CDF of this host in Fig. 5. Assuming a single AAT, the results of our investigation are reported in Fig. 6, along with the 95% confidence interval.

Note that the location of the *remote server* has a slight impact on the latencies. The highest (average) delay is observed for $S10$, located in Germany, with a mean value of the delay of 2.262 s. Overall, we can notice that the transaction can be completed in a limited time, not impacting on the usability of the solution, while guaranteeing high level of security.

To provide further insights, we also evaluated the time to complete a transaction, increasing the AATs. As a reference, we assumed a scenario consistent

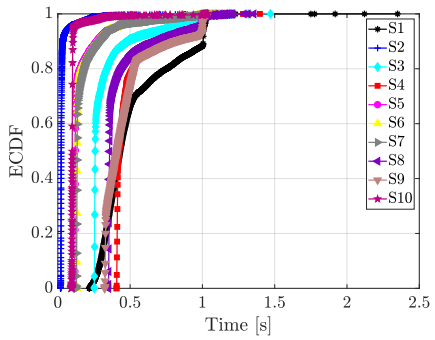

Fig. 5. CDF of the RTTs measured for 10 geographically-distributed hosts.

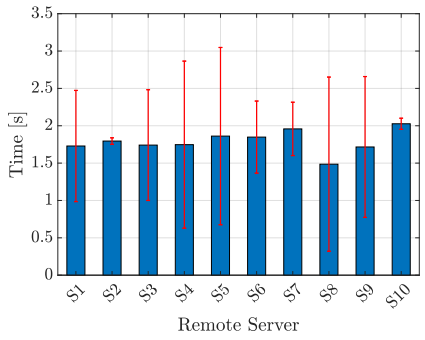

Fig. 6. Time to complete *FRACTAL*, with single AAT and single *remote server* located in Mountain View (S2), neglecting the time to input the code.

with the *Scenario #1*, where the AATs are connected to the PAT via Bluetooth, with the PAT being the hub of a logical star network topology. The results are provided in Fig. 7, along with the 95% confidence interval over 100 tests.

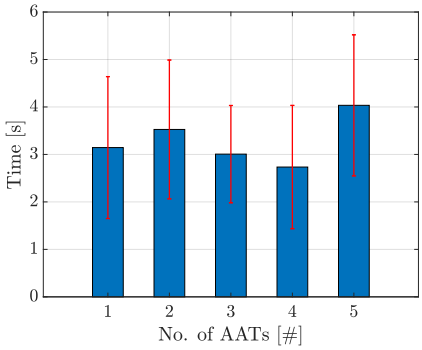

Fig. 7. Time required to complete *FRACTAL*, with multiple AATs directly connected to the PAT via Bluetooth.

As the number of AATs increases, the delay does not increase. Indeed, due to the non deterministic nature of devices interactions, when 3 and 4 AATs are connected, the average delay is less than what experienced for 1 and 2 AATs. Even in the worst case (5 AATs), the transaction can be completed in only 4.035 s, keeping the delay manageable. We remark that this computation delay does not consider the time for the user to input on the PAT.

6 Related Work and Qualitative Comparison

The majority of MFA systems in the literature focus on user/client MFA, using biometric factors to enforce client authentication. For instance, the authors in [20], proposed a two-phase authentication mechanism for federated identity management systems. Unfortunately, the reliance on biometrics prevents the use of legacy devices, not equipped with modules able to gather user biometric features. In [21], the authors addressed the problem of outsourcing the biometric features of users to untrusted servers, by designing a MFA scheme where the biometric features of users remain on their devices. Relying on biometric information, older devices are out of the equation as well. In [22], the authors introduced a zero-effort MFA system, requiring a smartphone and a smartwatch (replaceable by a smart bracelet) able to capture the gait patterns of an individual, her mid/lower body movements, and her wrist/arm movements. Given the alleged uniqueness of the combination of these patterns, only the account holder should be able to authenticate. Relying on specific sensors able to capture users' gait, the scheme is unusable in many contexts.

The authors in [23] introduced *graphical passwords* as new means for MFA. They proposed different architectures where such MFA could be employed, and they demonstrated the enhanced usability of their tool. However, their scheme relies on the delivery of SMS via out-of-band channels, not being usable when this is not available. A similar limitation can be found also in in [24]. The authors in [25] introduced a MFA technique specifically tailored to fragile communications. They contextualized the protocol in a smart grid scenario, where the user interacts with an Intelligent Electronic Device (IED) to perform the authentication. The interaction of users with online banking systems has been studied also by [26], where the authors proposed a MFA scheme without using additional devices to prove the identity of the user. However, the adversary models considered here do not assume tampering of the PAT. In a document released by the Federal Financial Institutions Examination Council [27], the authors embrace the new legal and technological changes with respect to the protection of customer information. However, this document only took into account the customer authentication, thus giving the banks' one for granted. Thus, under the assumption of a possible compromise of the PAT by an attacker, the mutual authentication feature could be easily disrupted. Note that many commercial solutions addresses two-factor and multi-factor authentication, based on the FIDO Alliance Specifications. An example is YubiKey, using *Security Keys*, i.e., hardware devices that authenticate the user after the user presses a button on the security key [28]. Requiring dedicated devices, such solutions are not applicable here. Finally, note that standard security protocols such as Extensible Authentication Protocol (EAP) cannot be used directly to solve our problem, since they all assume that the PAT is not compromised. We summarized the above discus-

Table 2. Qualitative comparison of *FRACTAL* against competing solutions.

Ref.	Single Channel	No Dedicated Devices as AATs	No Biometric Interfaces Required	Robust against PAT Tampering
[20]	✓	✗	✗	✓
[23]	✗	✓	✓	✓
[25]	✓	✗	✗	✓
[21]	✗	✗	✗	✓
[26]	✓	Not Applicable	Not Applicable	✗
[29]	✗	✓	✗	✓
[22]	✓	✗	✗	✓
[24]	✗	✓	✗	✓
FRACTAL	✓	✓	✓	✓

sion in Tab. 2. Note that all the schemes based on single-channel authentication are not effective when the the PAT is compromised. Moreover, they often require biometric interfaces on the supporting devices. Conversely, *FRACTAL* leverages a single communication channel, and it is robust also when the PAT is compromised. Besides, *FRACTAL* does not require additional dedicated interfaces on the supporting devices.

7 Conclusion

In this paper, we have presented *FRACTAL*, a flexible Multi-Factor Authentication scheme for securing banking transactions. *FRACTAL* uses a single communication channel shared between the bank servers and the Primary Authentication Terminal to provide security to the transaction, even if the Primary Authentication Terminal is compromised. The solution does not require any specialized hardware: it involves only (CoTS) Additional Authentication Terminals (AAT)—e.g. mobile phone, smart watch.
We discussed the security features of *FRACTAL*, and we proved its security using *ProVerif*. We also implemented it in a real client-server scenario, using *Spring* micro-services. Our experimental campaign demonstrated that, using a single AAT, independently from the remote server location, the transaction can be completed in about 2 seconds, while additional AATs can be added with just a slight impact on the end-to-end delay.
In addition to its striking security properties, we believe that our solution could also play a key role in overcoming the need for dedicated devices currently used by banks for MFA, and to reduce the need for separate out-of-band channels. Finally, the applicability of *FRACTAL* goes beyond the presented use-case, extending to other domains where additional communication channels are not available and dedicated devices are not suitable.

Acknowledgements

This work was supported by both the HBKU Technology Development Fund under contract TDF 02-0618-190005 and the NPRP-S-11-0109-180242 from the QNRF-Qatar National Research Fund. Both HBKU and QNRF are members of The Qatar Foundation. This work has been partially supported also by the INTERSCT project, Grant No. NWA.1162.18.301, funded by Netherlands Organisation for Scientific Research (NWO). The findings reported herein are solely responsibility of the authors.

References

1. F. Chandio, Z. Irani, A. Zeki, et al., "Online banking information systems acceptance: An empirical examination of system characteristics and web security," *Information Systs. Manag.*, vol. 34, no. 1, pp. 50–64, 2017.
2. G. Luo, et al. , "Overview of Intelligent Online Banking System Based on HERCULES Architecture," *IEEE Access*, vol. 8, 2020.
3. M. Carminati, R. Caron, F. Maggi, I. Epifani, and S. Zanero, "BankSealer: A decision support system for online banking fraud analysis and investigation," *Computers & Security*, vol. 53, pp. 175–186, 2015.
4. F. Sinigaglia, et al. , "A survey on multi-factor authentication for online banking in the wild," *Computers & Security*, p. 101745, 2020.
5. S. Kiljan, et al., "Evaluation of transaction authentication methods for online banking," *Future Generation Computer Systems*, vol. 80, pp. 430–447, 2018.

6. FIDO Alliance Specifications, https://fidoalliance.org/specifications, accessed: 2022-04-05.
7. R. Di Pietro, S. Sciancalepore, and S. Raponi, "Methods And Systems For Verifying The Authenticity Of A Remote Service," Jul. 2020, US Patent App. 16/657,088.
8. D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. on information theory*, vol. 29, no. 2, pp. 198–208, 1983.
9. B. Blanchet, et al., "ProVerif 2.02pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," Tech. Rep., Sep. 2020.
10. P. Tedeschi, S. Sciancalepore, A. Eliyan, R. Di Pietro, "LiKe: Lightweight Certificateless Key Agreement for Secure IoT Communications," *IEEE Internet of Things J.*, vol. 7, no. 1, pp. 621–638, 2020.
11. L. Hirschi and C. Cremers, "Improving Automated Symbolic Analysis of Ballot Secrecy for E-voting Protocols: A Method Based on Sufficient Conditions," in *IEEE Euro S&P*, 2019, pp. 635–650.
12. CRI-LAB, "Code of FRACTAL in ProVerif," https://github.com/cri-lab-hbku/tdf-proverif, 2021, (Accessed: 2022-04-05).
13. Spring Community, https://spring.io/why-spring, accessed: 2022-04-05.
14. Kotlin Foundation, https://kotlinlang.org/, accessed: 2022-04-05.
15. MongoDB, Inc., https://mongodb.com, accessed: 2022-04-05.
16. Oracle, https://tinyurl.com/y62ds856, accessed: 2022-04-05.
17. ZXing Project, https://github.com/zxing/zxing, accessed: 2022-04-05.
18. Jetbrains, https://developer.android.com/studio, accessed: 2022-04-05.
19. S. Sciancalepore, et al., "On the Design of a Decentralized and Multiauthority Access Control Scheme in Federated and Cloud-Assisted Cyber-Physical Systems," *IEEE Internet of Things J.*, vol. 5, no. 6, 2018.
20. A. Bhargav-Spantzel, et al. , "Privacy preserving multi-factor authentication with biometrics," *Journal of Computer Security*, vol. 15, no. 5, pp. 529–560, 2007.
21. Z. Han, L. Yang, and Q. Liu, "A Novel Multifactor Two-Server Authentication Scheme under the Mobile Cloud Computing," in *Intern. Conf. on Networking and Network Applications (NaNA)*, 2017, pp. 341–346.
22. B. Shrestha, M. Mohamed, and N. Saxena, "ZEMFA: Zero-Effort Multi-Factor Authentication based on Multi-Modal Gait Biometrics," in *Intern. Conf. on Privacy, Security and Trust (PST)*. IEEE, 2019, pp. 1–10.
23. A. P. Sabzevar and A. Stavrou, "Universal Multi-Factor Authentication Using Graphical Passwords," in *IEEE Int. Conf. on Signal Image Technology and Internet Based Systems*, 2008, pp. 625–632.
24. M. M. Mohammed and M. Elsadig, "A multi-layer of multi factors authentication model for online banking services," in *Intern. Conf. on Computing, Electrical And Electronic Engineering*, 2013, pp. 220–224.
25. X. Huang et al., "Robust Multi-Factor Authentication for Fragile Communications," *IEEE Trans. on Dependable and Secure Comput.*, vol. 11, no. 6, pp. 568–581, 2014.
26. S. Boonkrong, "Internet Banking Login with Multi-Factor Authentication," *KSII Trans. on Internet & Information Sys.*, vol. 11, no. 1, 2017.
27. Council, Federal Financial Institutions Examination, "Authentication in an internet banking environment," *FFIEC*, 2005.
28. J. Reynolds, et al., "A Tale of Two Studies: The Best and Worst of YubiKey Usability," in *IEEE Symp. on Security and Privacy (SP)*, 2018, pp. 872–888.
29. S. Nagaraju and L. Parthiban, "Trusted framework for online banking in public cloud using multi-factor authentication and privacy protection gateway," *Journal of Cloud Computing*, vol. 4, no. 1, p. 22, 2015.

Annex A

Fig. 8. Screen shown on the AAT to validate the transaction. $\mathcal{A}$ can verify that the details of the intended transaction match the ones on the screen. Then, in case of *Scenario #1*, $\mathcal{A}$ can validate the transaction by pressing *confirm*. In case of *Scenario #2*, $\mathcal{A}$ can insert the code on the PAT to verify the transaction (see Figure 9.)

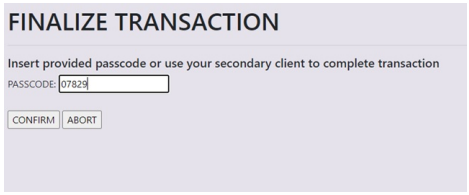

Fig. 9. Screen shown on the PAT to validate the transaction in case of *Scenario #2*. If the details of the transaction shown on the AAT match the intended ones, $\mathcal{A}$ can insert the code in the *passcode* field and press the *confirm* button to validate the transaction.