ZITA: Zero-Interaction Two-Factor Authentication using Contact Traces and In-band Proximity Verification

Nirnimesh Ghose*, Kaustubh Gupta*, Loukas Lazos†, Ming Li†, Ziqi Xu†, and Jincheng Li†
*School of Computing, University of Nebraska–Lincoln, USA
Email: {nghose, kgupta97}@unl.edu
†Department of Electrical and Computer Engineering, University of Arizona, USA
Email: {llazos, lim, zxu1969, jli2972}@arizona.edu

APPENDIX

In the appendix, first, we present the formal security analysis on ProVerif. Next, we present the proof-of-concept ZITA application. Followed by the user study undertaken using the application.

A. FORMAL SECURITY ANALYSIS ON PROVERIF

We verified the security of ZITA using the ProVerif 2.04 platform [1]. ProVerif is an automatic cryptographic protocol verifier, that can verify the security of a protocol under different attacker models. These models are specified in terms of the attacker's capabilities (passive or active), and the exposed secrets that are known by the adversary. ProVerif is based on the protocol representation using Horn clauses. For verification, we have defined all relevant use cases for ZITA. We represented ZITA using applied-pi calculus [2], [3]. First, we defined the basic primitives. Then, for each entity, we defined the processes, followed by their sequence as determined by the ZITA protocol specification.

Primitives: ZITA specifies the following primitive types and functions.

Declarations: :

1) **type** `mac` is the type for a keyed hash.
2) **type** `key` is the type for the random keys used in ZITA. This includes the key `key`, and the long-term shared key `longTermKey`.
3) **type** `nonce.` is the type for the pseudorandom nonce.
4) **type** `token.` is the type for the token generated by the primary and secondary device to send to the verifier.
5) **type** `contactTrace` is the type for the contact trace generated by the primary and secondary devices to the verifier.
6) **type** `requestToAuthenticate` is the type for the authentication initialization message generated by the verifier and transmitted to the primary and secondary devices.
7) **type** `promximitySamples` is the type for proximity samples collected by the primary and secondary device to send to the verifier.

We set `key`, one `contactTrace`, and `promximitySamples` as private because they are not compromised by the adversary according to our adversary model. Further, we defined the public channel as `c` and the private channel `p`.

Functions:

1) **MAC function** is a hash function `MAC(bitstring): mac` with $\forall$ `m:bitstring; n:mac; checkMAC(MAC(m), n) = m`.
2) **AE function:** The authenticated encryption function is realized as a MAC-then-Encrypt function. Both the MAC and the encryption functions are defined as symmetric key operations. We define the authenticated encryption function as `auE(bitstring, key): bitstring`, with $\forall$ `m: bitstring, k: key; auD(auE(m, k), k) = m` and $\forall$ `m: bitstring, k: key; checkMAC(auE(m, k), k) = m`.

We then defined inputs, outputs, and processes at different legitimate entities:

The primary and secondary devices generate `contactTrace` and measure `promximitySamples`. Each device generates the `token` with the current `contactTrace` and `nonce` sent by the verifier $\mathcal{V}$ using `mac` function. Each device transmits the `token` using `auE`. To simulate a compromise, `token` is transmitted over the public channel `c`, whereas the other `tokens` are transmitted over the private channel `p`. The primary device $\mathcal{D}$ and the verifier $\mathcal{V}$ evolved the `key` using the `mac` function.

Verification and Results: For the protocol verification, we considered the following use cases:

(a) In the benign case, the primary device $\mathcal{D}$ and secondary device $\mathcal{S}$ are in proximity and executing TFA with the verifier $\mathcal{V}$ in absence of the adversary $\mathcal{M}$.
Result: In this use case, all the legitimate entities achieve mutual authentication and key confidentiality is also achieved.

(b) The adversary with knowledge of the first factor credentials attempts to execute TFA with $\mathcal{V}$.
Result: The adversary $\mathcal{M}$ is not able to achieve mutual authentication with $\mathcal{V}$.

(c) The adversary with knowledge of the first factor credentials and temporarily compromising $\mathcal{S}$ to obtain the contact-trace attempts to execute TFA with $\mathcal{V}$. $\mathcal{M}$ does not have access to the valid RSS data. $\mathcal{D}$ and $\mathcal{S}$ have exchanged one heartbeat after the compromising has ended. They are in proximity and also attempting to execute TFA with the $\mathcal{V}$.
Result: All the legitimate entities achieve mutual authentication and key confidentiality is also achieved. The adversary $\mathcal{M}$ is not able to achieve mutual authentication with $\mathcal{V}$.

(d) The adversary $\mathcal{M}$ has compromised first factor credential and the latest contact-trace from $\mathcal{S}$. However, $\mathcal{D}$ and $\mathcal{S}$ are not in proximity and $\mathcal{M}$ does not have access to the valid RSS data. $\mathcal{M}$ attempts to execute TFA with $\mathcal{V}$.
Result: The adversary $\mathcal{M}$ is not able to achieve mutual authentication with $\mathcal{V}$.

B. Proof-of-Concept Application

The proof-of-concept application was developed on Android 10 platform. First, we describe the various entities. Followed by the different steps for authentication. The implementation on the Android 10 platform of the primary/secondary device app as well as the verifier are available at https://github.com/kaustubh-gupta/ZITA.

Login and Secondary Devices: We implemented the ZITA protocol on the Android 10 platform [4] running on two Samsung A9 mobile phones, one acting as the login device $\mathcal{D}$ and the other acting as the secondary device $\mathcal{S}$. The application consists of two parts: (a) the heartbeat exchange using Bluetooth connectivity and (b) the execution of the first and second-factor authentication using Wi-Fi connectivity. To generate the contact trace, the application verifies the connectivity between $\mathcal{D}$ and $\mathcal{S}$ using the BluetoothDevice and BluetoothAdapter modules. Heartbeats are transmitted and received using the BluetoothSocket module. The message exchange with the verifier $\mathcal{V}$ is implemented as a client-server application using java socket programming over Wi-Fi.

Verifier: We implemented the verifier $\mathcal{V}$ as a web application. The back-end of the web application consists of a java server that is running on a fixed port and is constantly accepting connections from any client on the same wireless network. The connection between $\mathcal{D}$ and $\mathcal{S}$ with the verifier is secured using CryptChat [5]. CryptChat was used to establish the session keys and secure sent and received messages encrypted with AES with authentication using a MAC based on MD5 (since this is a proof-of-concept implementation, we did not address well-known vulnerabilities of MD5 and MD5 was already part of our library). The front end of the web application is implemented using a Java GUI with relevant input fields for the listening port number and shows a notification once the authentication is completed. The messages sent over the network socket are encoded in BASE64.

Contact Trace: Figure 1 shows the contact tracing between $\mathcal{D}$ and $\mathcal{S}$. To collect the contact trace, $\mathcal{D}$ and $\mathcal{S}$ exchange heartbeats over Bluetooth 5.0. To initiate tracing, the $\mathcal{D}$ sends the RTC frame to the secondary device $\mathcal{S}$. The period of heartbeat exchange is set to one second, although longer periods can be set to accommodate lower power consumption. Reliability is ensured by the implementation of the stop-and-go protocol.

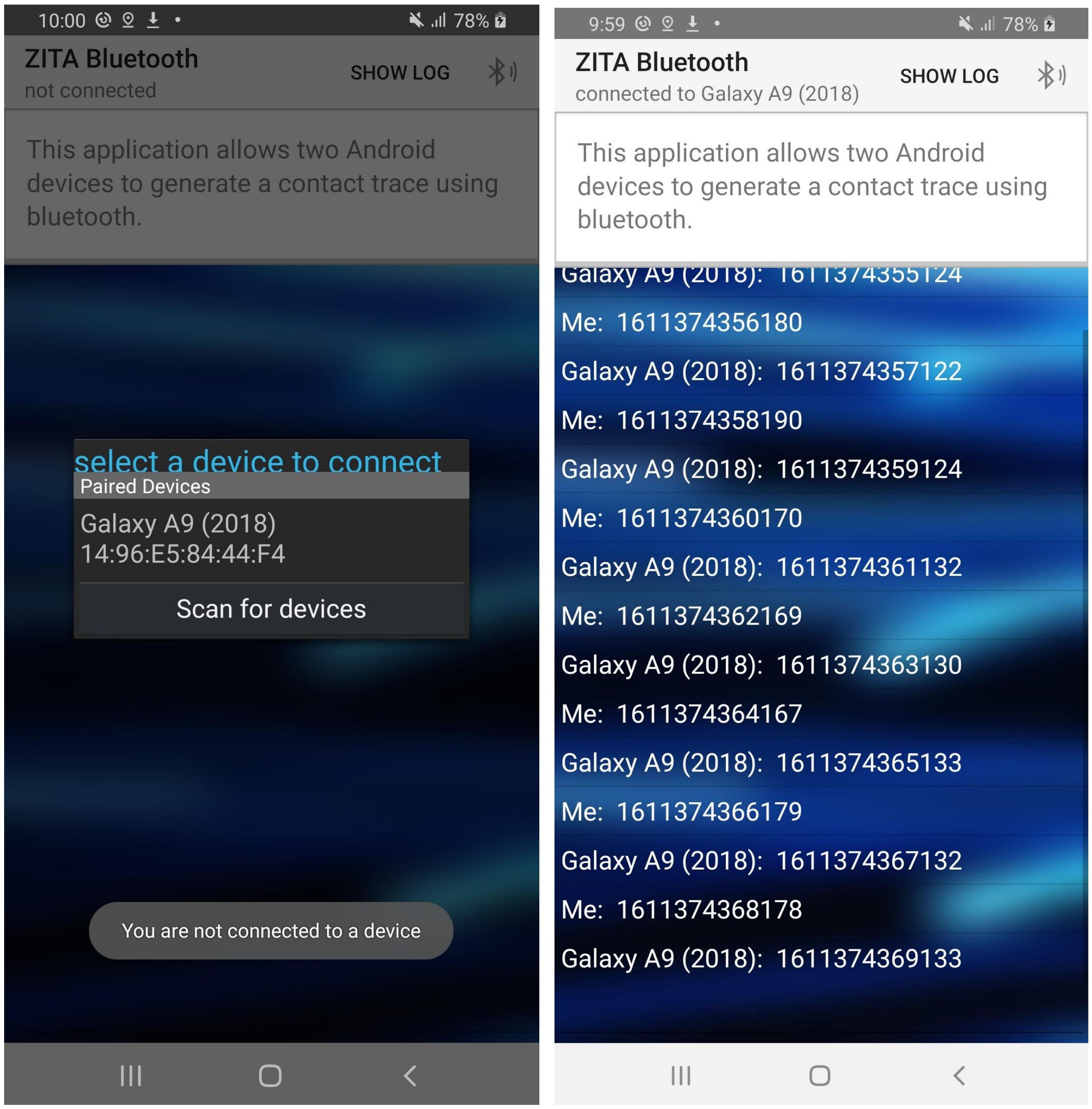
Setup Phase: The initial one-time setup phase includes establishing the connections over Wi-Fi by manually opening a port on the verifier and entering the verifier's IP address and port number on $\mathcal{D}$ and $\mathcal{S}$, as shown in Fig. 2. Further, the security of the $\mathcal{S}$-to-$\mathcal{V}$ channel is established by automatically generated long-term secret keys K_S.

First-factor AKA: The first-factor AKA is implemented using an authenticated DH key exchange, as shown in Fig. 3. It is initialized when the user enters their password and logs in on device $\mathcal{D}$. Upon a successful login, $\mathcal{D}$ and $\mathcal{S}$ establish two 128 bits session keys K_{VD} and K_{VS} with $\mathcal{V}$.

Second-factor: The second-factor is initiated right after the First-factor AKA is completed, as shown in Fig. 4. The authentication process at $\mathcal{D}$ is performed in an automated fashion and it is seamless to the user. The identity of $\mathcal{D}$ is verified using the token $TK'_{VD} = f(TK_D, K_{VD})$, where $TK_D = f(\eta, Tr_D)$. To keep the application secure, we have hidden the session key update and mutual authentication of both $\mathcal{S}$ and $\mathcal{D}$ with $\mathcal{V}$.

C. User Study

We conducted a user study on the usability of ZITA by developing an application on the Android 10 platform. The user study was approved as exempt by the Institutional Review Board (IRB) of the University of Nebraska – Lincoln.



This application allows two Android devices to generate a contact trace using bluetooth.

```
Galaxy A9 (2018):  1611374355124
Me:  1611374356180
Galaxy A9 (2018):  1611374357122
Me:  1611374358190
Galaxy A9 (2018):  1611374359124
Me:  1611374360170
Galaxy A9 (2018):  1611374361132
Me:  1611374362169
Galaxy A9 (2018):  1611374363130
Me:  1611374364167
Galaxy A9 (2018):  1611374365133
Me:  1611374366179
Galaxy A9 (2018):  1611374367132
Me:  1611374368178
Galaxy A9 (2018):  1611374369133
```

Fig. 1. Setup of the login and secondary devices to generate contact trace and exchange of nonces (time in ms) every 1 sec between login and secondary devices.

Study design: Due to COVID restrictions, only 21 candidates were recruited to conduct the user study. The candidate pool was 76% male and 24% female in the age range of 21-56 years. The study took place over several days in a lab space and was conducted in person. The participants first received a brief introduction to the concept of TFA and the conductor observed the participants' familiarity with SMS-based TFA, Duo, YubiKey, and Google Authenticator. The users were then given a brief introduction to the purpose of ZITA. Next, the study conductor demonstrated the functionality of the ZITA application. Each user was asked to log into a web server using the ZITA application. Upon completion, users were asked to complete the System Usability Scale (SUS) [6] and a qualitative questionnaire.

Results: The candidates evaluated ZITA with an average SUS score of 71 with a standard deviation of 14. This will rate ZITA as "Good" and "Acceptable" according to the scale proposed by Bangor *et al.* [7]. Most of the users found ZITA to be "Seamless" after an easy setup process. However, one of the candidates pointed out that non-tech-savvy users may question safety due to the automated nature of the second factor. For comparison, [8] performed a usability study on five common TFA methods on a completely different set of users: Short Message/Messaging Service (SMS), Time-based one-time password (TOTP), pre-

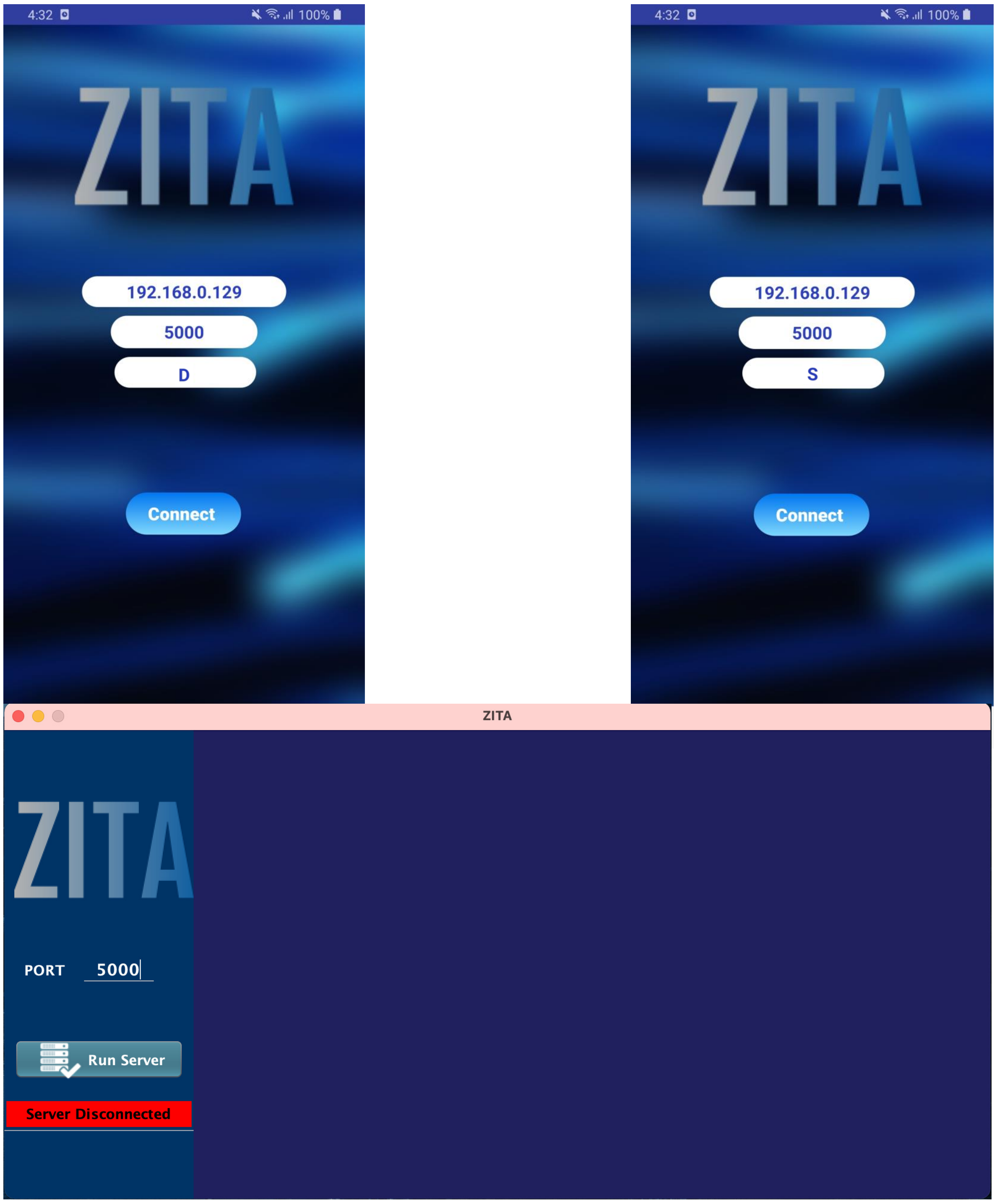


Fig. 2. Setup of the login and secondary devices with the verifier.

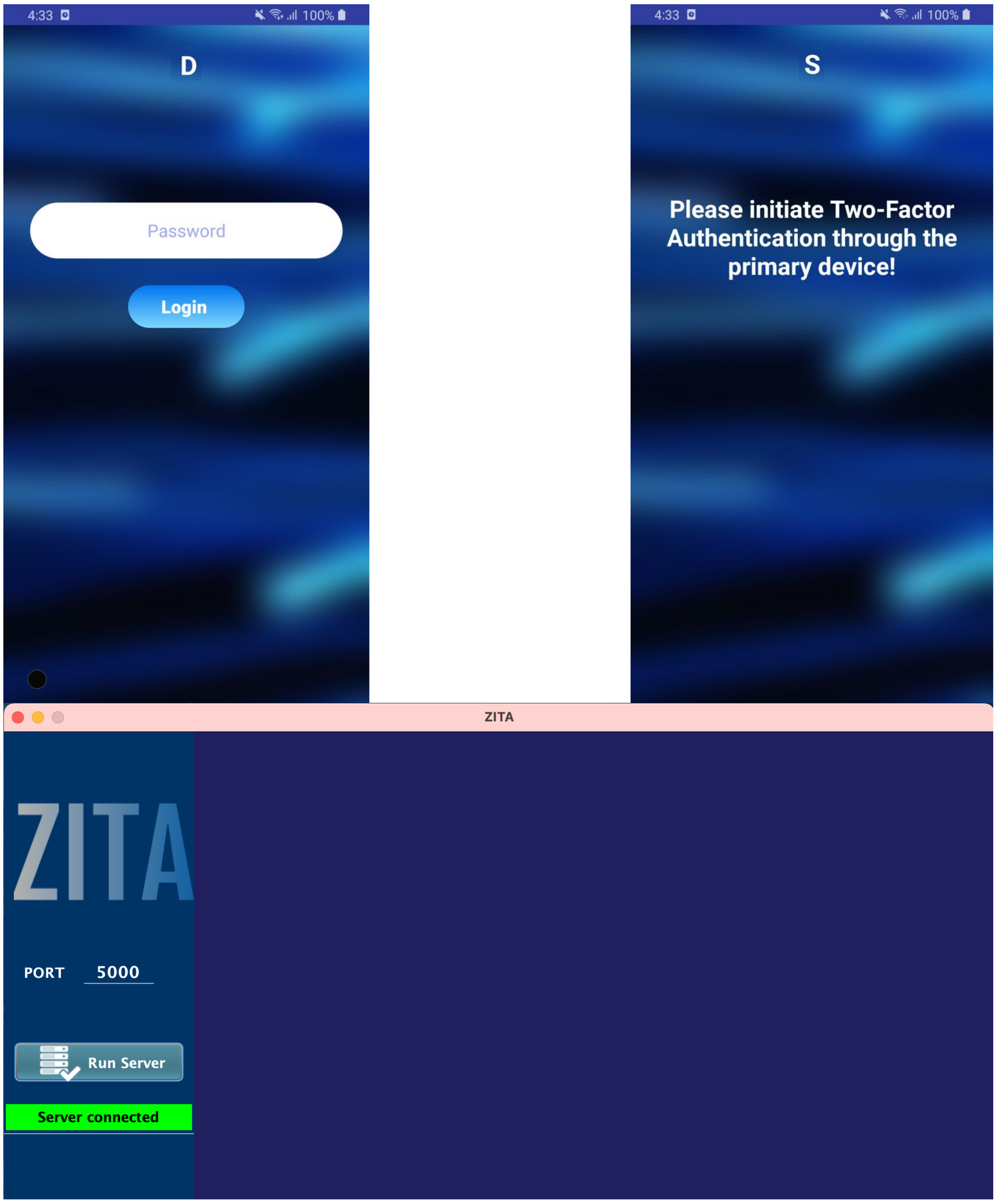
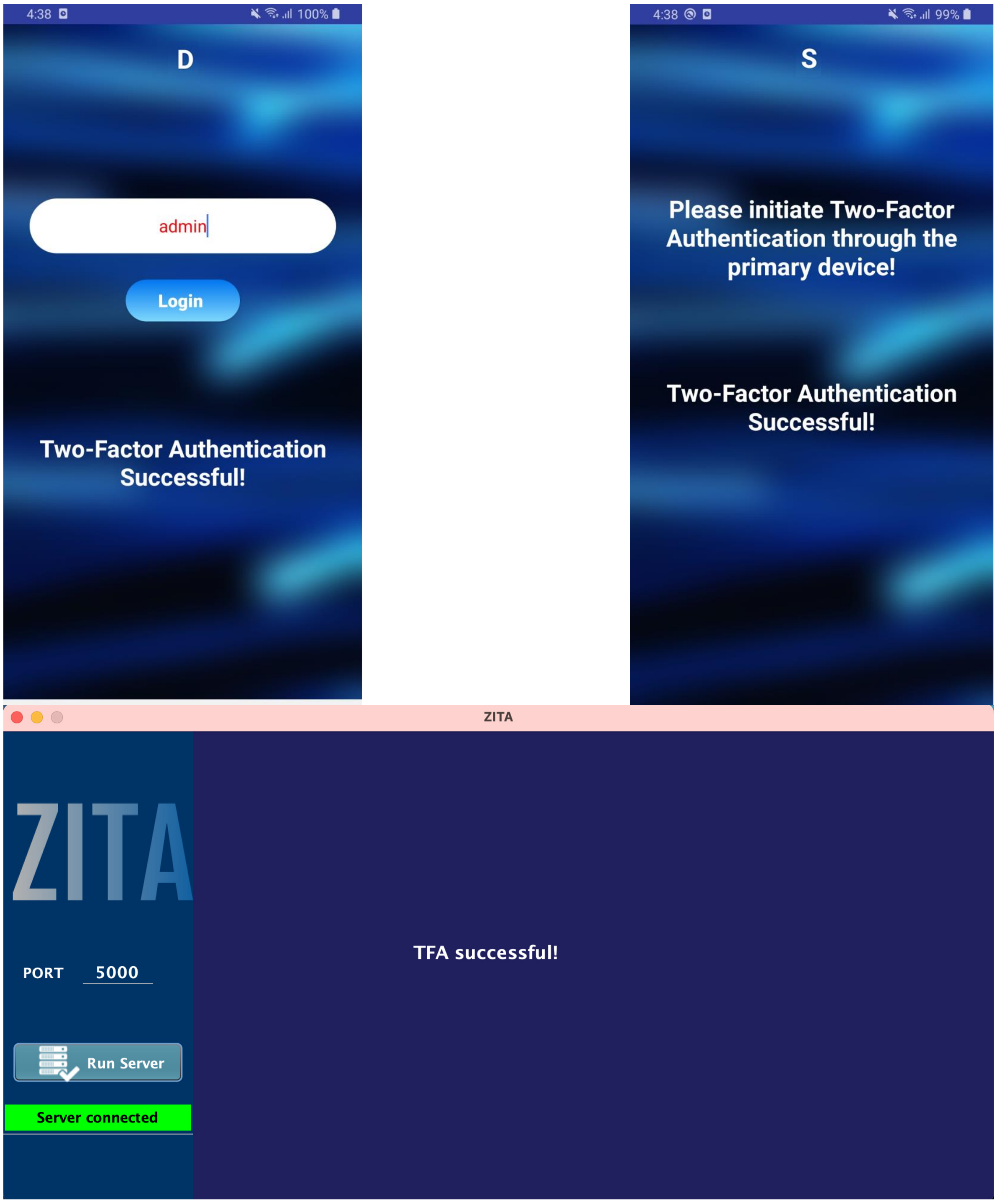


Fig. 3. Implementation of the pairwise first-factor AKA between devices $\mathcal{D}, \mathcal{V}$ and the verifier $\mathcal{V}$.



Two-Factor Authentication Successful!

Please initiate Two-Factor Authentication through the primary device!

Two-Factor Authentication Successful!

TFA successful!

Fig. 4. Implementation of the zero-interaction second-factor AKA between devices $\mathcal{D}, \mathcal{V}$ and the verifier $\mathcal{V}$.

generated codes, push, and Universal 2nd Factor (U2F) security keys with 72 participants (55% female and 45% male). They designed a sign-on application simulating a banking website that incorporated all five methods. The SUS scores for each of them were reported as Password (no TFA) 92.5, Codes 80.2, Push 81.0, SMS 75.0, TOTP 83.1, and U2F 73.1.

REFERENCES

[1] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "ProVerif: Cryptographic protocol verifier in the formal model," http://prosecco.gforge.inria.fr/personal/bblanche/proverif/, 2021.

[2] M. D. Ryan and B. Smyth, "Applied pi calculus," *Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols*, 2011.

[3] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *ProVerif 2.04: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, The organization, 2021. [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf

[4] Google, "Android 10," https://www.android.com/android-10/, 2019, [Online; accessed 16-Nov-2020].

[5] ejupialked, "Cryptchat," https://github.com/ejupialked/chat-diffie-hellman, 2020, [Online; accessed 01-Dec-2020].

[6] U. G. S. A. T. T. Services, "System usability scale (sus)," https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html, 2021, [Online; accessed 05-Aug-2021].

[7] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.

[8] K. Reese, T. Smith, J. Dutson, J. Armknecht, J. Cameron, and K. Seamons, "A usability study of five {Two-Factor} authentication methods," in *Proc. of Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019, pp. 357–370.