Two-factor Password-authenticated Key Exchange with End-to-end Security

STANISLAW JARECKI, University of California Irvine
MOHAMMED JUBUR, University of Alabama at Birmingham
HUGO KRAWCZYK, Algorand Foundation
NITESH SAXENA, University of Alabama at Birmingham
MALIHEH SHIRVANIAN, Visa Research

We present a secure two-factor authentication (TFA) scheme based on the user's possession of a password and a crypto-capable device. Security is "end-to-end" in the sense that the attacker can attack all parts of the system, including all communication links and any subset of parties (servers, devices, client terminals), can learn users' passwords, and perform active and passive attacks, online and offline. In all cases the scheme provides the highest attainable security bounds given the set of compromised components. Our solution builds a TFA scheme using any Device-enhanced Password-authenticated Key Exchange (PAKE), defined by Jarecki et al., and any Short Authenticated String (SAS) Message Authentication, defined by Vaudenay. We show an efficient instantiation of this modular construction, which utilizes any password-based client-server authentication method, with or without reliance on public-key infrastructure. The security of the proposed scheme is proven in a formal model that we formulate as an extension of the traditional PAKE model. We also report on a prototype implementation of our schemes, including TLS-based and PKI-free variants, as well as several instantiations of the SAS mechanism, all demonstrating the practicality of our approach. Finally, we present a usability study evaluating the viability of our protocol contrasted with the traditional PIN-based TFA approach in terms of efficiency, potential for errors, user experience, and security perception of the underlying manual process.[1]

CCS Concepts: • **Security and privacy** → *Distributed systems security*; *Usability in security and privacy*; *Multi-factor authentication*;

Additional Key Words and Phrases: Authentication, two factor authentication, passwords

17

[1]This submission is an extension of our work published in PKC 2018 [46].

Hugo Krawczyk Work done at IBM Research.
Maliheh Shirvanian Work done at UAB.
This work has been supported in part by NSF Grants No. CNS-1714807, No. CNS-1526524, No. OAC-1547350, and No. CNS-2030501 and Jazan University.
Authors' addresses: S. Jarecki, University of California Irvine, Irvine, CA, USA, 92697; email: sjarecki@uci.edu; M. Jubur and N. Saxena, University of Alabama at Birmingham, Birmingham, AL, USA, 35294; emails: {mjabour, saxena}@uab.edu; H. Krawczyk, Algorand Foundation, New York, NY, USA, 10025; email: hugokraw@gmail.com; M. Shirvanian, Visa Research, Palo Alto, CA, USA, 94306; email: mshirvan@visa.com.

2471-2566/2021/04-ART17 $15.00
https://doi.org/10.1145/3446807

ACM Reference format:
Stanislaw Jarecki, Mohammed Jubur, Hugo Krawczyk, Nitesh Saxena, and Maliheh Shirvanian. 2021. Two-factor Password-authenticated Key Exchange with End-to-end Security. *ACM Trans. Priv. Secur.* 24, 3, Article 17 (April 2021), 37 pages.
https://doi.org/10.1145/3446807

1 INTRODUCTION

Passwords provide the dominant mechanism for electronic authentication, protecting a plethora of sensitive information. However, passwords are vulnerable to both *online* and *offline* attacks. A network adversary can test password guesses in online interactions with the server while an attacker who compromises the authentication data stored by the server (i.e., a database of salted password hashes) can mount an *offline dictionary attack* by testing each user's authentication information against a dictionary of likely password choices. Offline dictionary attacks are a major threat, routinely experienced by commercial vendors, and they lead to the compromise of *billions* of user accounts [1, 2, 4, 5, 8, 9]. Moreover, because users often re-use their passwords across multiple services, compromising one service typically also compromises user accounts at other services.

***Two-factor password authentication* (TFA)**, where user U authenticates to server S by "proving possession" of an auxiliary personal device D (e.g., a smartphone or a USB token) in addition to knowing her password, forms a common defense against *online* password attacks as well as a second line of defense in case of password leakage. A TFA scheme, which uses a device that is not directly connected to U's client terminal C, typically works as follows: D displays a short one-time secret PIN, either received from S (e.g., using an SMS message) or computed by D based on a key shared with S, and the user manually types the PIN into client C in addition to her password. Examples of systems that are based on such one-time PINs include SMS-based PINs, TOTP [63], HOTP [62], Google Authenticator [14], FIDO U2F [13], and schemes in the literature such as Reference [67].

Vulnerabilities of traditional TFA schemes. Our work addresses a large set of vulnerabilities unresolved by the current practice of composing the standard password-over-TLS authentication with PIN-based TFAs. These vulnerabilities include:

- Password is always visible to the server at the TLS decryption endpoint leading to password exposure to insiders and accidental storage of plaintext passwords [12, 15].
- Password is open to PKI attacks and exposure at midpoints (e.g., TLS decryption points for content inspection, at CDNs, etc.).
- Password is vulnerable to offline dictionary attacks upon compromise of the user's state ("password file") at the server.
- Password guesses can be validated through login attempts at the server (not prevented by TFA schemes that first verify the password and only then activate the TFA mechanism).
- TFA defense is broken if keys shared between TFA device and server leak to the attacker (e.g., they are stolen from the server).
- Low-entropy PINs have non-negligible probability of being guessed, e.g., PIN guessing can be used in a large-scale online attack against accounts whose passwords the attacker already collected [4, 5, 8, 9].
- PIN sent from server to device is vulnerable to PIN redirection attacks, e.g., via SMS hijacking and SIM card swap attacks [6].[2]

[2]PINs diverted to the attacker's phone exploiting SS7 vulnerabilities [54] led to NIST's recent decision to deprecate SMS PINs as a TFA mechanism [7].

- PIN entered by user into the host computer is vulnerable to eavesdropping via shoulder-surfing, PIN recording, keyloggers, screen scrapers [58], PIN phishing [49], and so on. (Note that some eavesdropping attacks are also possible with high-entropy PIN's such as QR codes.)

The first two vulnerabilities, specific to password-over-TLS, can be addressed by replacing this protocol with a PKI-free **asymmetric PAKE (aPAKE)** (e.g., Reference [47]). The other weaknesses are prevented by our TFA design even if used with password-over-TLS!

Our Contributions. *The main contribution of the present article is the design of a device-based TFA and PAKE solution that eliminates all of the above weaknesses.* Particularly, we: (1) introduce a precise security model for TFA schemes capturing well-defined *maximally-attainable* security bounds, (2) exhibit a practical TFA scheme, which we prove to achieve the strong security guaranteed by our formal model, and (3) prototype several methods for validating user's possession of the secondary factor, and evaluating usability of each method.

TFA Security Model with End-to-end Security. We introduce a ***Two-factor Authenticated Key Exchange (TFA-KE)*** model in which a user authenticates to server S by (1) entering a password into client terminal C and (2) proving possession of a personal device D, which forms the second authenticator factor by the user confirming in the device equality of a t-bit *checksum* displayed by D with a *checksum* displayed by C. Following Reference [72] (see below), this implements a t-bit C-to-D *user-authenticated channel*, which confirms that the same person is in control of client C and device D. This channel authentication requirement is weaker than the *private* channel required by current PIN-based TFAs and, as we show, it allows TFA schemes to be both more secure *and* easier to use.

The TFA-KE model, that we define as an extension of the standard **Password-authenticated Key Exchange (PAKE)** [25] and the **Device-enhanced PAKE (DE-PAKE)** [45] models, captures what we call *end-to-end security* by allowing the adversary to *control all communication channels and compromise any protocol party*. For each subset of compromised parties, the model specifies *best-possible security bounds*, leaving inevitable exhaustive *online* guessing attacks as the only feasible attack option. In particular, in the common case that D and S are uncorrupted, the only feasible attack is an active *simultaneous online* attack against *both* S and D that also requires guessing the password *and* the t-bit checksum. Compromising server S allows the attacker to impersonate S, but does not help in impersonating the user to S, and in particular does not enable an offline-dictionary attack against the user's password. Compromising device D makes the authentication effectively password-only, hence offering best possible bounds in the PAKE model (in particular, the offline dictionary attack is possible only if D and S are both compromised). Finally, compromising client C leaks the password, but even then impersonating the user to the server requires an active attack on D. We prove our protocols in this strong security model.

Practical TFA with End-to-end Security. Our main result is a TFA scheme, GenTFA that achieves end-to-end security as formalized in our TFA-KE model and is based on two general tools. The first is a DE-PAKE scheme as introduced by Jarecki et al. [45]. Such a scheme assumes the availability of a user's auxiliary device, as in our setting, and utilizes the device to protect against offline dictionary attacks in case of server compromise. However, DE-PAKE schemes provide no protection in case that the client machine C is compromised and, moreover, security completely breaks down if the user's password is leaked. Thus, our approach for achieving TFA-KE security is to start with a DE-PAKE scheme and armor it against client compromise (and password leakage) using our second tool, namely, a SAS-MA (Short-Authentication-String Message Authentication) as defined by Vaudenay [72]. In our application, a SAS-MA scheme utilizes a

t-bit user-authenticated channel, called a *SAS channel*, to authenticate data sent from C to D. More specifically, the SAS channel is implemented by having the user verify and confirm the equality of two t-bit strings, called *checksums*, displayed by both C and D. It follows from Reference [72] that if the displayed checksums coincide then the information received by D from C is correct except for a 2^{-t} probability of authentication error. We then show how to combine a DE-PAKE scheme with such a SAS channel to obtain a scheme, GenTFA, for which we can prove TFA-KE security, hence provably avoiding the shortcomings of PIN-based schemes. Moreover, the use of the SAS channel relaxes the required user's actions from a read-and-copy action in traditional schemes to a simpler compare-and-confirm, which also serves as a proof of physical possession of the device by the user (see more below).

We show a concrete *practical* instantiation of our general scheme GenTFA, named OpTFA, that inherits from GenTFA its TFA-KE security. Protocol OpTFA is modular with respect to the (asymmetric) password protocol run between client and server, thus it can utilize protocols that assume PKI as the traditional password-over-TLS, or those that do not require any form of secure channels, as in the (PKI-free) asymmetric PAKE schemes [26, 39]. In the PKI case, OpTFA can run over TLS, offering a ready replacement of current TFA schemes in the PKI setting. In the PKI-free case one gets the advantages of the TFA-KE setting without relying on PKI, thus obtaining a strict strengthening of (password-only) PAKE security [25, 59] as defined by the TFA-KE model.

The cost of OpTFA is two communication rounds between D and C, with 4 exponentiations by C and 3 by D, a one-round Diffie-Hellman exchange between C and S, plus the cost of a password authentication protocol between C and S. In the PKI setting the latter is the cost of establishing a server-authenticated TLS channel, while in the PKI-free case one can use an asymmetric PAKE (e.g., Reference [47]) with cost (some of it computable offline) of three exponentiations for C, two for S, and one multi-exponentiation for each.

Implementation and SAS Channel Designs. We prototyped protocol OpTFA, in both the PKI and PKI-free versions, with the client implemented as a Chrome browser extension, the device as an Android app, and D-C communication implemented using Google Cloud Messaging. We also designed and implemented several instantiations of the human-assisted C-to-D SAS channel required by our TFA-KE solution and model. Recall that a SAS channel replaces the user's *read-and-copy* action of a PIN-based TFA with the *compare-and-confirm* action used to validate the checksums displayed by C and D. The security of a SAS-model TFA-KE depends on the checksum entropy t, called the *SAS channel capacity,* hence the two important characteristics of a physical design of a SAS channel are its capacity t and the ease of the compare-and-confirm action required of the user. In Section 6, we show several SAS designs with different channel capacity and usability.

Our base-line implementation of a SAS channel encodes 20-bit checksums as six-digit decimal PINs, which the user compares when displayed by C and D (no copying involved). We also propose two novel and higher-capacity SAS channels. In the first design, the device D is assumed to have a camera and the checksum calculated by the client is encoded as a QR code and displayed by C. The user prompts D to capture this QR code, which D decodes and compares against its own checksum. The second design is based on an audio channel implemented using speech transcription. If device D is a smartphone, then the user can read out an alphanumeric checksum displayed by C into D's microphone,[3] and D decodes the audio using the transcriber tool and compares it to its checksum.

Usability Study of our SAS Channel Designs. Perhaps the most interesting aspect of OpTFA, from the usability perspective, is that the user interaction in this method is changed from copying

[3]Thanks to the full resistance of our TFA-KE to eavesdropping, overhearing the spoken checksum is of no use to the attacker.

the PIN (as in PIN-based TFA) to verifying the equality of the checksums. The hypothesis is that such verification provides higher usability compared to manual PIN copying of PIN-based TFA, while the use of a full-size PIN over the authenticated and secure channel improves the security of TFA. Also, while in the OpTFA, the SAS-MA protocol is used in conjunction with DE-PAKE, its use could be extended to any other standard TFA method to improve the security against online guessing and offline dictionary attacks, and to provide resistance against PIN eavesdropping by authenticating the device-client channel. Hence, it is important to evaluate the usability of such strong protocol.

We ran a lab-based study with 30 participants and asked them to use each of the aforementioned SAS transfer methods and the traditional TFA PIN entry (as the baseline of the study) multiple times as part of the login procedure to a website we created for this study. We recorded the participants' responses to analyze the user errors that might occur while inputting the PIN or checksum, as well as errors that might occur due to the transcriber, or the QR code decoder while automatically verifying the checksum. We also recorded the time it takes to perform each of the tasks to measure the efficiency/delay overhead of each method. We then asked the participants to answer several questions about the usability of the system, and their perception regarding the adoptability, security, trust, and efficiency of the system. Our results showed that OpTFA could provide a higher usability compared to PIN-TFA if the QR code checksum comparison method is to be deployed. This method also seems to be more efficient compared to other approaches and offers higher usability in terms of user perception. Our study design is in line with other TFA studies [35, 40, 73–75].

Contribution over Conference Publication. This work is an extension of our earlier PKC 2018 publication [46], in which we introduced our end-to-end secure two-factor authentication scheme. In this publication, we extend our PKC 2018 work by reporting on an extensive usability study of the PKC 2018 protocol and comparing it as the baseline to the traditional PIN-based two-factor authentication. We also include cryptographic proofs of the security theorems claimed in Reference [46].

2 TFA-KE SECURITY

We introduce the *TFA-KE* security model that defines the assumed environment and participants in our protocols as well as the attacker's capabilities and the model's security guarantees. Our starting point is the *DE-PAKE* model, introduced in Reference [45], which extends the well-known two-party *PAKE* model [25] to a multi-party setting that includes users U, communicating from client machines C, servers S to which users log in, and auxiliary *devices* D, e.g., a smartphone. A DE-PAKE scheme has the security properties of a **two-server PAKE (2-PAKE)** [30, 53] where D plays the role of the second server. Namely, a compromise of either S or D (but not both) essentially does not help the attacker, and in particular leaks no information about the user's password. However, a DE-PAKE scheme has the additional crucial property that even an adversary who compromises both S and D must still stage an offline dictionary attack to learn the password.

The TFA-KE model considers the same set of parties as in the DE-PAKE model and all the same adversarial capabilities, including controlling all communication links, the ability to mount online active attacks, offline dictionary attacks, and to compromise devices and servers. However, the DE-PAKE model does not consider client corruption or password leakage. Indeed, in case of password leakage an active adversary can authenticate to S by impersonating the legitimate user in a single DE-PAKE session with D and S. Since a TFA scheme is supposed to protect against the client corruption and password leakage attacks, our TFA-KE model enhances the DE-PAKE model by adding these capabilities to the adversary while preserving all the other strict security requirements of DE-PAKE. DE-PAKE requirements were such that the only allowable attacks,

under a given set of corrupted parties, are the unavoidable exhaustive online guessing attacks for that setting; the same holds for TFA-KE but with additional best resilience to client compromise and password leakage.

Note, however, that if C, D, S communicate only over insecure links then an attacker who learns the user's password will always be able to authenticate to S, by impersonating the user to D and S. Consequently, to allow device D to become a true *second factor* and maintain security in case the password leaks, one has to assume some form of authentication in the C to D communication, which would allow the user to validate that D communicates with the user's own client terminal C and not with the attacker who performs a man-in-the-middle attack and impersonates this user to D.

To that end our TFA-KE model augments the communication model by an authentication abstraction on the client-to-device channel, but it does so without requiring the client to store any long-term keys (other than the user's password). Namely, we assume a uni-directional C-to-D "Short Authenticated String" (SAS) channel, introduced by Vaudenay [72], which allows C to communicate t bits to D that cannot be changed by the attacker. The t-bit C-to-D SAS channel abstraction comes down to a requirement that the human user compares a t-bit *checksum* displayed by both C and D, and approves (or denies) their equality by choosing the corresponding option on device D.

As is standard, we quantify security by attacker's resources that include the computation time and the number of instances of each protocol party the adversary interacts with. We denote these as q_D, q_S, q_C, q'_C, where the first two count the number of active sessions between the attacker and D and S, respectively, while q_C (respectively, q'_C) counts the number of sessions where the attacker poses to C as S (respectively, as D). Security is further quantified by the password entropy d (we assume the password is chosen from a dictionary of size 2^d known to the attacker), and parameter t, which is called the SAS channel *capacity*. As we explain in Section 3, a C-to-D SAS channel allows for establishing a confidential channel between D and C, except for the 2^{-t} probability of error [72], which explains 2^{-t} factors in the TFA-KE security bounds stated below.

TFA Security Definition. We consider a communication model of open channels plus the t-bit SAS-channel between C and D, and a man-in-the-middle adversary that interacts with q_D, q_S, q_C, q'_C sessions of D, S, C, as described above. The adversary can also corrupt any party, S, D, or C, learning its stored secrets and the internal state as that party executes its protocol, which in the case of C implies learning the user's password. All other adversarial capabilities, as well as the test session experiment, are as in the DE-PAKE model, and we refer to Reference [45] for the detailed exposition of this model. In particular, the adversary's advantage is in distinguishing between a random string and a key computed by S or C on a tested session, and this advantage can be intuitively understood as the probability that the adversary successfully attacks *any* session key output by either client C or server S in the course of adversary's interaction with the TFA scheme.

The security requirements set by Definition 2.1 below are the *strictest* one can hope for given the communication and party corruption model. That is, wherever we require the attacker's advantage to be no more than a given bound with a set of corrupted parties, then there is an (unavoidable) attack - in the form of exhaustive guessing attack - that achieves this bound under the given compromised parties. Importantly, and *in contrast to typical two-factor authentication solutions*, the TFA-KE model requires that the second authentication factor D not only provides security in case of client and/or password compromise, but that *it also strengthens online and offline security (by 2^t factors) even when the password has not been learned by the attacker.*

Definition 2.1. A TFA-KE protocol TFA is (T, ϵ)-secure if for any dictionary Dict of size 2^d, t-bit SAS channel, and attacker A bounded by time T, attackers's advantage $\mathrm{Adv}^{\mathrm{TFA}}_{\mathrm{A}}$ in

distinguishing the tested session key from random is bounded as follows, for q_S, q_C, q'_C, q_D as defined above:

(1) If S, D, and C are all uncorrupted, then
$$\mathrm{Adv}_{\mathrm{A}}^{\mathrm{TFA}} \leq \min\{q_C + q_S/2^t, q'_C + q_D/2^t\}/2^d + \epsilon.$$
(2) If only D is corrupted, then $\mathrm{Adv}_{\mathrm{A}}^{\mathrm{TFA}} \leq (q_C + q_S)/2^d + \epsilon$.
(3) If only S is corrupted, then $\mathrm{Adv}_{\mathrm{A}}^{\mathrm{TFA}} \leq (q'_C + q_D/2^t)/2^d + \epsilon$.
(4) If only C is corrupted (or the user's password leaks by any other means), then $\mathrm{Adv}_{\mathrm{A}}^{\mathrm{TFA}} \leq \min(q_S, q_D)/2^t + \epsilon$.
(5) If both D and S are corrupted (but not C), and $\overline{q}_S$ and $\overline{q}_D$ count A's offline operations performed based on, respectively, S's and D's state, then $\mathrm{Adv}_{\mathrm{A}}^{\mathrm{TFA}} \leq \min\{\overline{q}_S, \overline{q}_D\}/2^d$.

Explaining Security Bounds. The security of the TFA scheme relative to the DE-PAKE model can be seen by comparing the above bounds to those in Reference [45]. Here, we explain the meaning of some of these bounds. In the default case of no corruptions, the adversary's probability of attack is at most $\min(q_C+q_S/2^t, q'_C+q_D/2^t)/2^d$ improving on DE-PAKE bound $\min(q_C+q_S, q'_C+q_D)/2^d$ and on the PAKE bound $(q_C+q_S)/2^d$. For simplicity, assume that $q_C = q'_C = 0$ (e.g., in the PKI setting where C talks to S over TLS and the communication from D to C is authenticated), in which case the bound reduces to $\min(q_S, q_D)/2^{t+d}$. The interpretation of this bound, and similarly for the other bounds in this model, is that to have a probability $q/2^{t+d}$ to impersonate the user, the attacker needs to run q online sessions with *S and also* q online sessions with *D*. (In each such session, the attacker can test one password out of a dictionary of 2^d passwords and can do so successfully only if its communication with D is accepted over the SAS channel, which happens with probability 2^{-t}.) This is the optimal security bound in the TFA-KE setting, since an adversary who guesses both the user's password and the t-bit checksum can successfully authenticate as the user to the server.

In case of client corruption (and password leakage), the adversary's probability of impersonating the user to the server is at most $\min(q_S, q_D)/2^t$, which is the best possible bound if the password leaks. In case of device corruption, the adversary's advantage is at most $(q_C+q_S)/2^d$, which matches the optimal advantage of PAKE, i.e., where there is no device. In case of server corruption, the adversary's probability of impersonating the user to an uncorrupted server session is (assuming $q'_C = 0$ for simplicity) at most $q_D/2^{t+d}$. In other words, learning server's private information allows the adversary to authenticate as the server to the client, but it does not help to impersonate the client to the server. In contrast, widely deployed PIN-based TFA schemes that transmit passwords and PINs over a TLS channel are subject to an offline dictionary attack in this case.

Note on Security under S and C Corruptions. If S is corrupted, then the adversary cannot test any C session keys (technically, such sessions are declared "not fresh," see Reference [45]). Indeed, an adversary who learns S's long-term secrets can successfully authenticate as S to C. However, even if S is corrupted and its long-term secrets leak, we can still achieve security for S sessions whose local state is not compromised by the adversary. (This is known as KCI-security of Authenticated Key Exchange, see, e.g., Reference [45].) By contrast, if client C is compromised and its password leaks, then we must also declare all C sessions "not fresh," because in our model the client has no other input than the password, and it has no other means of authenticating *either* server S or device D,[4] as our assumption is that the SAS-channel authenticates C to D, and not

[4]However, see the discussion of the "password-over-TLS" implementation option under the *aPAKE* heading in Section 3.

Input: Sender C holds message $\mathsf{M_C}$; Receiver D holds $\mathsf{M_C}'$.

Output: Receiver D accepts if $\mathsf{M_C} = \mathsf{M_C}'$ and rejects otherwise.

Assumptions: C-to-D SAS channel with capacity t; security parameter κ; hash function H_{com} onto $\{0,1\}^\kappa$.

SAS-MA Protocol:

(1) C sends $\mathsf{Com} = H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$ to D for random R_C, d s.t. $|R_C| = t$ and $|d| = \kappa$;
(2) D sends to C a random string R_D of length t;
(3) C sends (R_C, d) to D and enters $\mathsf{checksum}_C = R_C \oplus R_D$ into C-to-D SAS channel;
(4) D sets $\mathsf{checksum}_D = R_C \oplus R_D$ and it accepts if and only if $\mathsf{Com} = H_{\mathsf{com}}(\mathsf{M_C}', R_C, d)$ *and* $\mathsf{checksum}_C$ received on the SAS channel equals $\mathsf{checksum}_D$.

Fig. 1. SAS Message Authentication (SAS-MA) [72].

vice versa.[5] Consequently, an adversary who learns the password can successfully authenticate to the client. However, our TFA model requires security of S sessions in the C-corruption case, which is the main concern of TFA authentication: If the password leaks, then the adversary must still have at most 2^{-t} probability of authenticating to the server per each attempt, which involves an interaction with *both* server S and device D.

Extension: The Case of simultaneous C and S Corruption. Note that when C and D are corrupted, there is no security to be offered, because the attacker has possession of all authenticator factors, the password and the auxiliary device. However, in the case that both C and S are corrupted one can hope that the attacker could not authenticate to sessions of S that the attacker does not actively control. Indeed, the above model can be extended to include this case with a bound of $\min(q_S, q_D)/2^t$. Our protocols as described in Figures 2 and 4 do not achieve this stronger bound, but it can be achieved by the following small modification (refer to the figures): S is initialized with a public key of D and before sending the value zid to D (via C), S encrypts it under D's public key.

3 BUILDING BLOCKS

We recall several of the building blocks used in our TFA-KE protocol.

SAS-MA Scheme of Vaudenay [72]. The **Short Authentication String Message Authentication (SAS-MA)** scheme allows the transmission of a message from a sender to a receiver so that the receiver can check the integrity of the received message. A SAS-MA scheme considers two communication channels. One that allows the transmission of messages of arbitrary length and is controlled by an active man-in-the-middle, and another that allows sending up to t bits that cannot be changed by the attacker (neither channel is assumed to provide secrecy). We refer to these as the *open channel* and the *SAS channel,* respectively, and call the parameter t the *SAS channel capacity.* A SAS-MA scheme is called *secure* if the probability that the receiver accepts a message modified by a (computationally bounded) attacker on the open channel is no more than 2^{-t} (plus a negligible fraction). In Figure 1, we show a secure SAS-MA implementation of Reference [72] for a sender C and a receiver D. The SAS channel is abstracted as a comparison of two t-bit strings $\mathsf{checksum}_C$ and $\mathsf{checksum}_D$ computed by sender and receiver, respectively. As shown in Reference [72], the probability that an active man-in-the-middle attacker between D and C succeeds in changing message $\mathsf{M_C}$ while D and C compute the same checksum is at most 2^{-t}. Note that this level of security is achieved without any keying material (secret or public) pre-shared between the

[5] A *bi-directional* SAS channel would allow client session security in the case of leaked password, up the 2^{-t} bound, and it would authenticate D to C in the password-over-TLS implementation; see footnote 3. Note that all SAS channel implementations in Section 7.1 extend to bi-directional authentication if checksum validation is done on *both* D and C.

parties. Also, importantly, there is no requirement for checksums to be secret. (In Section 5, we present a formal SAS-MA security definition.)

Thus, the SAS-MA protocol reduces integrity verification of a received message M_C to verifying the equality of two strings (checksums) assumed to be transmitted "out-of-band," i.e., away from adversarial control. In our application, the checksums will be values displayed by device D and client C whose equality the human user verifies and confirms via a physical action, e.g., a click, a QR snapshot, or an audio read-out (see Section 6). In the TFA-KE application this user-confirmation of checksum equality serves as evidence of the physical control of terminal C and device D by the same user, and a confirmation of user's possession of the second authentication factor implemented by D.

SAS-SMT. One can use a SAS-MA mechanism from C to D to bootstrap a *confidential channel* from D to C. The transformation is standard: To send a message m securely from D to C (in our application m is a one-time key and D's PTR response, see below), C picks a CCA-secure public key encryption key pair (sk, pk) (e.g., pair (x, g^x)) for an encryption scheme (KG, Enc, Dec), sends pk to D, and then C and D execute the SAS-MA protocol on M_C = pk. If D accepts, then it sends m encrypted under pk to C, who decrypts it using sk. The security of SAS-MA and the public-key encryption imply that an attacker can intercept m (or modify it to some related message) only by supplying its own key pk′ instead of C's key, and causing D to accept in the SAS-MA authentication of pk′, which by SAS-MA security can happen with probability at most 2^{-t}. The resulting protocol has four messages, and the cost of a plain Diffie-Hellman exchange if implemented using ECIES [23] encryption. We refer to this scheme as SAS-SMT (SMT for "secure message transmission").

aPAKE. Informally, an aPAKE (i.e., an asymmetric or augmented PAKE) is a password protocol that offers limited form of security against server compromise [26, 39]. Namely, the server stores a one-way function of the user's password, and the attacker who breaks into the server can only learn information on the password through an exhaustive offline dictionary attack. While the aPAKE terminology is typically used in the context of password-only protocols that do not rely on public keys, we extend it here (following Reference [45]) to the PKI-based password-over-TLS protocol. This enables the use of our techniques in the context of TLS, a major benefit of our TFA schemes. Note that password-over-TLS, while secure against server compromise, is not strictly an aPAKE as it allows an attacker to learn plaintext passwords (decrypted by TLS) while the attacker is in control of the server. As shown in Reference [45], dealing with this property requires a tweak in the DE-PAKE protocol (C needs to authenticate the value b sent by D in the PTR protocol described below - see also Section 6).

DE-PAKE. A DE-PAKE [45] is an extension of the asymmetric PAKE model by an auxiliary device, which strengthens aPAKE protocols by eliminating offline dictionary attacks upon server compromise. We use DE-PAKE protocols as a main module in our general construction of TFA-KE, and our practical instantiation of this construction, protocol OpTFA, uses the DE-PAKE scheme of Reference [45], which combines an asymmetric aPAKE with a password hardening procedure PTR described next.

Password-to-Random Scheme PTR**.** A PTR is a password hardening procedure that allows client C to translate with the help of device D (which stores a key k) a user's *master password* pwd into independent pseudorandom passwords (denoted rwd) for each user account. The PTR instantiation from Reference [45] is based on the Ford-Kaliski's Blind Hashed Diffie-Hellman technique [38]: Let G be a group of prime order q, let H' and H be hash functions that map onto, respectively, elements of G and κ-bit strings, where κ is a security parameter. Define $F_k(x) = H(x, (H'(x))^k)$, where the key k is chosen at random in $\mathbb{Z}_q$. In PTR this function is computed jointly between C

Components: In addition to the SAS-MA, PTR and aPAKE tools introduced in Sec. 3, OpTFA uses an unauthenticated KE (uKE) protocol, a PRF R, a CCA-secure public key encryption scheme (KG, Enc, Dec), and a MAC function.

Initialization:

(1) On input the user's password pwd, pick random k in $\mathbb{Z}_q$ and set rwd = $F_k(\text{pwd}) = H(\text{pwd}, (H'(\text{pwd}))^k)$;
(2) Initialize the asymmetric PAKE scheme aPAKE on input rwd and let σ denote the user's state at the server.
(3) Choose random key K_z for PRF R, and set zidSet to the empty set;
(4) Give (k, K_z, zidSet) to D and (σ, K_z) to S.

Login step I (C-S uKE + zid **generation):**

(1) S and C run a (unauthenticated) key exchange uKE which establishes session key K_{CS} between them;
(2) S generates random κ-bit nonce zid, computes $z \leftarrow \mathsf{R}(K_z, zid)$, and sends zid to C authenticated under key K_{CS}.

Login step II (C-D **SAS-MA + PTR):**

(1) C generates PKE key pair (sk, pk) $\leftarrow$ KG, t-bit random value R_C, κ-bit random value d, and random r in Z_q. C then computes $a \leftarrow H'(\text{pwd})^r$, $\mathsf{M_C} \leftarrow (\text{pk}, zid, a)$, $\mathsf{Com} \leftarrow H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$, and sends ($\mathsf{M_C}$, Com) to D;
(2) D on ((pk, zid, a), Com), aborts if $zid \in$ zidSet, else adds zid to zidSet and sends random t-bit value R_D to C.
(3) C receives R_D, computes $\text{checksum}_C \leftarrow R_C \oplus R_D$, sends ($R_C$, d) to D, and inputs checksum_C in C-D SAS channel.
(4) D computes $\text{checksum}_D \leftarrow R_C \oplus R_D$ and upon receiving checksum_C on the C-to-D SAS channel, it checks if $\text{checksum}_C = \text{checksum}_D$ and $\mathsf{Com} = H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$ and aborts if not. Otherwise D computes $b \leftarrow a^k$ and $z \leftarrow \mathsf{R}(K_z, zid)$, and sends $e_D \leftarrow \mathsf{Enc}(\text{pk}, (z, b))$ to C.
(5) C computes $(z, b) \leftarrow \mathsf{Dec}(\text{sk}, e_D)$ and rwd $\leftarrow H(\text{pwd}, b^{1/r})$ [$= F_k(\text{pwd})$], and aborts if Dec outputs $\perp$.

Login step III (C-S **aPAKE over Authenticated Link):**

(1) C and S run protocol aPAKE on resp. inputs rwd and σ with all aPAKE messages authenticated by keys z and K_{CS} (each key is used to compute a MAC on each aPAKE message).
Each party aborts and sets local output to $\perp$ if any of the MAC verifications fails.
(2) The final output of C and S equals their outputs in the aPAKE instance: either a session key K or a rejection sign $\perp$.

Fig. 2. OpTFA: Efficient TFA-KE Protocol with Optimal Security Bounds.

and D where D inputs key k and C inputs x = pwd as the argument, and the output, denoted rwd = $F_k(\text{pwd})$, is learned by C only. The protocol is simple: C sends $a = (H'(\text{pwd}))^r$ for r random in $\mathbb{Z}_q$, D responds with $b = a^k$, and C computes rwd = $H(x, b^{1/r})$. Under the **One-More (Gap) Diffie-Hellman (OM-DH)** assumption in the **Random Oracle Model (ROM)**, this scheme realizes a universally composable **oblivious PRF (OPRF)** [44], which in particular implies that x = pwd is hidden from all observers and function $F_k(\cdot)$ remains pseudorandom on all inputs that are not queried to D.

4 OpTFA: A PRACTICAL SECURE TFA-KE PROTOCOL

In Section 5, we present a general design, GenTFA, of a TFA-KE scheme based on two generic components, namely, SAS-MA and DE-PAKE. But first, in this section, we show a practical instantiation of GenTFA, called OpTFA, using the specific building blocks presented in Section 3, namely, the SAS-MA scheme from Figure 1 and the DE-PAKE scheme from Reference [45] (that uses the DH-based PTR scheme described in Section 3 composed with any asymmetric PAKE). This concrete instantiation serves as the basis of our implementation in Section 6 and helps explaining the rationale of our general construction. Protocol OpTFA is presented in Figure 2, and in a schematic form in Figure 3.

Enhanced TFA via SAS. Before going into the specifics of OpTFA, we describe a *general technique* for designing TFA schemes using a SAS channel. In traditional TFA schemes, a PIN is displayed to the user who copies it into a login screen to prove access to that PIN. As discussed in the introduction, this mechanism suffers of significant weaknesses mainly due to the low entropy of PINs (and inconvenience of copying them). We suggest automating the transmission of the PIN

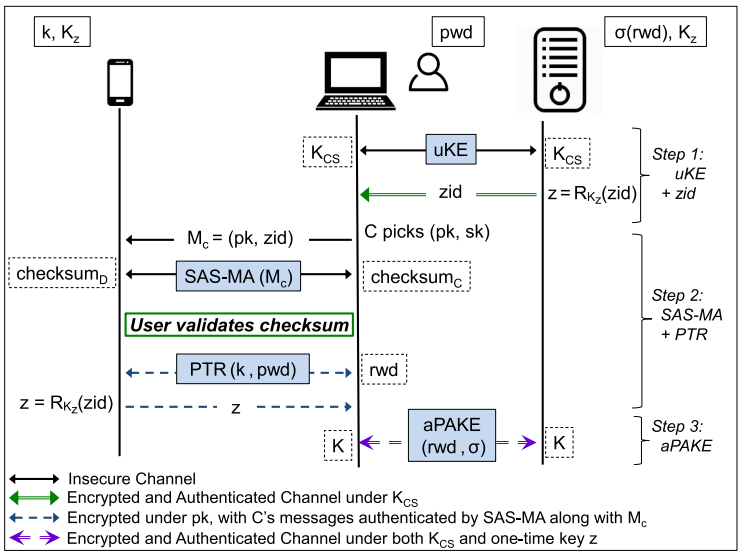


Fig. 3. Schematic Representation of Protocol OpTFA of Figure 2.

over a *confidential channel* from device D to client C. To implement such channel, we use the SAS-SMT scheme from Section 3 where security boils down to having D and C display t-bit strings (checksums) that the user checks for equality. In this way, low-entropy PINs can be replaced with full-entropy values (we refer to them as ***one-time keys (OTK)***) that are immune to eavesdropping and bound active attacks to a success probability of 2^{-t}. These active attacks are impractical even for $t = 20$ (more a denial-of-service than an impersonation threat) and with larger t's they are even more so, as illustrated in Section 6. Note that this approach works with any form of generation of OTK's, e.g., time-based mechanisms, challenge-response between device and server, and so on.

4.1 OpTFA Explained

Protocol OpTFA (Figure 2) requires several mechanisms that are necessary to obtain the strong security bounds of the TFA-KE model. To provide rationale for the need of these mechanisms we show how the protocol is built bottom-up to deliver the required security properties (refer to the introduction for a list of vulnerabilities this design addresses). We stress that while the design is involved the resultant protocol is efficient and practical. The presentation and discussion of security properties here is informal but the intuition can be formalized as we do via the TFA-KE model (Section 2), the generic protocol GenTFA in next section and the proof of Theorem 5.1.

In general terms, OpTFA can be seen as a DE-PAKE protocol using the PTR scheme from Section 3 and enhanced with fresh OTKs transmitted from D to C via the above SAS-SMT mechanism. The OTK is generated by the device and server for each session and then included in the aPAKE interaction between C and S. We note that OpTFA treats aPAKE generically, so any such scheme can be used. In particular, we start by illustrating how OpTFA works with the standard password-over-TLS aPAKE, and then generalize to the use of any aPAKE, including PKI-free ones.

• OpTFA 0.0. This is standard password-over-TLS where the user's password is transmitted from C to S under the protection of TLS.

• OpTFA 0.1. We enhance password-over-TLS with the OTK-over-SAS mechanism described above. First, C transmits the user's password to S over TLS and if the password verifies at S, S sends a

nonce *zid* to C who relays it to D. On the basis of *zid* (which also acts as session identifier in our analysis), D computes an OTK $z = \mathrm{R}_{K_z}(zid)$ where R is a PRF and K_z a key shared between D and S. D transmits z to C over the SAS-SMT channel and C relays it to S over TLS. The user is authenticated only if the received value z is the same as the one computed by S.

This scheme offers defense in case of password leakage. With a full-entropy OTK it ensures security against eavesdroppers on the D-C link and limits the advantage of an active attacker to a probability of 2^{-t} for SAS checksums of length t. However, the scheme is open to online password attacks (as in current commonly deployed schemes), because the attacker can try online guesses without having to deal with the transmission of OTK z. In addition, it offers no security against offline dictionary attacks upon server compromise.

• OpTFA 0.2. We change OpTFA 0.1 so that the user's password pwd is only transmitted to S at the end of the protocol together with the OTK z (it is important that if z does not verify as the correct OTK, that the server does not reveal if pwd is correct or not). This change protects the protocol against online guessing attacks and reduces the probability of the successful testing of a candidate password to $2^{-(d+t)}$ rather than 2^{-d} in version 0.1.

• OpTFA 0.3. We add defense against offline dictionary attacks upon server compromise by resorting to the DE-PAKE construction of Reference [45] and, in particular, to the password-to-random hardening procedure PTR from Section 3. For this, we now assume that the user has a master password pwd that PTR converts into randomized passwords rwd for each user account. By registering rwd with server S and using PTR for the conversion, DE-PAKE security ensures that offline dictionary attacks are infeasible even if the server is compromised (case (3) in Definition 2.1). Note that the PTR procedure runs between D and C following the establishment of the SAS-SMT channel.

• OpTFA 0.4. We change the run of PTR between D and C so that the value a computed by C as part of PTR is transmitted over the SAS-authenticated channel from C to D. Without this authentication the strict bound of case (3) in Definition 2.1 (simplified for $q'_C = 0$), namely, $\mathrm{Adv}^{\mathrm{TFA}}_{\mathrm{A}} \leq q_D/2^{d+t} + \epsilon$ upon server compromise, would not be met. Indeed, when the attacker compromises server S, it learns the key K_z used to compute the OTK z so the defense provided by OTK is lost. So, how can we still ensure the 2^t denominator in the above bound expression? The answer is that by authenticating the PTR value a under SAS-MA, the attacker is forced to run (expected) 2^t sessions to be able to inject its own value a over that channel. Such injection is necessary for testing a password guess even when K_z is known. When considering a password dictionary of size 2^d this ensures the denominator 2^{d+t} in the security bound.

• OpTFA 0.5. We add the following mechanism to OpTFA: Upon initialization of an authentication session (for a given user), C and S run an *unauthenticated* (a.k.a. anonymous) key exchange uKE (e.g., a plain Diffie-Hellman protocol) to establish a shared key K_{CS} that they use as a MAC key applied to all subsequent OpTFA messages. To see the need for uKE assume it is omitted. For simplicity, consider the case where attacker A knows the user's password. In this case, all A needs for impersonating the user is to learn one value of z, which it can attempt by acting as a man-in-the-middle on the C-D channel. After q_D such attempts, A has probability of $q_D/2^t$ to learn z, which together with the user's password allows A to authenticate to S. In contrast, the bound required by Definition 2.1 in this case is the stricter $\min\{q_S, q_D\}/2^t$. This requires that for *each* attempt at learning z in the C-D channel, not only A needs to try to break SAS-MA authentication but it also needs to establish a new session with S. For this, we resort to the uKE channel. It ensures that a response z to a value *zid* sent by S over a uKE session will only be accepted by S if this response comes back on the *same* uKE session (i.e., authenticated with the same keys used by S to send the challenge *zid*). It means that both *zid* and z are exchanged with the same party. If *zid* was

sent to the legitimate user, then the attacker, even if it learns the corresponding z, cannot use it to authenticate back to S. We note that uKE is also needed in the case that the attacker does not know the password. Without it, the success probability for this case is about a factor $2^d/q_S$ higher than acceptable by Definition 2.1.
Note. When all communication between C and S goes over TLS, there is no need to establish a dedicated uKE channel; TLS serves as such.

• OpTFA 0.6. We stipulate that D never responds twice to the same *zid* value (for this, D keeps a stash of recently seen *zid*'s; older values become useless to the attacker once they time out at the server). Without this mechanism the attacker gets multiple attempts at learning z for a single challenge *zid*. However, this would violate bound (1) (for the case $q_C = q'_C = 0$) $\min\{q_S, q_D\}/2^{d+t}$, which requires that each guess attempt at z be bound to the establishment of a new session of the attacker with S.

• OpTFA 0.7. Finally, we generalize OpTFA so that the password protocol run as the last stage of OpTFA (after PTR generates rwd) can be implemented with *any* asymmetric aPAKE protocol, with or without assuming PKI, using the server-specific user's password rwd. As shown in Reference [45], running any aPAKE protocol on a password rwd produced by PTR results in a DE-PAKE scheme, a property that we use in an essential way in our analysis.

We need one last mechanism for C to prove knowledge of z to S, namely, we specify that both C and S use z as a MAC key to authenticate the messages sent by protocol aPAKE (this is in addition to the authentication of these messages with key K_{CS}). Without this, an attack is possible where in case that OpTFA fails the attacker learns if the reason for it was an aPAKE failure or a wrong z. This allows the attacker to mount an online attack on the password without the attacker having to learn the OTK. (When the aPAKE is password-over-TLS the MAC mechanism is not needed, since authentication is achieved by encrypting rwd and z under the same CCA-secure ciphertext [41]).

• OpTFA. Version 0.7 constitutes the full specification of the OpTFA protocol, described in Figure 2, with generic aPAKE.

Performance: The number of exponentiations in OpTFA is reported in the introduction; implementation and performance information is presented in Section 6.

OpTFA *Security.* Security of OpTFA follows from that of protocol GenTFA, because OpTFA is its instantiation. See Theorem 5.1 in Section 5 and Corollary 5.2.

5 THE GENERIC GenTFA PROTOCOL

In Figure 4, we show protocol GenTFA, which is a generalization of protocol OpTFA shown in Figure 2 in Section 4. (Figure 4 shows a simplified protocol that separates the C-D secure channel establishment in step II from DE-PAKE in step III, but see a note below on a round-optimized version of this protocol.) Protocol GenTFA is a compiler that converts *any* secure DE-PAKE and SAS-MA schemes into a secure TFA-KE. It uses the same uKE and CCA-PKE tools as protocol OpTFA, but it also generalizes two other mechanisms used in OpTFA as, respectively, a symmetric-key *Key Encapsulation Mechanism* (KEM) scheme and an ***Authenticated Channel* (AC)** scheme.

A (symmetric-key) Key Encapsulation Mechanism (KemE, KemD) (see, e.g., Reference [69]), allows for encrypting a random session key given a (long-term) symmetric key K_z, i.e., if $(zid, z) \leftarrow \mathsf{KemE}(K_z)$, then $z \leftarrow \mathsf{KemD}(K_z, zid)$. An adversarial distinguishing advantage $\epsilon^{\mathsf{KEM}}(n)$ against n instances of KEM is defined as the distinguishing advantage between pairs $(zid_1, z_1), \ldots, (zid_n, z_n)$ output by n runs of $\mathsf{KemE}(K_z)$ and values $(zid_1, z_1^*) \ldots, (zid_n, z_n^*)$, where $z_1*, \ldots, z_n^*$ are chosen as n independent random κ-bit strings. In protocol OpTFA of Figure 2, KEM is implemented using PRF R: *zid* is a random κ-bit string and $z = R(K_z, zid)$, in which case $\epsilon^{\mathsf{KEM}}(n) \leq q^2/(2^\kappa) + \epsilon^{\mathsf{PRF}}(n)$ where

Initialization: Given the user's password pwd, we initialize the DE-PAKE scheme on pwd. Let k and σ be the resulting user-specific states stored at resp. D and S. Let K_z be a random KEM key. Let zidSet be an empty set. D is initialized with (k, K_z, zidSet) and S is initialized with (σ, K_z).

Login step I (C-S **KE + KEM generation):**

(1) S and C create shared key K_{CS} using a (non-authenticated) key exchange uKE.
(2) S generates $(zid, z) \leftarrow \mathsf{KemE}(K_z)$, sets $e_S \leftarrow \mathsf{ACSend}(K_{CS}, zid)$, and sends e_S to C, who computes $zid \leftarrow \mathsf{ACRec}(K_{CS}, e_S)$, or aborts if decryption fails.

Login step II (C-D SAS-MA **+ KEM decryption):**

(1) C generates a PKE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}$, sends $\mathsf{M_C} = (\mathsf{pk}, zid)$ to D, and C and D run SAS-MA to authenticate $\mathsf{M_C}$ using the t-bit C-to-D SAS channel.
(2) D aborts if $zid \in$ zidSet or if the SAS scheme fails. Otherwise, D adds zid to zidSet, computes $z \leftarrow \mathsf{KemD}(K_z, zid)$, picks a random MAC key K_{CD}, computes $e_D \leftarrow \mathsf{Enc}(\mathsf{pk}, (z, K_{CD}))$ and sends e_D to C.
(3) C computes $(z, K_{CD}) \leftarrow \mathsf{Dec}(\mathsf{sk}, e_D)$ (aborts if $\perp$).

Login step III (DE-PAKE over Authenticated Links):

C, D, and S run DE-PAKE on resp. inputs pwd, k, and σ, modified as follows:

(a) All communication between D and S is routed through C.

(b) Communication between C and D goes over a channel authenticated by key K_{CD}, i.e. it is sent via $\mathsf{ACSend}(K_{CD}, \cdot)$ and received via $\mathsf{ACRec}(K_{CD}, \cdot)$, Either party aborts if its ACRec ever outputs $\perp$.

(c) Communication between C and S goes over a channel authenticated by key z and then the result of that is sent over a channel authenticated by key K_{CS}, i.e. it is sent via $\mathsf{ACSend}(K_{CS}, \mathsf{ACSend}(z, \cdot))$ and received via $\mathsf{ACRec}(K_{CS}, \mathsf{ACRec}(z, \cdot))$. Each party aborts and sets local output to $\perp$ if its ACRec instance ever outputs $\perp$.

The final outputs of C and S are their respective outputs in this DE-PAKE instance, either session key K or a rejection $\perp$.

Fig. 4. Generic TFA-KE Scheme: Protocol GenTFA.

$\epsilon^{\mathsf{PRF}}(n)$ is the bound on the distinguishing advantage against PRF R where n is the number of PRF quaries. We also generalize the usage of the MAC function in OpTFA as an Authenticated Channel, defined by a pair ACSend, ACRec, which implements bi-directional authenticated communication between two parties sharing a symmetric key K [32, 43]. Algorithm ACSend takes inputs key K and message m and outputs m with authentication tag computed with key K, while the receiver procedure, $\mathsf{ACRec}(K, \cdot)$, outputs either a message or the rejection symbol $\perp$. We assume that the AC scheme is stateful and provides authenticity and protection against replay.

Round optimization. Protocol GenTFA in Figure 4 is simplified by separating the C-D secure channel set-up in step II from DE-PAKE in step III. This is not round-optimal if the first step of the DE-PAKE scheme also consists of a round of C-D interaction, as is the case for, e.g., the DE-PAKE scheme of Reference [45], which we use to instantiate protocol GenTFA in Section 4. Indeed, such round of DE-PAKE communication could be piggy-backed onto the C-D communication in step II as follows: C can generate its first DE-PAKE message a on its input password pwd, and run step (1) as in Figure 4 but for $\mathsf{M_C} = (\mathsf{pk}, zid, a)$. Then device D runs step (2) as in Figure 4, but it forms the ciphertext it sends to C as $e_D \leftarrow \mathsf{Enc}(\mathsf{pk}, (z, K_{CD}, b))$, where b is its DE-PAKE response computed on C's message a and D's local input k. Finally, in step (3) C parses the decryption of e_D as $(z, K_{CD}, b) \leftarrow \mathsf{Dec}(\mathsf{sk}, e_D)$ and runs *the rest* of the DE-PAKE execution as in step III in Figure 4 from this point on. Protocol OpTFA of Section 4 is an instantiation of this round-optimized version of GenTFA.

The security of GenTFA is stated in the following theorem:

Theorem 5.1. *Assuming security of the building blocks DE-PAKE, SAS, uKE, PKE, KEM, and AC, protocol* GenTFA *is a* (T, ϵ)*-secure TFA-KE scheme for* ϵ *upper bounded by*

$$\epsilon^{\mathsf{DEPAKE}} + n \cdot (\epsilon^{\mathsf{SAS}} + \epsilon^{\mathsf{uKE}} + \epsilon^{\mathsf{PKE}} + 6\epsilon^{\mathsf{AC}}) + 2\epsilon^{\mathsf{KEM}}(n),$$

for $n = q_{HbC} + \max(q_S, q_D, q_C, q'_C)$, *where* q_{HbC} *denotes the number of* GenTFA *protocol sessions in which the adversary is only eavesdropping, and each quantity of the form* ϵ^{P} *is a bound on the advantage of an attacker that works in time* $\approx T$ *against a single instance of protocol building block* P*, or against n instances in case of* $\epsilon^{\mathsf{KEM}}(n)$.

Theorem 5.1 applies to both the GenTFA protocol as shown in Figure 4 and to its round-optimized version. Thus, as a corollary, we obtain a proof of TFA-KE security for protocol OpTFA from Figure 2, which uses specific secure instantiations of GenTFA components. The corollary follows by applying the result of Vaudenay [72] on the security of the SAS-MA scheme used in OpTFA, assuming ROM, and the result of Reference [45] on the security of DE-PAKE used in OpTFA, assuming OM-DH assumption and that a secure asymmetric PAKE scheme. The factors in front of each expression of the form ϵ^{P} in Theorem 5.1 are upper-bounded by $n = q_{HbC} + \max(q_S, q_D, q_C, q'_C)$, and the exact quantities can be found in the corresponding step in the security proof.

Corollary 5.2. *Assuming that* aPAKE *is a secure asymmetric PAKE,* uKE *is secure Key Exchange,* (KG, Enc, Dec) *is a CCA-secure PKE,* R *is a secure PRF, and MAC is a secure message authentication code,* OpTFA *is a secure TFA-KE scheme under the OM-DH assumption in ROM.*

Security definition of SAS authentication. For the purpose of the proof below, we state the security property assumed of a SAS-MA scheme, which was informally described in Section 3. While Reference [72] defines the security of SAS-MA using a game-based formulation, here we do it via the following (universally composable) functionality $\mathsf{F}_{\mathsf{SAS[t]}}$: On input a message [SAS.SEND, *sid*, P', m] from an honest party P, functionality $\mathsf{F}_{\mathsf{SAS[t]}}$ sends [SAS.SEND, *sid*, P, P', m] to A, and then, if A's response is [SAS.CONNECT, *sid*], then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends [SAS.SEND, *sid*, P, m] to P', if A's response is [SAS.ABORT, *sid*], then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends [SAS.SEND, *sid*, P, $\perp$] to P', and if A's response is [SAS.ATTACK, *sid*, m'] then $\mathsf{F}_{\mathsf{SAS[t]}}$ throws a coin ρ, which comes out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$, and if $\rho = 1$ then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends succ to A and [SAS.SEND, *sid*, P, m'] to P', and if $\rho = 0$ then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends fail to A and [SAS.SEND, *sid*, P, $\perp$] to P'.

In our main instantiation of the generic protocol GenTFA of Figure 4, i.e., in protocol OpTFA of Figure 2, we instantiate SAS-MA with the scheme of Reference [72], but even though the original security argument given for it in Reference [72] used the game-based security notion, it is straightforward to adopt this argument to see that this scheme securely realizes the above (universally composable) functionality.

Proof of Theorem 5.1. We consider first protocol GenTFA as shown in Figure 4, and we explain separately below how this proof extends to the round-optimized version. Let A be an adversary limited by time T playing the TFA-KE security game, which we will denote G_0, instantiated with the TFA-KE scheme GenTFA. Let the security advantage defined in Definition 2.1 for adversary A satisfy $\mathsf{Adv}^{\mathsf{TFA}}_{\mathsf{A}} = \epsilon$. Let Π^{S}_i, Π^{C}_j, Π^{D}_l refer to, respectively, the ith, jth, and lth instances of S, C, and D entities, which A starts up. Let t be the SAS channel capacity, κ the security parameter, q_S, q_D, q_C, q'_C the limits on the numbers of rogue sessions of S, D, C when communicating with S, and C when communicating with D, and let q_{HbC} be the number of GenTFA protocol sessions in which A plays only a passive eavesdropper role except that we allow A to abort any of these protocol executions at any step. Let $n_S = q_S + q_{HbC}$, $n_D = q_D + q_{HbC}$, $n_C = \max(q_C, q'_C) + q_{HbC}$, and note that these are the ranges of indexes i, j, l for instances Π^{S}_i, Π^{C}_j, and Π^{D}_l. We will use $[n]$ to denote range $\{1, \ldots, n\}$.

The security proof goes by cases depending on the type of corrupt queries A makes. In all cases the proof starts from the security-experiment game G_0 and proceeds via a series of game changes, G_1, G_2, etc, until a modified game G_i allows us to reduce an attack on the DE-PAKE with the same corruption pattern (except in the case of corrupt client C) to the attack on G_i. In the case of

the corrupt client the argument is different, because it does not rely on the underlying DE-PAKE (note that DE-PAKE does not provide any security properties in the case of client corruption). In some game changes, we will consider a modified adversary algorithm, for example an algorithm constructed from the original adversary A interacting with a simulator of some higher-level procedure, e.g., the SAS-MA simulator. Wlog, we use A_i for an adversary algorithm in game G_i.

We will use p_i to denote the probability that A_i interacting with game G_i outputs b' s.t. $b' = b$ where b is the bit chosen by the game on the test session. Recall that when A makes the test session query test(P, i), for $P \in \{\mathsf{S}, \mathsf{C}\}$, then, assuming that instance Π_i^P produced a session key sk, game G_0 outputs that session key if $b = 1$ or produces a random string of equal size if $b = 0$ (and if session Π_i^P did not produce the key then G_0 outputs $\perp$ regardless of bit b). Note that by assumption $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} = \epsilon$, we have that $p_0 = 1/2 + 1/2 \cdot \mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} = 1/2 + \epsilon/2$.

Case 1: No party is compromised. This is the case when A makes no corrupt queries, i.e., it is the default "network adversary" case.[6]

Game G_1*:* Let (zid_i, z_i) be the KEM (ciphertext,key) pair generated in Step I.1 by Π_i^{S}. Let Z be a random function that maps onto κ-bit strings. Let $\mathsf{E}_{\mathsf{Zcol}}$ be the event that any two S sessions pick the same zid field, i.e., that for any i_1, i_2 in $[n_S]$, we have $i_1 \neq i_2$ and $zid_{i_1} = zid_{i_2}$. Let $\mathsf{A}_1 = \mathsf{A}_0$ and let game G_1 be like G_0 except that (1) it aborts if $\mathsf{E}_{\mathsf{Zcol}}$ happens and (2) it sets each z_i as $z_i \leftarrow Z(zid_i)$. We have that $p_1 \leq p_0 + 2\epsilon^{\mathsf{KEM}}(n_S)$, because the difference between G_0 and G_1 can be upper-bounded by the distringuishing advantage between n_S KEM instances using, respectively, real and random keys, which is $\epsilon^{\mathsf{KEM}}(n_S)$, plus the probability of zid-collision. However, the last probability can also be upper-bounded by $\epsilon^{\mathsf{KEM}}(n_S)$, because a zid-collision immediately implies an attack on KEM, since in the real execution zid-collision implies a repeat of the key z, while in the random-key KEM game each z is independently random.

Game G_2*:* Let $\mathsf{SIM}_{\mathsf{SAS}}$ be the simulator for the SAS-MA scheme. Let $\mathsf{A}_2 = \mathsf{A}_1$, and let G_2 be like G_1 except that in Step II.1 when instance Π_j^{C} of C and instance Π_l^{D} of D execute the SAS-MA sub-protocol, we replace this SAS-MA execution with a simulator $\mathsf{SIM}_{\mathsf{SAS}}$ interacting with A_1 and the ideal SAS-MA functionality $\mathsf{F}_{\mathsf{SAS[t]}}$. Namely, instance Π_j^{C}, instead of sending $\mathsf{M}_{\mathsf{C}} = (\mathsf{pk}, zid)$ to A_1 and starting a SAS-MA instance to authenticate M_{C} to D, will issue command $[\mathsf{SAS.SEND}, sid, \Pi_l^{\mathsf{D}}, \mathsf{M}_{\mathsf{C}}]$ to $\mathsf{F}_{\mathsf{SAS[t]}}$, which triggers $\mathsf{SIM}_{\mathsf{SAS}}$ to start simulating to A_1 the SAS-MA protocol between Π_j^{C} and Π_l^{D} on message M_{C} as an input. Depending on the way A_1 responds, $\mathsf{SIM}_{\mathsf{SAS}}$ can act in one of the following three ways: (1) If $\mathsf{SIM}_{\mathsf{SAS}}$ sends $[\mathsf{SAS.CONNECT}, sid]$ to $\mathsf{F}_{\mathsf{SAS[t]}}$, then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \mathsf{M}_{\mathsf{C}}]$ to Π_l^{D} and Π_l^{D} proceeds to step II.2 using this received message; (2) If $\mathsf{SIM}_{\mathsf{SAS}}$ sends $[\mathsf{SAS.ABORT}, sid]$ to $\mathsf{F}_{\mathsf{SAS[t]}}$, then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends $\perp$ to Π_l^{D} and Π_l^{D} aborts; (3) If $\mathsf{SIM}_{\mathsf{SAS}}$ sends $[\mathsf{SAS.ATTACK}, sid, \mathsf{M}_{\mathsf{C}}{}^*]$ to $\mathsf{SIM}_{\mathsf{SAS}}$ for some $\mathsf{M}_{\mathsf{C}}{}^*$ (w.l.o.g. $\mathsf{M}_{\mathsf{C}}{}^* \neq \mathsf{M}_{\mathsf{C}}$), then $\mathsf{F}_{\mathsf{SAS[t]}}$ throws a coin ρ_l, which comes out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$, and if $\rho = 0$ then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends fail to $\mathsf{SIM}_{\mathsf{SAS}}$ and $\perp$ to Π_l^{D} and Π_l^{D} aborts, and if $\rho = 1$ then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends succ to A and $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \mathsf{M}_{\mathsf{C}}{}^*]$ to Π_l^{D}, and then Π_l^{D} proceeds to step II.2 using message $\mathsf{M}_{\mathsf{C}}{}^*$. Since the SAS-MA protocol realizes the UC functionality $\mathsf{F}_{\mathsf{SAS[t]}}$ with at most error ϵ^{SAS} (per instance), and the simulator $\mathsf{SIM}_{\mathsf{SAS}}$ executes independently from the rest of the security game G_2, it follows that $p_2 \leq p_1 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{SAS}}$.

Game G_3*:* Note that in the above security game adversary A_2 interacts with game G_2, which internally runs interactive algorithms $\mathsf{SIM}_{\mathsf{SAS}}$ and $\mathsf{F}_{\mathsf{SAS[t]}}$. Note also that the $\mathsf{SIM}_{\mathsf{SAS}}$ algorithm interacts only with $\mathsf{F}_{\mathsf{SAS[t]}}$ on one end and A_2 on the other. We can, therefore, draw the boundaries between

[6]We refer to the eprint version of this article [48] for an overview of the game changes in the proof below, where each game is described more intuitively and in less technical terms.

the adversarial algorithm A and the security game G slightly differently: Consider an adversarial algorithm A_3, which executes the steps of A_2 and SIM_{SAS}, and a security game G_3, which executes the rest of game G_2, including the operation of functionality $F_{SAS[t]}$. Note that G_3 does not execute the SAS-MA protocol, but interacts with A_3 using the $F_{SAS[t]}$ interface to SIM_{SAS}, i.e., G_3 sends to A_3 messages of the type $[\text{SAS.SEND}, sid, \Pi_j^C, \Pi_l^D, M_C]$, and A_3's response must be one of $[\text{SAS.CONNECT}, sid]$, $[\text{SAS.ABORT}, sid]$, and $[\text{SAS.ATTACK}, sid, M_C{}^*]$. Since we are only redrawing the boundaries between the adversarial algorithm and the security game, we have that $p_3 = p_2$.

Game G_4*:* Let $A_4 = A_3$ and let G_4 be as G_3 except that if G_3 sends $[\text{SAS.SEND}, sid, \Pi_j^C, \Pi_l^D, M_C]$ for some (j, l) pair, and A_4 sends $[\text{SAS.CONNECT}, sid]$ in response, then we make the following changes: First, e_D sent by Π_l^D is formed as $\text{Enc}(\text{pk}, (0^\kappa, 0^\kappa))$ instead of $\text{Enc}(\text{pk}, (z, K_{CD}))$ as in G_3, for pk specified in $M_C = (\text{pk}, zid)$. Second, if A_3 passes this e_D to Π_j^C, then Π_j^C decrypts it as the (z, K_{CD}) pair, which was generated by Π_l^D. Otherwise, the game does not change, and in particular if A_3 passes some other ciphertext $e_D^* \neq e_D$ to Π_j^C then Π_j^C decrypts e_D^* in a standard way. By the reduction to CCA security of PKE (KG, Enc, Dec), it follows that $p_4 \leq p_3 + \min(n_C, n_D) \cdot \epsilon^{\text{PKE}}$.

Game G_5*:* Let $E_{\text{ACbreak(CD)}}$ be an event that there is some session pair (Π_j^C, Π_l^D) s.t. (a) A_4 responded with $[\text{SAS.CONNECT}, sid]$ to $[\text{SAS.SEND}, sid, \Pi_j^C, \Pi_l^D, M_C]$, and (b) A_4 delivered e_D sent by Π_l^D to Π_j^C, and (c) in the DE-PAKE interaction between Π_j^C and Π_l^D authenticated by key K_{CD} in step III either party accepts a message either not sent by the counterparty or delivered out of order. Let $A_5 = A_4$ and G_5 be as G_4 except that G_5 aborts if $E_{\text{ACbreak(CD)}}$ ever happens. Since in game G_4, under conditions (a) and (b), the adversary has no information about key K_{CD} used by both Π_j^C and Π_l^D, by the security of the authentic channel implementation, we have that condition (c) can hold with probability at most $\min(n_C, n_D) \cdot \epsilon^{\text{AC}}$, hence $p_5 \leq p_4 + \min(n_C, n_D) \cdot \epsilon^{\text{AC}}$.

Game G_6*:* Let $E_{\text{ACbreak(CD}')}$ be an event that there is some session pair (Π_j^C, Π_l^D) s.t. (a) A_4 responded with $[\text{SAS.CONNECT}, sid]$ to $[\text{SAS.SEND}, sid, \Pi_j^C, \Pi_l^D, M_C]$, (b) A_4 did not deliver e_D sent by Π_l^D to Π_j^C, and (c) instance Π_l^D did not abort in step III. Let $A_6 = A_5$ and G_6 be as G_5 except that G_6 aborts if $E_{\text{ACbreak(CD}')}$ ever happens. Since in game G_5, under conditions (a) and (b), only Π_l^D has information on key K_{CD}, by the security of the authenticated channel implementation we have that condition (c) can hold with probability at most $q_D \cdot \epsilon^{\text{AC}}$, hence $p_6 \leq p_5 + q_D \cdot \epsilon^{\text{AC}}$.

Game G_7*:* Let $A_7 = A_6$ and G_7 be as G_6 except that for every uKE instance in step I.1 between Π_i^S and Π_j^C, if the adversary is an eavesdropper on it then G_7 replaces key K_{CS} output by Π_i^S and Π_j^C with a random key. By uKE security it follows that $p_7 \leq p_6 + \min(n_C, n_S) \cdot \epsilon^{\text{uKE}}$.

Game G_8*:* Let $E_{\text{ACbreak(CS)}}$ be an event that there is some session pair Π_i^S, Π_j^C s.t. (a) the adversary is passive on the KE executed in step I.1 and (b) in the DE-PAKE interaction between Π_j^C and Π_i^S authenticated by key K_{CS} in step III either party accepts a message either not sent by the counterparty or delivered out of order. Let $A_8 = A_7$ and G_8 be as G_7 except that G_8 aborts if $E_{\text{ACbreak(CS)}}$ ever happens. Since in game G_7 the adversary has no information about K_{CS}, by the security of the authenticated channel implementation, we have that $p_8 \leq p_7 + \max(n_C, n_S) \cdot \epsilon^{\text{AC}}$.

Note that at this point the game has the following properties: If A is passive on the C-S key exchange in step I, then A is forced, by game G_8, to be passive on the C-S link in the DE-PAKE in step III. Also, if A does not attack the SAS-MA sub-protocol and delivers D's ciphertext to C in step II then A is forced, by game G_5, to be passive on the C-D link in the DE-PAKE in step III (and if A does not deliver D's ciphertext to C, then this D instance will not respond to any further messages, by game G_6). The remaining cases are thus active attacks on the key exchange in step

I and when A either attacks the SAS-MA sub-protocol and gets D to accept $\mathsf{M_C}* \neq \mathsf{M_C}$ or sends $e_D^* \neq e_D$ to C.

We will handle these cases next, and the crucial issue will be what the adversary does with the *zid* values created by S. Consider any S instance Π_i^{S} in which the adversary interferes with the key exchange protocol in step I.1. Without loss of generality assume that the adversary learns key K_{CS} output by Π_i^{S} in this step. Note that D keeps a variable zidSet in which it stores all *zid* values it ever receives, and that D aborts if it sees any *zid* more than once. Therefore, each game execution defines a 1-1 function $L : [n_S] \rightarrow [n_D] \cup \{\perp\}$ s.t. if $L(i) \neq \perp$, then $L(i)$ is the unique index in $[n_D]$ s.t. $\Pi_{L(i)}^{\mathsf{D}}$ receives $\mathsf{M_C} = (\mathsf{pk}, zid_i)$ in step II.1 for some pk, and $L(i) = \perp$ if and only if no D session receives zid_i. If $L(i) \neq \perp$, then consider two cases: First, if $\mathsf{M_C} = (\mathsf{pk}, zid_i)$, which contains zid_i, originates with some session Π_j^{C}, and second if $\mathsf{M_C} = (\mathsf{pk}, zid_i)$ is created by the adversary.

Game G_9*:* Consider first the case of a rogue session Π_i^{S} and a rogue session Π_j^{C} to which the adversary sends zid_i in step I.2. Consider first the case when the adversary stops Π_j^{C} from getting the corresponding z_i. Namely, let $\mathsf{E}_{\mathsf{zidOmit(i)}}$ be an event s.t. the adversary (a) either never issues $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^*]$ for $\mathsf{M_C}^*$ containing zid_i or it does but the corresponding coin toss comes out $\rho = 0$, (b) does not send zid_i to any C instance, or it does send it to Π_j^{C} for some $j \in [n_C]$, but either responds with $[\mathsf{SAS.ABORT}, sid]$ to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ in step II.1 or responds with $[\mathsf{SAS.CONNECT}, sid]$ but does not deliver e_D sent by Π_l^{D} to Π_j^{C} in step II.2. Note that by conditions (a) and (b), and the fact that already in game G_4 ciphertext e_D created in response to $[\mathsf{SAS.CONNECT}, sid]$ does not contain any information about $z_i = Z(zid_i)$, neither session Π_j^{C} nor the adversary have any information about z_i. Therefore, by the security of the authenticated channel implementation Π_i^{S} should reject. Consider $\mathsf{A}_9 = \mathsf{A}_8$ and G_9 like G_8 except G_9 sets Π_i^{S}'s output to $\perp$ at the end of step III if $\mathsf{E}_{\mathsf{zidOmit(i)}}$ happens. By the argument above we have that $p_9 \leq p_8 + q_S \cdot \epsilon^{\mathsf{AC}}$.

Game G_{10}*:* Consider the same case of a rogue session Π_i^{S} and a rogue session Π_j^{C} to which the adversary sends zid_i in step I.2, but now consider the possibility that the adversary lets Π_j^{C} get the corresponding z_i but does not learn z_i itself. Namely, let $\mathsf{E}_{\mathsf{zidPass(i,j)}}$ be an event for some $i \in [n_S]$ and $j \in [n_C]$, (a) Π_j^{C} receives zid_i in step I.2, (b) the adversary responds with $[\mathsf{SAS.CONNECT}, sid]$ to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ in step II.1, (c) the adversary never issues $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^*]$ for $\mathsf{M_C}^*$ containing zid_i, and (d) the adversary delivers e_D sent by Π_l^{D} to Π_j^{C} in step II.2. Consider $\mathsf{A}_{10} = \mathsf{A}_9$ and G_{10} like G_9 except that if $\mathsf{E}_{\mathsf{zidPass(i,j)}}$ happens and in the DE-PAKE interaction between Π_j^{C} and Π_i^{S} (where both parties use z_i to authenticate this interaction), if the adversary does *not* deliver to either Π_i^{S} or Π_j^{C} the messages of the counterparty in the correct order, G_{10} makes this party abort and sets its output to $\perp$. (Note that this means that the other party will also abort, unless the misdelivered message was the last message this party sent.) Note that by conditions (a) and (b) instance Π_l^{D} receives zid_i in $\mathsf{M_C}$ sent by Π_j^{C}. By condition (c) this is the first time D receives zid_i, hence it will not abort, and by condition (d) Π_j^{C} will receive z_i corresponding to zid_i. Since the adversary has no information about z_i, by the security of the authenticated channel implementation it follows that Π_j^{C} and Π_i^{S} output $K \neq \perp$ only (except for the probability of an attack on the authenticated channel) if the adversary passes the DE-PAKE messages m' (authenticated by z) between these two rogue instances as a man-in-the-middle. It follows that $p_{10} \leq p_9 + \min(q_C, q_S) \cdot \epsilon^{\mathsf{AC}}$.

Note that by the changes done by games G_9 and G_{10}, if the adversary interferes with the KE in step I.1 with session Π_i^{S}, sends zid_i to some Π_j^{C} and does not send it to some Π_l^{D} in a $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid_i)]$ message for any l then the adversary is forced to be a passive eavesdropper on the DE-PAKE protocol in step III, or otherwise Π_i^{S} will output $\perp$. Note that this is the

case when $L(i) = l$ s.t. the game issues [SAS.SEND, $sid, \Pi^{\mathsf{C}}_j, \Pi^{\mathsf{D}}_l$, (pk, zid_i)] for some pk, i.e., if some Π^{D}_l receives zid_i, then it receives it as part of a message $\mathsf{M_C}$ originated by some client session Π^{C}_j.

Game G_{11}*:* Consider now the case when the adversary sends zid_i to D by itself, i.e., when $L(i) = l$ s.t. the adversary does sends [SAS.ATTACK, sid, $\mathsf{M_C}^* = (\mathsf{pk}^*, zid_i)$] for some pk^* in response to [SAS.SEND, $sid, \Pi^{\mathsf{C}}_j, \Pi^{\mathsf{D}}_l, \mathsf{M_C}$] for some j and $\mathsf{M_C}$. Let $\mathsf{E}_{\mathsf{zFail(i,l)}}$ be an event that (a) the above conditions hold, (b) that the adversary does not send zid_i to any client instance in step I.2, and (c) that $\rho_l = 0$, i.e., that Π^{D}_l rejects $\mathsf{M_C}^*$ and aborts. Consider $\mathsf{A}_{11} = \mathsf{A}_{10}$ and G_{11} just like G_{10} except that G_{10} makes Π^{S}_i abort in step III and sets its output to $\perp$ in case of event $\mathsf{E}_{\mathsf{zFail(i,l)}}$ for any $l \in [n_D]$. Note that by condition (a) and (b) session $l = L(i)$ of D is the only one that gets zid_i, hence if $\rho_l = 0$ then the adversary has no information about $z_i = Z(zid_i)$, hence by the security of the authenticated channel it follows that $p_{11} \leq p_{10} + q_S \cdot \epsilon^{\mathsf{AC}}$.

After these game changes, we are finally ready to make a reduction from an attack on underlying DE-PAKE to an attack on the TFA-KE. Specifically, we will construct an algorithm A^*, which runs in time comparable to A, achieves advantage $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*} = 2 \cdot (p_{11} - 1/2)$ against the underlying DE-PAKE scheme, and makes q^*_S, q^*_D, q_C, q_C rogue queries, respectively, to S, D, to C on its connection to S, and to C on its connection with D, where $q^*_S = q^*_D = q^*$, where q^* is a random variable equal to the sum of $q = \min(q_S, q_D)$ coin tosses that come out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$. Recall that $\mathsf{Adv}^{\mathsf{TFA}}_{\mathsf{A}} = 2 \cdot (p_0 - 1/2)$ and that by the game changes above, we have that $|p_{11} - p_0|$ is a negligible quantity, and hence $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*}$ is negligibly close to $\mathsf{Adv}^{\mathsf{TFA}}_{\mathsf{A}}$.

Reducing DE-PAKE attack to TFA-KE attack. The reduction works by A^* internally running algorithm A and emulating entities S, C, and D to A as in game G_{11}. If A starts up an instance Π^{S}_i, Π^{C}_j, and Π^{D}_l, then A^* starts up its local state for these sessions, which we will denote $\bar{\Pi}^{\mathsf{S}}_i$, $\bar{\Pi}^{\mathsf{C}}_j$, and $\bar{\Pi}^{\mathsf{D}}_l$.

Emulation of Step I of GenTFA *to* A*:* When A^* starts up $\bar{\Pi}^{\mathsf{S}}_i$ or $\bar{\Pi}^{\mathsf{C}}_j$, it runs the KE on their behalf in step I.1. Let $K^{\mathsf{S}}_{CS,i}$, $K^{\mathsf{C}}_{CS,j}$ be the keys these instances output from the KE step. If A connects $\bar{\Pi}^{\mathsf{S}}_i$ and $\bar{\Pi}^{\mathsf{C}}_j$ in HbC fashion, then we call this pair *HbC-paired*, and A^* sets $K^{\mathsf{S}}_{CS,i} = K^{\mathsf{C}}_{CS,j}$ to a random key, as in G_{11} (see G_7). In Step I.2 for $\bar{\Pi}^{\mathsf{S}}_i$, A^* picks zid_i and sets $z_i = Z(zid_i)$ as in G_{11} (see G_1), and sends $\mathsf{ACSend}(K^{\mathsf{S}}_{CS,i}, 1, zid_i)$. Denote this (zid_i, z_i) pair as $(zid^{\mathsf{S}}_i, z^{\mathsf{S}}_i)$. When $\bar{\Pi}^{\mathsf{C}}_j$ receives a message in step I.2, it decodes it as zid^{C}_j using $\mathsf{ACRec}(K^{\mathsf{C}}_{CS,i}, 1, \cdot)$. If ACRec fails, then $\bar{\Pi}^{\mathsf{C}}_j$ aborts. If $\bar{\Pi}^{\mathsf{S}}_i$ and $\bar{\Pi}^{\mathsf{C}}_j$ are not HbC-paired but $zid^{\mathsf{C}}_j = zid^{\mathsf{S}}_i$, then we call these instances *zid-paired.*

Emulation of Step II of GenTFA *to* A*:* A^* picks (sk, pk) as C in step II.1 and sends [SAS.SEND, $sid, \Pi^{\mathsf{C}}_j, \Pi^{\mathsf{D}}_l, \mathsf{M_C}$] to A for $\mathsf{M_C} = (\mathsf{pk}, zid)$ and $zid = zid^{\mathsf{C}}_j$, where l is an index in $[n_D]$ set by A. If A responds with [SAS.CONNECT, sid] and zid was not sent to D before (otherwise $\bar{\Pi}^{\mathsf{D}}_l$ aborts), then A^* generates e_D as an encryption of two constants, as in G_{11}. If A forwards this e_D to $\bar{\Pi}^{\mathsf{C}}_j$, then A^* sets $z^{\mathsf{C}}_j = Z(zid^{\mathsf{C}}_j)$, picks a random key $K^{\mathsf{C}}_{CD,j}$, sets $K^{\mathsf{D}}_{CD,l} = K^{\mathsf{C}}_{CD,j}$, and denotes such $\bar{\Pi}^{\mathsf{C}}_j, \bar{\Pi}^{\mathsf{D}}_l$ instances as *paired.* However, if A responds with [SAS.ATTACK, sid, $\mathsf{M_C}^*$] for $\mathsf{M_C}* = (\mathsf{pk}^*, zid^*)$ s.t. zid^* was not sent to D before (otherwise $\bar{\Pi}^{\mathsf{D}}_l$ aborts), A^* picks coin ρ_l as in G_{11} (see G_2) and aborts $\bar{\Pi}^{\mathsf{D}}_l$ unless $\rho_l = 1$ (which happens with probability 2^{-t}). If $\bar{\Pi}^{\mathsf{D}}_l$ does not abort, then A^* picks a random key $K^{\mathsf{D}}_{CD,l}$ and sends out $e_D = \mathsf{Enc}(\mathsf{pk}^*, (Z(zid^*), K^{\mathsf{D}}_{CD,l}))$. If A did not respond with [SAS.CONNECT, sid] or $\bar{\Pi}^{\mathsf{C}}_j$ receives e^*_D, which differs from e_D sent by $\bar{\Pi}^{\mathsf{D}}_l$, then A^* sets $(z^{\mathsf{C}}_j, K^{\mathsf{C}}_{CD,j}) \leftarrow \mathsf{Dec}(\mathsf{sk}, e^*_D)$.

As in G_{11}, A^* can abort some sessions at this point: (1) A^* aborts $\bar{\Pi}^{\mathsf{D}}_l$ if A responds with [SAS.CONNECT, sid] above but does not forward e_D to $\bar{\Pi}^{\mathsf{C}}_j$ (see G_6); (2) A^* aborts $\bar{\Pi}^{\mathsf{S}}_i$ and sets its output to $\perp$ if the conditions of event $\mathsf{E}_{\mathsf{zidOmit(i)}}$ are satisfied (see G_9), i.e., (a) A was not HbC

in the key exchange with $\bar{\Pi}^{\mathsf{S}}_i$ in step I, (b) A either does not send [SAS.ATTACK, *sid*, ·] with zid^{S}_i or it does but the corresponding coin-toss ρ comes out 0, (c) A does not sent zid^{S}_i to any $\bar{\Pi}^{\mathsf{C}}_j$ session, or it does for some j but then either does not do [SAS.CONNECT, *sid*] or does not deliver the resulting e_D to $\bar{\Pi}^{\mathsf{C}}_j$; (3) A* aborts $\bar{\Pi}^{\mathsf{S}}_i$ and sets its output to $\perp$ if the conditions of event $\mathsf{E}_{\mathsf{zFail(i,l)}}$ are satisfied for some $l \in [n_D]$ (see G_{11}), i.e., A does not send zid^{S}_i to any $\bar{\Pi}^{\mathsf{C}}_j$ instance, sends [SAS.ATTACK, *sid*, $(\mathsf{pk}^*, zid^{\mathsf{S}}_i)$] to some $\bar{\Pi}^{\mathsf{D}}_l$ but coin ρ_l comes out 0.

Emulation of Step III of GenTFA *to* A*:* Finally, A* emulates step III of TFA-KE by using the state held by $\bar{\Pi}^P_i$ for any $P \in \{\mathsf{S}, \mathsf{C}, \mathsf{D}\}$ and i s.t. $\bar{\Pi}^P_i$ reached step III of GenTFA without aborting. A* performs this emulation by implementing the Authenticated Channel layer as in step III of GenTFA using the corresponding state computed above, i.e., $K^{\mathsf{S}}_{CS,i}, z^{\mathsf{S}}_i$ for $\bar{\Pi}^{\mathsf{S}}_i$, $K^{\mathsf{C}}_{CS,j}, z^{\mathsf{C}}_j, K^{\mathsf{C}}_{CD,j}$ for $\bar{\Pi}^{\mathsf{C}}_j$, and $K^{\mathsf{D}}_{CD,l}$ for $\bar{\Pi}^{\mathsf{D}}_l$, and implementing the DE-PAKE messages by initiating and communicating with the external DE-PAKE parties, respectively, $\Pi^{\mathsf{S}}_i, \Pi^{\mathsf{C}}_j$, and Π^{D}_l. However, if at any point the authenticated channel receiver $\mathsf{ACRec}(\cdot,\cdot,\cdot)$ outputs $\perp$ for any $\bar{\Pi}^P_i$, then A* aborts this $\bar{\Pi}^P_i$ and never communicates with Π^P_i again. Moreover, A* aborts whenever (1) event $\mathsf{E}_{\mathsf{ACbreak(CD)}}$ ever happens for paired sessions $\bar{\Pi}^{\mathsf{C}}_j, \bar{\Pi}^{\mathsf{D}}_l$ (see G_5), (2) event $\mathsf{E}_{\mathsf{ACbreak(CS)}}$ ever happens for HbC-paired sessions $\bar{\Pi}^{\mathsf{C}}_j, \bar{\Pi}^{\mathsf{S}}_i$ (see G_8), (3) if $\bar{\Pi}^{\mathsf{S}}_i$ and $\bar{\Pi}^{\mathsf{C}}_j$ are zid-paired and $\bar{\Pi}^{\mathsf{C}}_j$ and $\bar{\Pi}^{\mathsf{D}}_l$ are paired (i.e., if event $\mathsf{E}_{\mathsf{zidPass(i,j)}}$ occurs), but $\bar{\Pi}^{\mathsf{S}}_i$ or $\bar{\Pi}^{\mathsf{C}}_j$ accept any message except that sent by the counterparty in the corrent order (see G_{10}).

By the above rules the only Π^{S}_i instances on which A* can be rogue are s.t. A was not passive in the key exchange with $\bar{\Pi}^{\mathsf{S}}_i$ in step I, and there is a *unique* $l \in [n_S]$ s.t. A sent [SAS.ATTACK, *sid*, $(\mathsf{pk}^*, zid^{\mathsf{S}}_i)$] in response to [SAS.SEND, *sid*, $\Pi^{\mathsf{C}}_j, \Pi^{\mathsf{D}}_l, \cdot$], and $\bar{\Pi}^{\mathsf{D}}_l$ did not abort, which in particular implies that coin ρ_l came out 1. Note also that the only Π^{D}_l instances on which A* can be rogue are s.t. A sent [SAS.ATTACK, *sid*, (pk^*, zid^*)] in response to [SAS.SEND, *sid*, $\Pi^{\mathsf{C}}_j, \Pi^{\mathsf{D}}_l, \cdot$], and $\bar{\Pi}^{\mathsf{D}}_l$ did not abort, implying again $\rho_l = 1$. Therefore, each rogue session Π^{S}_i corresponds to a unique rogue session Π^{D}_l, hence w.l.o.g. we can assume a 1-1 relation between rogue Π^{S}_i sessions and rogue Π^{D}_l sessions. Since for each such pair of sessions A* aborts them unless ρ_l comes out 1, which happens with probability 2^{-t}, we have that the number of both S and D rogue sessions A* makes is bounded by $q^*_S = q^*_D = q^*$ where q^* is a random variable equal to the sum of $q = \min(q_S, q_D)$ coin tosses that come out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$. Since the interaction of A* with the DE-PAKE scheme emulates the security experiment G_{11} to A exactly, it follows that A* advantage in this DE-PAKE attack is $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*} = 2 \cdot (p_{11} - 1/2)$, and hence $\mathsf{Adv}^{\mathsf{TFA}}_{\mathsf{A}} \le \mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}*} + 2(p_{11} - p_0)$.

Finally, we consider an attacker A*, which makes $(q^*_S, q^*_D, q_C, q'_C)$ rogue queries of respective type, where $q^*_S = q^*_D = q^*$ is a random variable as above to the overall advantage of A*. We will treat q_C, q'_C, q_D, q_S as constants, we will set $q = \min(q_S, q_D)$, and we will treat q^* as a random variable. Note that for every $(q_C, q'_C, q^*_S, q^*_D)$ where $q^*_S = q^*_D = q^*$, the assumption of DE-PAKE security implies that $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*}$ is bounded by a linear expression of the type $a \cdot q_C + b \cdot q'_C + c \cdot q^*$. Since q^* is a random variable whose expectation is $q/2^{-t}$ when we measure $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*}$ over all the randomness in the reduction and the DE-PAKE game, which includes the randomness in q^* (i.e., the coins ρ_l for $l \in [n_D]$), the overall contribution of term $c \cdot q^*$ will be $\sum_{i=0}^{q} \Pr[q^* = i] * (c \cdot q^*) = c \cdot \mathsf{Exp}(q^*) = c \cdot q/2^t$.

Hence, over all the randomness of A, A*, and the DE-PAKE security game, $\mathsf{Adv}^{\mathsf{DEPAKE}}_{\mathsf{A}^*}$ is bounded by $a \cdot q_C + b \cdot q'_C + c \cdot \min(q_S, q_D)/2^t$. Consequently, if the DE-PAKE is $(T', \epsilon^{\mathsf{DEPAKE}})$-secure for $T' \approx T$ (namely, T plus the emulation work of A*, which takes $O(1)$ cryptographic ops per each party instance), then the TFA-KE scheme GenTFA is (T, ϵ)-secure for $\epsilon \le \epsilon^{\mathsf{DEPAKE}} + (p_{11} - p_0) \le n \cdot$

$(\epsilon^{\mathsf{KEM}} + \epsilon^{\mathsf{SAS}} + \epsilon^{\mathsf{PKE}} + \epsilon^{\mathsf{uKE}} + 6\epsilon^{\mathsf{AC}}) + n^2/2^{\kappa}$ where $n = q_{HbC} + \max(q_S, q_D, q_C, q'_C)$, which implies the theorem statement for the case where no party is corrupted.

Extension to the round-optimized version. Recall that if the DE-PAKE protocol starts by a round of C-D communication then the round-optimized version of GenTFA amends the protocol by forming the SAS-authenticated C-to-D message as $\mathsf{M_C} = (\mathsf{pk}, zid, a)$ where a is C's first DE-PAKE message, and forming the D-to-C's response as $e_D \leftarrow \mathsf{Enc}(\mathsf{pk}, (z, K_{CD}, b))$ where b is D's DE-PAKE response to a. The security proof extends to this version, because SAS-MA authentication of $\mathsf{M_C}$ and CCA-security of PKE bind DE-PAKE messages a, b to this session in the same as the $\mathsf{ACSend}(K_{CD}, \cdot)$ mechanism binds the DE-PAKE to this session in the non-optimized protocol. Specifically, by G_6 applied to the round-optimized protocl, we have the following cases: (1) If A let message $\mathsf{M_C}$ pass from C to D and message e_D pass from D to C, then the C-D DE-PAKE exchange a+b was delivered honestly *and* A is likewise reduced to only passive attack on the rest of C-to-D DE-PAKE communication; (2) If A attacks this SAS session and succeeds, then it gets access to a rogue D instance of DE-PAKE, just like in the non-optimized protocol; (3) If A sends its own ciphertext $e^*_D \neq e_D$ to C, then it gets access to a rogue C instance of DE-PAKE, again just like above.

Case 2: Party Corruptions. Due to lack of space, we defer to the eprint version of the article [48] for the security proofs for the cases of client, device, and server corruption, showing that our scheme achieves all the bounds of Definition 2.1. Here, we briefly comment on how these bounds are derived. For the D-corruption case, the value z is learned by the attacker hence it is equivalent to setting $t = 0$. Also, rogue queries to D are free for the attacker, hence, q_D is virtually unbounded (can think of it as "infinity"). Setting these values in the bound of Case 1, one obtains the claimed bound $(q_C + q_S)/2^d$ for the D-corruption case. Similarly, in the S-corruption case one sets q_S to "infinity." In addition, and in spite of the attacker learning z in this case, one obtains a bound involving 2^{-t} thanks to the fact that we run the DE-PAKE over the SAS channel, thus reducing the probability of an adversarial password test by 2^{-t}. In the C-corruption case, where the attacker learns the user's password pwd, we can set $d = 0$ (i.e., consider a dictionary of size 1) and $q_C = q'_C = 0$, because interaction with C can be emulated given the leaked password. Finally, when both D and S are corrupted one gets the same security as plain DE-PAKE, namely, requiring a full offline dictionary attack to recover pwd.

6 SYSTEM DEVELOPMENT AND PERFORMANCE EVALUATION

Here, we report on an experimental prototype of protocol OpTFA from Figure 2 on page 10 and present novel designs for the SAS channel implementation. We experiment with OpTFA using two different instantiations of the password protocol between C and S. One is PKI-based that runs OpTFA over a server-authenticated TLS connection; in particular, it uses this connection in lieu of the uKE in step I and implements step III by simply transmitting the concatenation of password rwd and the value z under the TLS authenticated encryption. The second protocol we experimented with is a PKI-free asymmetric PAKE borrowed from References [29, 44, 47]. Roughly, it runs the same PTR protocol as described in Section 3 but this time between C and S. C's input is rwd and the result $F_k(\mathsf{rwd})$ serves as a user's private key for the execution of an authenticated key-exchange between C and S. We implement the latter with HMQV [55] (as an optimization, the DH exchange used to implement uKE in step I of OpTFA is "reused" in HMQV).

In Table 1, we provide execution times for the various protocol components, including times for the TLS-based protocol and the PKI-free one with some elements borrowed from the implementation work from Reference [45]. As mentioned in Section 1, the cost of OpTFA is two communication

Table 1. Average Execution Time of OpTFA and Its Components (10,000 Iterations)

Protocol	Purpose	Parties	Average Time in ms (std. dev.)
SAS (excluding user's checksum validation)	Authenticate C-D Channel	C and D	128.59 (0.48)
PTR	Reconstruct rwd	C and D	160.46 (3.71)
PKI-free PAKE	PAKE	C and S	182.27 (3.67)
PKI PAKE (TLS)	C-S link encryption	C and S	32.54 (1.38)
Overall in PKI-free Model		C, D, and S	**410.77 ms**
Overall in PKI Model		C, D, and S	**263.27 ms**

rounds between D and C, with 4 and 3 exponentiations by C and D, respectively, and a one-round Diffie-Hellman exchange between C and S.

We build on the following platform. The webserver S is a Virtual Machine running Debian 8.0 with 2 Intel Xeon 3.20 GHz and 3.87 GB of memory. Client terminal C is a MacBook Air with 1.3 GHz Intel Core i5 and 4 GB of memory. Device D is a Samsung Galaxy S5 smartphone running Android 6.0.1. C and D are connected to the same WiFi network with the speed of 100 Mbps and S has Internet connection speed of 1Gbps. The server side code is implemented in HTML5, PHP, and JavaScipt. On the client terminal, the protocol is implemented in JavaScript as an extension for the Chrome browser and the smartphone app in Java for Android phones.

All DH-based operations (PTR, key exchange and SAS-SMT encryption) use elliptic curve NIST P-256, and hashing and PRF use HMAC-SHA256. Hashing into the curve is implemented with simple iterated hashing till an abscissa x on the curve is found (it will be replaced with a secure mechanism such as Reference [27]).

Communication between C and S uses a regular internet connection between the browser C and web server S. Communication between C and D (except for checksum comparison) goes over the internet using a bidirectional **Google Cloud Messaging (GCM)** [10], in which D acts as the GCM server and C acts as the GCM client. GCM involves a registration phase during which GCM client (here C) registers with the GCM generated client ID to the GCM server (here D), to assure that D only responds to the registered clients. In case that the PAKE protocol in OpTFA is implemented with password-over-TLS, Reference [45] specifies the need for D to authenticate the PTR value b sent to C (see Section 3). In this case, during the GCM registration, we install at C a signature public key of D.

7 CHECKSUM VALIDATION DESIGN AND USABILITY STUDY

7.1 Checksum Validation Design

An essential component in our approach and solutions (in particular in protocol OpTFA) is the use of a SAS channel implemented via the user-assisted equality verification of checksums displayed by both C and D (denoted hereafter as checksum_C and checksum_D, respectively). Here, we discuss different implementations of such user-assisted verification, which we have designed and experimented with.

Manual Checksum Validation. In the simplest approach, the human user compares the checksums displayed on D and C and taps the Confirm button on D in case the two match [71]. Although, this type of code comparison has recently been deployed in TFA systems, e.g., Reference [16], it carries the danger of neglectful users pressing the confirm button without comparing the checksum strings. Another common solution for checksum validation is "Copy-Confirm" [71] where

the user types the checksum displayed on C into D, and only if this matches D's checksum does D proceeds with the protocol. We refer to this method as Num-C-D. We implemented this scheme using a six-digit number. We stress that in spite of the similarity between this mechanism and PIN copying in traditional TFA schemes, there is an essential security difference: Stealing the PIN in traditional schemes suffices to authenticate instead of the user (for an attacker that holds the user's password) while stealing the checksum value entered by the user in OpTFA is worthless to the attacker (the checksum is a validation code, not the OTK value needed for authentication).

The above methods using human visual examination and/or copying limit the SAS channel capacity (typically to four to six digits) and may degrade usability [65]. As an alternative, we consider the following designs (however, one may fallback to the manual schemes when the more secure schemes below cannot be used, e.g., missing camera or noisy environments).

QR Code Checksum Validation. In this checksum validation model, which we refer to as QR-C-D, we encode the full, 256-bit checksum computed in protocol OpTFA into a hexstring and show it as a 230×230 pixel QR Code on the web-page. We used ZXing library to encode the QR code and display it on the web page and read and decode it on D. To send the checksum to D, the user opens the app on D and captures the QR code. D decodes the QR code and compares checksums, and proceeds with the protocol if the match happens. With the larger checksum ($t = 256$) active attacks on SAS-SMT turn infeasible and the expressions 2^{-t} in Definition 2.1) negligible.

Voice-based Checksum Validation. We implement a voice-based checksum validation approach that assumes a microphone-equipped device (typically a smartphone) where the user speaks a numerical checksum displayed by the client into the device. We refer to this method as Voice-C-D. The device D receives this audio, recognizes and transcribes it using a speech recognition tool, and then compares the result with the checksum computed by D itself. The client side uses a Chrome extension as in the manual checksum validation case, while on the device we developed a transcriber application using Android.Speech API. The user clicks on a "Speak" button added to the app and speaks out loud the displayed number (six-digit in our implementation). The transcriber application in D recognizes the speech and convert it to text that is then compared to D's checksum. To further improve the usability of this approach one can incorporate a text-to-speech tool that would speak the checksum automatically (i.e., replacing the user). The transcription approach would perhaps be easy for the users to employ compared to the QR-based approach, but would only be suitable if the user is in an environment that is non-noisy and allows her to speak out-loud. We note that the QR-code and audio-based approaches do not require a browser plugin or add-on and can be deployed on any browser with HTML5 support.

The three concrete checksum validation user interaction methods we implemented and tested for usability are described in Section 7.2.2.

7.2 Usability Study Implementation and Preliminaries

7.2.1 The Study Setup. **Overview:** To evaluate the usability of different OpTFA checksum comparison methods and to compare them with PIN-TFA as the baseline, we built a study platform. In this setup, we designed a webpage to show the instructions, receive the PIN (related to PIN-TFA), and show the checksums (related to OpTFA). We also developed an Android application to display the PIN (for the PIN-TFA approach) and to receive the checksum (for OpTFA checksum comparison). This setup mimics a TFA login experience where the user should input a correct PIN/checksum to login to an account. Since the password entry procedure could be the same for both PIN-TFA and OpTFA schemes, we skip the password entry and proceed with PIN/checksum entry. That is, we assume that the user has already entered the username and password and is navigated to the TFA page to prove the possession of the secondary device. The participants could

open the study webpage from a client and perform the tasks as instructed on the webpage. In our implementation, the webserver is a virtual server running Apache HTTP Server. Client is a desktop with 2.38 GHz Intel 2 Core Duo and 8 GB of memory. Device is a Samsung Galaxy S5 running Android 6.0.1. The server-side code is implemented in HTML5, PHP, and JavaScript. The smartphone app is developed in Java for Android.

One-Time PIN Generation: We mimic the generation of the PIN/checksum on the server and the device using a random generation function. In the PIN-TFA approach, the PIN is generated by the study app on the smartphone using the Random() function in Java and is displayed as a six-digit number to the participants. In OpTFA, the checksum is generated using rand() function in PHP and is displayed to the participants as a six-digit number for Num-C-D, and Voice-C-D, and is encoded into QR code for QR-C-D.

Storing Participant Responses: To store the responses provided by the participants (the entered PIN/checksum), we use a MySQL database. In case of Num-C-D and PIN-D-C, the responses are stored as entered by each participant. In case of QR-C-D the checksum is recorded as captured and decoded by the QR code decoder, and for Voice-C-D, the checksum is stored as transcribed by the transcriber. We also keep the displayed PIN/checksum in the same database for further offline comparison of the displayed and the entered value. The time it takes to complete each task, and participants' responses and ratings are also stored in the same database.

Off-line Processing: To verify the correctness of the PIN/checksum entered by the participants, we process the data stored on the database offline and report on any error committed by the participants in entering the PIN (PIN-D-C), entering the checksum (Num-C-D), encoding/decoding the checksum (QR-C-D), and speaking or transcription of the checksum (Voice-C-D). While the number of failed attempts would have remained the same whether the processing was to be done in real-time or offline, a real-time analysis could have given feedback to the users and requested them to make another attempt, which might impact the usability score. However, this impact would probably have been the same on all methods equally.

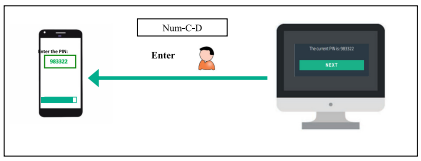
7.2.2 Implementation of User Interaction Methods. We implemented the following user interaction methods tested via our study:

Num-C-D: In this manual checksum approach of OpTFA, the checksum is displayed as a six-digit number on the webpage on C. We ask the participants to enter the checksum into the smartphone app. This method is shown in Figure 5(a).

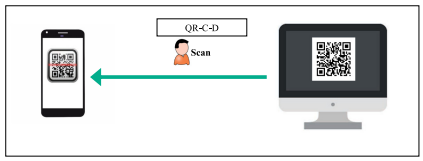
QR-C-D: In this OpTFA method, we encode the six-digit checksum as a 300×300 pixel QR code on the webpage using Google Chart API. To send the checksum to D, each participant opens the app on D and captures the QR code. We used ZXing library [22] to decode the captured checksum on the app. In this setting, the participant does not need to enter the checksum but only needs to hold her/his phone and scan the QR code displayed on the browser's screen as shown in Figure 5(b).

Voice-C-D: In the Voice-C-D approach of OpTFA, similar to Num-C-D, we display the checksum on C. However, rather than entering the checksum on D or capturing the QR code, we ask the participants to speak the checksum into her/his smartphone as shown in Figure 5(c). The smartphone receives this audio, recognizes and transcribes it using a speech recognition tool based on IBM Research Speech-to-Text API in our current implementation. The participant clicks on a "Record" button we embedded in the app and speaks the six-digit number. The transcriber application recognizes the speech and converts it to numbers that can be compared against the locally computed checksum.

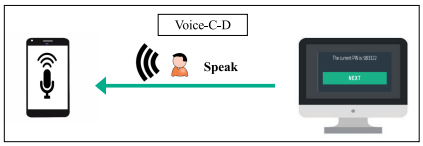
PIN-D-C: In the PIN-D-C approach (PIN-TFA), we map the PIN into a six-digit number. We ask the participants to press the generate button to display the six-digit number in textview box on the study app and to enter it on the webpage, as it is presented in Figure 6.



(a) User interaction in Num-C-D



(b) User interaction in QR-C-D



(c) User interaction in Voice-C-D

Fig. 5. OpTFA user interaction methods.

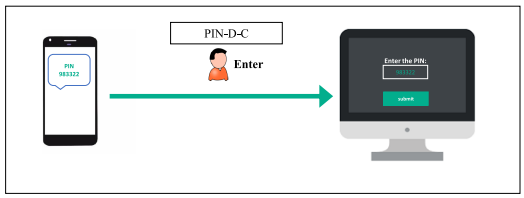


Fig. 6. Traditional PIN-D-C two-factor authentication (PIN-TFA).

7.3 Study Design

7.3.1 Study Objectives and Metrics. To analyze the effectiveness of the OpTFA approach from the point of view of usability and adoption potential, we conducted a formal lab-based study to quantify the following metrics:

(1) **Delay:** *How long does it take for the participants to perform each user interaction method?* The starting point is the time the PIN/checksum was generated and the ending point is the time each PIN/checksum was received at the other end. OpTFA is reported to have a negligible delay [46], and therefore we only time the user interaction.
(2) **Error rate:** *How often do the participants, transcriber, and QR encoder/decoder produce an error in transferring the checksum?* We recorded all PIN/checksum values the participants had entered and the one displayed to them and compared them with each other to determine the number/fraction of errors committed in each method.
(3) **Usability:** *How easy or difficult the participants find the system? Can they easily learn how to use the system? Do they need the support of a technical person?* To capture these aspects and to quantify the usability of the tested methods, we used the standard System Usability Scale (SUS)[7]. We also consider users' perception of *Adoptability*, *Trust*, *Security*, and *Efficiency* of the system.

[7]SUS is a conventional method to measure the usability of systems on 0–100 scale [31]. SUS has been designed to measure the usability of a system with respect to learnability, need for support, participants experience, and satisfaction.

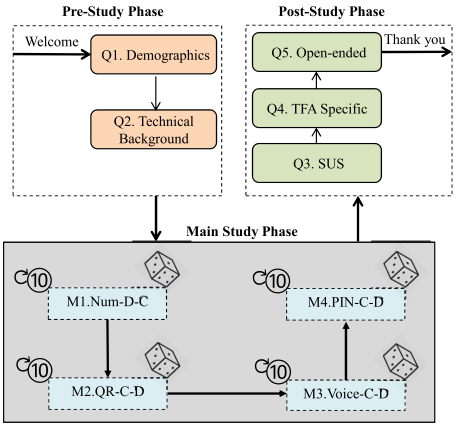


Fig. 7. Study protocol.

7.3.2 Study Protocol. We recruited 30 participants from diverse educational backgrounds from our university's campus (students and non-students), by word of mouth. After a brief introduction about TFA and our study, the participants were navigated by an examiner to a desk and were provided an Android phone that had the study app installed and a desktop that had the study webpage opened. The examiner supervised and observed the participants throughout the study. Upon completion of each task, the participants filled out a survey form. To assure that participants received equal guidance, all information and instructions were shown on each page. We only aim to compare the usability of the user interaction model in the TFA process, and therefore, the installation and setup was not evaluated in our study. Also, since we compare different OpTFA methods with PIN-TFA methods, and we do not solely evaluate the usability of OpTFA, we do not require to define a primary task for the users (e.g., checking emails). Hence, performing same set of tasks in multiple trials would be a sufficient and valid usability design to compare OpTFA with the traditional approach. The study took about 20 min for each participant to complete. The study was approved by our university's IRB. Participation in the study was voluntary, and standard ethical procedures were fully followed (e.g., participants being informed, given choice to discontinue, and not deceived).

The study was composed of three phases as shown in Figure 7: the pre-study, the main study, and the post-study phase. Analyzing the participants' answers, error rates, and behavior in the study helped us to: (1) reason about the usability of each method (or its lack thereof), (2) compare the usability of different methods, and (3) investigate possible security issues arising from the usability problems.

Pre-Study Phase: The quantitative/qualitative pre-study questions were grouped into two categories:

- *Q1. Demographics*: The participants were asked to fill out a demographic questionnaire. These questions polled for each participant's age, gender and education.
- *Q2. Technical Background*: The participants were asked about their general computer and security skills, and about their familiarity with the subject of the study (two-factor authentication).

Main Study Phase: The main study phase aims to evaluate the average error rate and the delay related to each of the tested methods. As discussed in Section 7.2, below is the list of the four user interaction methods that participants were asked to perform. We randomized the ordering of these

four methods to remove any learning biases. We asked the participants to perform the tasks related to each method ten times. Since inputting the username and password is similar regardless of the two-factor authentication scheme, we did not ask users to perform it to keep the study short and concise.

- *M1. Num-C-D*: In this method, we asked the participants to get the checksum number from the webpage and enter it into the app.
- *M2. QR-C-D*: In this method, the participants were asked to capture the QR code from the webpage using the phone.
- *M3. Voice-C-D*: In this method, the participants were asked to get the checksum number from the webpage and to speak it to the phone app.
- *M4. PIN-D-C*: As the baseline for our study, we asked the participants to enter the PIN number from the phone to the webpage.

The task related to each method was shown on a webpage followed by the post-study questions. After completion of each task and answering the post-study questions related to that specific method, the participants were instructed to test the next user interaction method.

Post-Study Phase: The post-study phase consists of the following set of questionnaires to evaluate and compare the usability of the four tested methods.

- *Q3. System Usability Scale*: In the first set of post-study questionnaires, the participants were asked to fill out the SUS questionnaire for each of the four user interaction methods.
- *Q4. TFA-Specific Questions*: In the second questionnaire, we asked more specific questions about each of the user interaction methods to figure out how participants felt about the security and usability of each TFA interaction method. This questionnaire addressed users' perception of: Adoptability, Trust, Security, and Efficiency.
- *Q5. Open-Ended Question*: The study concluded with one open-ended question about the system (i.e., we asked if the participants had additional comments and if they preferred any method).

7.4 Results and Analysis

7.4.1 Pre-Study Analysis. The pre-study demographics questionnaire shows that the 30 participants were from the age group of 18–24 years (30%), 25–34 (60%), and 35–44 (10%) with an equal number of undergraduate and graduate students from diverse educational backgrounds, including education, engineering, healthcare, and science. Only one of the participants was specialized in computer security. Twenty-three percent of the participants were female and 77% were male. Seventy-seven percent of the participants speak English as a second language, and 23% speak English as their mother language. They ranked their general computer background as Poor (4%), Average (73%), and Excellent (23%), and their general computer security skills as Poor (3%), Average (83%), and Excellent (14%). Therefore, we believe that our sample is representative of diverse participants.

7.4.2 Main Study Analysis.

Delay: We estimated the time it takes the participants to transfer the PIN/checksum in each of the user interaction methods. As mentioned in Section 7.3, in computing the delay, we considered the starting point to be the time the PIN/checksum was generated and the ending point to be the time the PIN/checksum was received by the client/device. Figure 8 shows the average delay of each method.

Table 2. The Average Error Rate for Each Method

TFA Method	Average Error Rate
Num-C-D	4%
QR-C-D	2%
Voice-C-D	5.3%
PIN-D-C	5%

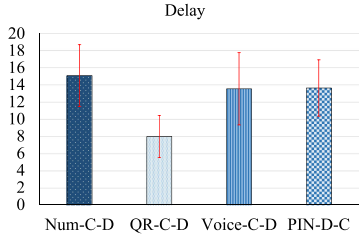


Fig. 8. Mean (std. dev) of delay in seconds.

Num-C-D had the highest delay compared to other methods, with the average and standard deviation of 15.03s (3.62s). This result was expected, as in this method, the user enters the checksum manually on the phone using the small phone keypad. In contrast, QR-C-D imposes the least amount of delay compared to the other methods, with an average of 7.96 s (2.45 s). The average delay for Voice-C-D was 13.53 s (4.23 s), and for PIN-D-C was 13.62 s (3.29 s).

The Friedman test was conducted to compare the delay among different user interaction methods in PIN-D-C, Num-C-D, Voice-C-D, and QR-C-D and rendered a Chi-square value of 49.375, which showed a statistically significant difference with a p-value of 0.00. All results of statistical significance are reported at a 95% confidence level (alpha level of 0.05). Further, Wilcoxon signed-rank test,[8] corrected using Bonferroni correction with an adjusted alpha level of 0.0125 per test (0.05/4) showed a statistically significant difference for the following pairs[9]: (QR-C-D, PIN-D-C), (QR-C-D, Voice-C-D), and (QR-C-D, Num-C-D), each with a p-value = 0.00. This confirms that QR-C-D outperforms all the other tested methods in terms of the delay incurred in the TFA process.

Error Rates: The error rates for all tested methods are presented in Table 2. The table shows the error rate for the Num-C-D method to be 4%, arising from the incorrect entry of the PIN numbers. The lowest error rate was reported to be 2% for QR-C-D. In this method, the user captures the QR code, while the phone makes the comparison by decoding the QR code. Since the QR decoder is almost error-free, we observe that the cause of the errors was the failure of the users in capturing the QR code, i.e., in some instances the participants failed to scan the QR Code and moved forward to the next task. As expected, it seems users have higher error rate in manual checksum entry Num-C-D compared to QR-C-D.

In the Voice-C-D method, we found the error rate to be 5.3%, which is higher than the other methods. To compute the error rate, we compare the transcribed audio checksum with the checksum generated and displayed to the participants. We accepted the transcription errors for zero

[8]This is a non-parametric statistical hypothesis test used to compare two related samples. This test results in a statistically significant outcome if the p value for comparison is less than 0.05 for a confidence level of 95%.
[9]In each reported (x, y) pair, the value of y is statistically significantly greater than x.

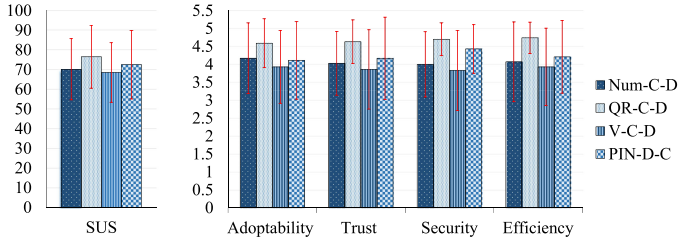


Fig. 9. Mean (std. dev) of user perception ratings.

being transcribed as "Oh," two being transcribed as "to," and four being transcribed as "for." To understand the root cause of the errors we manually reviewed several of the audio samples and noticed that the transcriber made errors in transcribing the spoken checksum in the presence of background noise. Moreover, the majority of the participants were not native English speakers, which may have increased the transcription errors, since the transcriber we used was designed for native English speakers. Since we used an off-the-shelf transcriber, we could not set the grammar to only generate digits. Access to the transcription grammar might improve the accuracy of Voice-C-D method.

As the baseline, PIN-D-C resulted in a 5% error rate, arising from the incorrect input of the PIN on the client. It seems that users make slightly higher errors compared to Num-C-D and QR-C-D, however, this traditional TFA method shows a better result compared to Voice-C-D.

We conducted a Friedman test to compare the error rate among multiple methods, which showed a statistically significant difference and rendered a Chi-square value of 7.847 with a p-value of 0.049. However, Wilcoxon signed-rank test, conducted using Bonferroni adjusted alpha levels of 0.0125 per test (0.05/4), did not show statistical significance for any of the pairs. It seems that most methods have similar error rates, statistically speaking.

7.4.3 Post-Study Analysis. In the post-study questionnaire, users were asked to rate their agreement level with several statements about the usability of each method. (5—strongly agree, 4—agree, 3—neither agree nor disagree, 2—disagree, 1—strongly disagree). The results are shown in Figure 9.

SUS Scores: For QR-C-D, we received the highest SUS score of 76.56 (15.12) compared to other methods. For Voice-C-D, we had the lowest usability with the SUS score of 68.53 (17.45). The average SUS for PIN-D-C reported by our study participants was 72.5 (15.96), and for Num-C-D the SUS score was 70.17 (15.56). Except for Voice-C-D, other methods seem to offer a SUS of higher than 70, which is generally representative of a system with good usability. We conducted the Friedman test to compare the SUS scores. The test did not show statistically significant difference.

User Perception Ratings: We analyze the different facets of the user perception ratings below.

- **Adoptability:** Most of the users found the QR-C-D method to be adoptable in practice as reflected in the adoptability score of 4.59. The Num-C-D method had the second rank in adoptability with the average score of 4.17, almost similar to PIN-D-C score with the average of 4.11. Compared to the other methods, users seem to find the Voice-C-D method to be less adoptable reflected in the average adoptability score of 3.93.
- **Trust:** We found the QR-C-D method to have the highest average trust score of 4.67. PIN-D-C has the second place with an average score of 4.43. The trust score for Num-C-D method was also high with the average of 4.03. Voice-C-D has the lowest trust score with the average score of 3.86. Evidently, the users are more in agreement than disagreement that they can trust the methods.

- **Security:** The motivation of this question is to evaluate the users' perception of security (and not to evaluate the theoretical and practical security of each method). Therefore, in this part of the post-study questionnaire, we asked the users how they felt about each method from the security point of view. We found that most of the participants ranked QR-C-D to be more secure compared to the other methods with the average score of 4.70. For PIN-D-C the score was 4.43 and for Num-C-D this score was 4.00. As was the case for other usability scores, compared to the other methods, the Voice-C-D method had a lower average security score of 3.83.
- **Efficiency:** At the end of our post-study questionnaire, we asked the users how efficient/fast they felt each method was. Most of the participants ranked QR-C-D to be the fastest schema with an average score of 4.74. The score for PIN-D-C was 4.21 standing at the second place and the score for Num-C-D was 4.07. The average score for Voice-C-D was 3.93 (the least among all). Note that this result is the perception of the users of the efficiency of the system. The actual delay is as reported in Section 7.4.2.

The Friedman test was conducted to compare the user's perception of adoptability, trust, security, and efficiency of the methods, among different user interaction methods and rendered Chi-square values of 8.968, 24.377, 18.485, and 11.349, respectively, which showed a statistically significant difference with p-values of 0.03, 0.00, 0.00, and 0.01, for adoptability, trust, security, and efficiency, respectively. This shows that users had a significantly different perception of the adoptability, trust, security, and efficiency of the methods.

Further, Wilcoxon signed-rank test, conducted using the Bonferroni adjusted alpha level of 0.0125 per test (0.05/4), showed statistical significance for adoptability between (Voice-C-D, QR-C-D) pair with a p-value of 0.009. For the level of trust, (Voice-C-D, PIN-D-C) and (Voice-C-D, QR-C-D) pairs showed statistically significant difference with a p-value of 0.001 for both. Similarly, for the perception of security between the (Voice-C-D, QR-C-D) pair with a p-value of 0.004 and (Num-C-D, QR-C-D) pair with a p-value of 0.012 we noticed statistical significance. Finally, for the perception of the efficiency, the (Voice-C-D, QR-C-D) pair with a p-value of 0.004 and (Num-C-D, QR-C-D) pair with a p-value of 0.012, a statistically significant result was observed.

Informal Participant Statements. Most of the participants found the QR-C-D method relatively effortless compared to the other methods. Many of the participants said the QR code is much easier and faster to use. Comparatively, the participants found the voice method to be less usable among other methods. Some of the participants expressed that they do not like to speak the checksum values out loud in public places, and therefore, are not comfortable with the Voice-C-D method. We quote some of the interesting comments:

- *"I would not suggest the voice recording, because it is insecure in my opinion, also mistakes can be made easily with voice recording. I suggest using the QR code, because it is faster and safer."*
- *"The first one [Num-C-D] wasn't easy for me, i needed support to get it done, the last one with QR codes was fun and easy, i feel maybe more secure to use than others"*
- *"For 2fa, if text input is required, I prefer using my computer to type (or copy) the text into the authenticating website. The best methods are those that don't require me to type at all (on computer or phone) but instead use confirmation links, Approval dialogues, or QR codes.*
- *"I prefer receiving codes via SMS, because I forward text messages to my computer, and can copy and paste the authentication code into my browser. Duo is my favorite 2FA app, because it gives me a pop up dialogue with an "approve" button, so I can login without typing any codes.*
- *In today's study, the QR option felt like the fastest option, but in real life it might not be the fastest. Today, I held the phone up to the screen and scanned one code after another. In real life,*

I would have to pull my phone out and open the app each time, so the total time to use the QR code would be similar to the other options."

7.4.4 Summary of the Results. After analyzing the 1,200 tasks that users performed in our study, we found out that the QR-C-D method has the lowest error rate, lowest delay, and highest usability perception ratings among all methods. While Voice-C-D had the lowest SUS score and highest error rate compared to other methods, the average delay of the Voice-C-D method was in the second place after QR-C-D. As expected, the two semi-automated user-assisted methods (i.e., QR-C-Dand Voice-C-D) incurred lower delay compared to the manual PIN/checksum copy-confirm methods. In contrast, PIN-D-C and Num-C-D showed a relatively higher delay compared to QR-C-D and Voice-C-D, due to the fully manual copying of the PIN/checksum in Num-C-D and PIN-D-C (one on the phone and the other on the client). The two tasks show somehow similar error rates (around 5%) and users' perception of adoptability, trust, security, efficiency, and usability. However, we observed that users are more comfortable entering the PIN on the client, than on the smartphone. Even though the size of the PIN and the checksum were the same in our study, users seemed to prefer using a full-size keyboard on the client than the smaller-form keypad on the smartphone to enter the numbers.

7.4.5 Limitations. Similar to any other study involving human subjects, our study also had certain limitations. Some of these limitations stem from the nature of the lab-study and the fact that the users may feel being controlled and under observance of the examiner. In some of the tasks, this may impact the users' perception of usability/security. For example, in the lab setup users may not be familiar with the people around them, hence they may be uncomfortable speaking the checksum in the Voice-C-D method. In real life, they may be in their homes and may not feel this discomfort. Although OpTFA does not require the secrecy of the checksum value, users may think otherwise, contributing to a lower usability ranking for the Voice-C-D method.

Recall that in OpTFA the user enters the master password on the client and compares the checksum on the device, and the scheme then computes the hardened password that authenticates the user to the server. From the user's perspective, there are two tasks: (1) entering the password, and (2) comparing the checksum. In the traditional PIN-based 2FA scheme, the user's tasks are: (1) entering the password, and (2) entering the PIN. Since the first task is the same in both schemes, in our study we only evaluated the usability of the system with respect to the second task.

We simulated the PIN/checksum entry or comparison, but not the setup and installation. Also, we did not set a primary task for the users (e.g., sending an email). This study design only compared the PIN/checksum entry method among the new OpTFA and traditional PIN-TFA models. This choice shortened the study and helped eliminate the fatigue affect while limiting the scope of the study.

In this study, we collected data from 30 participants. Our sample size is commensurate with that of many prior usability studies of authentication systems (e.g., Karole et al. [52], Chiasson et al. [33], and Acar et al. [24]). While collecting data from a larger and more diverse sample can be continued in future, we believe that our study has sufficient statistical power to provide meaningful results. Our analysis revealed that many of the differences we noted between the tested methods are statistically significant and could not have occurred by chance and, therefore, can be generalizable to larger samples. However, even though our participants' demographics shows higher number of male participants, the Mann-Whitney test between the female and the male participants to compare the two groups did not show statistically significant difference. Moreover, we do not think our results will be significantly affected based on education, technical background, or age, since the participants need to perform only simple tasks such as copying the PIN, scanning the QR Code, and reading few digits. In case these factors impact the usability scores, we assume all

methods will be impacted somewhat similar and not just one isolated apprach. Hence, the difference among different PIN/checksum entry methods will remain the same. Running an experiment with a larger number of participants from different age groups and technical background could help us to scientifically examine the impact of age and education on the usability.

8 DISCUSSION OF RELATED WORK

Device-enhanced password-authentication with security against offline dictionary attacks (ODA). There are several proposals in cryptographic literature for password authentication schemes that utilize an auxiliary computing component to protect against ODA in case of server compromise. This was a context of the *Password Hardening* proposal of Ford-Kaliski [38], which was generalized as *Hidden Credential Retrieval* by Boyen [28], and then formalized as ***(Cloud) Single Password Authentication*** **(SPA)** by Acar et al. [24] and as a DE-PAKE by Jarecki et al. [45]. These schemes are functionally similar to a TFA scheme if the role of the auxiliary component is played by the user's device D, but they are insecure in case of password leakage, e.g., via client compromise. We note that the scheme proposed in Reference [24] also shows a Mobile Device SPA, which provides client-compromise resistance, but it requires the user to type the password onto the device D, and to copy a low entropy value from D to C, thus increasing the amount of manually transmitted data. By contrast, OpTFA dispenses entirely with manual transmission of information to and from D. The threat of an ODA attack on compromise of an authentication server also motivated the notion of ***Threshold*** **Password-authenticated Key Exchange (T-PAKE)** [59], i.e., a PAKE in which the password-holding server is replaced by n servers so that a corruption of up to $t < n$ of them leaks no information about the password. In addition to general T-PAKE's, several solutions were also given for the specific case of $n = 2$ servers tolerating $t = 1$ corruption, known as *2-PAKE* [30, 53], and every 2-PAKE, with the user's device D playing the role of the second server, is a password authentication scheme that protects against ODA in case of server compromise. However, as in the case of References [24, 28, 38, 45], if a password is leaked, then 2-PAKE offers no security against an active attacker who engages with a single 2-PAKE session. Isler and Kupcu [76, 77] present generalizations of the DE-PAKE work [45] (the basis of our work too) by noting that the device in DE-PAKE and the login servers can be distributed over several machines essentially using Threshold OPRFs. However, none of these techniques provide second-factor security or security against password compromise. If the password leaks, then the security has already degraded. However, the ability to distribute servers and devices applies to our work too but the second factor requires physical possession by the user, hence it will typically be implemented with one device.

TFA with ODA security. Shirvanian et al. [67] proposed a TFA scheme that extends the security of traditional PIN-based TFAs against ODA in case of server compromise. However, OpTFA offers several advantages compared to Reference [67]: First, Reference [67] relies on PKI (the client sends the password and the one-time key, OTK, to the PKI-authenticated server), while OpTFA has both a PKI-model and a PKI-free instantiation. Second, Reference [67] assumes full security of the t-bit D-C channel for OTK transmission, while we reduce this assumption to a t-bit *authenticated* channel between C and D. Consequently, we improve user experience by replacing the *read-and-copy* action with simpler and easier *compare-and-confirm*. However, Reference [67] can use *only* the t-bit secure D-C link while OpTFA requires transmission of full-entropy values between D and C.

TFA with the second factor as a *local* cryptographic component. Some Two-factor Authentication schemes consider a scenario where the second factor is a device D capable of storing cryptographic keys and performing cryptographic algorithms, but unlike in our model, D is connected directly to client C, i.e., it effectively communicates with C over secure links. (However, security must hold assuming the adversary can stage a lunch-time attack on device D, so D cannot

simply hand off its private keys to C.) The primary example is a USB stick, like YubiKey [21], implementing, e.g., the FIDO U2F authentication protocol [13, 57]. A generalized version of this problem, including biometric authentication, was formalized by Pointcheval and Zimmer as *Multi-Factor Authentication* [64], but the difference between that model and our TFA-KE notion is that we consider device D, which has *no pre-set secure channel with client* C. Moreover, to the best of our knowledge, all existing MFA/TFA schemes even in the secure-channel D-C model are still insecure against ODA on server compromise, except for the aforementioned TFA of Shirvanian et al. [67].

Alternatives to PIN-based TFA with remote auxiliary device. Many TFA schemes improve on PIN-based TFAs by either reducing user involvement, by not requiring the user to copy a PIN from D to C, or by improving on its online security, but *none of them protect against ODA in case of server compromise*, and their usability and online security properties also have downsides.

PhoneAuth [34] and Authy [19] replace PINs with S-to-D challenge-response communication channeled by C, but they require a pre-paired Bluetooth connection to secure the C-D channel. A full-bandwidth secure C-D channel reduces the three-party TFA notion to a two-party setting, where device D is a local component of client C, but requiring an establishment of such secure connection between a browser C and a cell phone D makes a TFA scheme harder to use. TFA schemes like SlickLogin (acquired by Google) [3], Sound-Login [18], and Sound-Proof [51] in essence attempt to implement such secure C-to-D channel using physical security assumptions on physical media, e.g., near-ultrasounds [3], audible sounds [18], or ambient sounds detecting proximity of D to C [51], but they are subject to eavesdropping attacks and co-located attackers.

Several TFA proposals, including Google Prompt [16] and Duo [11], follow a *one-click* approach to minimize user's involvement if D is a data-connected device like a smartphone. In References [11, 16], S communicates directly over data-network to D, which prompts the user to approve (or deny) an authentication session, where the approve action prompts D to respond in an entity authentication protocol with S, e.g., following the U2F standard [13]. This takes even less user's involvement than the compare-and-confirm action of our TFA-KE, but it does not establish a strong binding between the C-S login session and the D-S interaction. E.g., if the adversary knows the user's password, and hence the TFA security depends entirely on D-S interaction, a man-in-the-middle adversary who detects C's attempt to establish a session with S, and succeeds in establishing a session with S before C does, will authenticate as that user to S, because the honest user's approval on D's prompt will result in S authenticating the adversarial session.

Usability Study of TFA Schemes. Several studies have evaluated the usability of two-factor authentication methods (e.g., hardware tokens, SMS, email). In a study published about the usability of one factor and TFA in phone banking [40], a survey was conducted and users answered questions about the usability of different phone banking authentication methods. The result indicated that while TFA is considered to be more secure compared to password-only authentications, it offers lower usability. In another study about the usability of e-banking authentication tokens [74], usability and efficiency of different tokens were compared. While the users' perception of the security and the usability of the tokens were different, this study once again confirms that users preferred the token with the highest usability even though their perception of the security of such token was the least among all. In Reference [35], three popular TFA schemes, i.e., codes generated by security tokens, one-time PINs received via email or SMS, and dedicated smartphone apps (e.g., Google Authenticator) were studied. This study shows that smartphone apps offer a higher adoption possibility compared to other methods.

The usability of different checksum/fingerprint verification with respect to the fingerprint exchange and presentation (e.g., hexstring, numeric, images) has also been studied in the past [36, 68]. Similarly, there exists several studies that use SAS protocols and different out-of-band channels for the purpose of device pairing [50, 56] to establish secure connection between two

(or more) wireless devices communicating over a short-range channel, such as WiFi or Bluetooth. Even though these studies have considered verification on smartphone applications, the user interaction in these schemes is completely different from the user interaction in OpTFA. In the mentioned studies, the user typically performs a compare-confirm verification of fingerprints displayed on their devices or verbally recited to them. In contrast, the type of checksum verification in OpTFA is copy-confirm where the copying part is performed manually by the user (e.g., Num-C-D and Voice-C-D) or somewhat automatically by user's assistance (e.g., QR-C-D) and the confirmation is performed automatically by the device.

Another communication channel that is being used in many security applications is the QR code (e.g., TFA setup [11, 14], TFA PIN transfer [67], device pairing [37, 61, 70], and checksum comparison [17, 20]). While QR codes have been studied in the past, one unique difference between OpTFA use of QR code from other security protocols could be in the asymmetric nature of the devices between which the code is transferred (i.e., a full computer terminal and a phone), as opposed to symmetric devices, such as two phones, in other applications. Besides, in some of these applications (such as device pairing and checksum comparison), the QR code should be transferred in both direction and its equality be verified on both parties of the protocol, while OpTFA only requires the client to be authenticated and therefore the QR code is transferred in only one direction (from the client to the device). Note that although some security applications such as TFA, transfers the code in the same direction, their purpose is the initial setup and hence the transfer is a performed only once.

Another line of studies related to the usability of device-based authentication is password manager apps, in which the user reads the passwords from the password manager apps on the device and copy it to the web page [60]. Isler et al. [42] studied usability of their mobile and cloud-based single password authentication and compared them with traditional password and 2FA authentication. Overall, their study with 25 participants shows that SPA could be a more usable alternative compared to traditional password-based and 2FA authentication. A fundamental difference between the usability of OpTFA and other device-based one factor and two-factor authentication solution is the direction of the user interaction task. While in other device-based authentication solutions the user should copy the authentication token from the device to the client, in OpTFA the checksum is transferred from the client to the device for verification. Of course, once the channel gets authenticated the PIN is transferred from the device to the client. However, this PIN transfer is not assisted by the user (only the checksum comparison is assisted by a human user). Therefore, while our work follows a similar user study methodology, it is essentially different from other studies.

Another related study is Reference [66], which performed a usability evaluation of the security code verification deployed for the purpose of end-to-end encryption in Signal. One main difference between code verification in protocols such as Signal and OpTFA is that, in our protocol, the checksum comparison is between two different devices (a phone and a laptop as opposed to two phones) that are in close proximity and in possession of one single user. Also, the signal protocol results in long security codes to compare, while we have short codes in OpTFA.

9 CONCLUSION

We designed a TFA system that offers end-to-end security by protecting against a "man-in-the-middle" attacker that controls the communication channels between all parties and can compromise any party. In particular, protection is provided upon server compromise, device compromise, and client compromise (which implies password leakage). Our system utilizes the "short authenticated strings" model [72] to add TFA security against attacks on the channel between the TFA device and the client machine. We formulated a rigorous security model for this setting and presented

a protocol that provably satisfies this security model. We also prototyped an implementation of this system based on device-to-client channels that require reduced user involvement compared to the TFA systems deployed today, and we evaluated the usability of the resulting system.

ACKNOWLEDGMENTS

We thank ACM TOPS anonymous reviewers for their helpful feedback and guidance.

REFERENCES

[1] RSA breach leaks data for hacking securid tokens. 2011. http://goo.gl/tcEoS.
[2] LinkedIn Confirms Account Passwords Hacked. 2012. http://goo.gl/AWB5KC.
[3] Google acquires slicklogin, the sound-based password alternative.2014. https://goo.gl/V9J8rv.
[4] Russian Hackers Amass Over a Billion Internet Passwords. 2014. Available at: http://goo.gl/aXzqj8.
[5] Hack Brief: Yahoo Breach Hits Half a Billion Users. 2016. https://goo.gl/nz4uJG.
[6] Sim swap fraud. 2016. http://goo.gl/y4Eogg.
[7] Sms-based two-factor authentication. 2016. https://bit.ly/2GiH4aN.
[8] Yahoo Says 1 Billion User Accounts Were Hacked. 2016. https://goo.gl/q4WZi9.
[9] Over 560 Million Passwords Discovered in Anonymous Online Database. 2017. https://goo.gl/upDqzt.
[10] Google Cloud Messaging. 2018. https://goo.gl/EFvXt9.
[11] Duo Security Two-Factor Authentication. 2019. https://goo.gl/e38UnB.
[12] Facebook stored hundreds of millions of passwords in plain text. 2019. https://www.theverge.com/2019/3/21/18275837/facebookplain-text-password-storage-hundreds-millions-users.
[13] FIDO Universal 2nd Factor (U2F) Overview. 2019. https://bit.ly/2IpPYH8.
[14] Google Authenticator Android app. 2019. https://goo.gl/Q4LU7k.
[15] Google stored some passwords in plain text for fourteen years. 2019. https://www.theverge.com/2019/5/21/18634842/googlepasswords-plain-text-g-suite-fourteen-years.
[16] Sign in faster with 2-Step Verification phone prompts. 2019. https://goo.gl/3vjngW.
[17] Signal by Open Whisper Systems. 2019. https://signal.org/.
[18] Sound Login Two Factor Authentication. 2019. https://goo.gl/LJFkvT.
[19] Two-factor authentication - authy. 2019. https://www.authy.com/.
[20] WhatsApp Simple, Secure, Reliable messaging. 2019. https://www.whatsapp.com/.
[21] YubiKeys: Your key to two-factor authentication. 2019. https://goo.gl/LLACvP.
[22] Zxing ("zebra crossing") barcode scanning library for java, android. 2019. https://github.com/zxing/zxing.
[23] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. 2001. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Proceedings of the Topics in Cryptology Conference (CT-RSA'01) (Lecture Notes in Computer Science)*, Vol. 2020. Springer.
[24] Tolga Acar, Mira Belenkiy, and Alptekin Küpçü. 2013. Single password authentication. *Comput. Netw.* 57, 13 (2013).
[25] Mihir Bellare, David Pointcheval, and Phillip Rogaway. 2000. Authenticated key exchange secure against dictionary attacks. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt'02)*.
[26] Steven M. Bellovin and Michael Merritt. 1993. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'93)*. 244–250.
[27] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security*. 967–980.
[28] Xavier Boyen. 2009. Hidden credential retrieval from a reusable password. In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (ASIACCS'09)*. DOI: https://doi.org/10.1145/1533057.1533089
[29] Xavier Boyen. 2009. HPAKE: Password authentication secure against cross-site user impersonation. In *Proceedings of the Conference on Cryptology and Network Security (CANS'09)*. Springer, 279–298.
[30] John Brainard, Ari Juels, Burt Kaliski, and Michael Szydlo. 2003. A new two-server approach for authentication with short secrets. In *Proceedings of the 12th USENIX Security Symposium*. 201–213.
[31] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usabil. Eval. Industry* 189, 194 (1996), 4–7. Retrieved from http://goo.gl/XDqBqg.
[32] Ran Canetti and Hugo Krawczyk. 2001. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. 453–474.

[33] Sonia Chiasson, Paul C. van Oorschot, and Robert Biddle. 2006. A usability study and critique of two password managers. In *Proceedings of the Usenix Security Conference*.

[34] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. 2012. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proceedings of ACM Conference on Computer and Communications Security*. ACM.

[35] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Greg Norcie. 2013. A comparative usability study of two-factor authentication. *arXiv preprint arXiv:1309.5344*.

[36] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. An empirical study of textual key-fingerprint representations. In *Proceedings of the USENIX Security Symposium*. 193–208.

[37] Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S. Lam. 2010. Secure, consumer-friendly web authentication and payments with a phone. In *Proceedings of the International Conference on Mobile Computing, Applications, and Services*. Springer.

[38] Warwick Ford and Burton S. Kaliski Jr. 2000. Server-assisted generation of a strong secret from a password. In *Proceedings of the IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'00)*. 176–180.

[39] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. 2006. A method for making password-based key exchange resilient to server compromise. In *Proceedings of the Advances in Cryptology Conference*.

[40] Nancie Gunson, Diarmid Marshall, Hazel Morton, and Mervyn Jack. 2011. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. *Comput. Secur.* 30, 4 (2011).

[41] Shai Halevi and Hugo Krawczyk. 1999. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)* 2, 3 (Aug. 1999), 230–268.

[42] Devriş İşler, Alptekin Küpçü, and Aykut Coskun. [n.d.]. User study on single password authentication. ([n.d.]).

[43] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2012. On the security of TLS-DHE in the standard model. In *Proceedings of the International Cryptology Conference (CRYPTO'12)*. 273–293.

[44] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. 2015. Highly efficient and composable password-protected secret sharing. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EuroS&P'15)*.

[45] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. 2016. Device-enhanced password protocols with optimal online-offline protection. In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (ASIACCS'16)*. Retrieved from http://ia.cr/2015/1099.

[46] Stanislaw Jarecki, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. 2018. Two-factor authentication with end-to-end password security. In *Proceedings of the International Conference on Practice and Theory of Public Key Cryptography (PKC'18)*.

[47] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *Proceedings of the Advances in Cryptology Conference (EUROCRYPT'18)*.

[48] Stanislaw Jarecki, Jubur Mohammed, Hugo Krawczyk, Maliheh Shirvanian, and Nitesh Saxena. [n.d.]. Two-factor password-authenticated key exchange with end-to-end password security. *Cryptology ePrint Archive* report 2018/033.

[49] Katie Kleemola John Scott-Railton. 2015. London Calling: Two-Factor Authentication Phishing From Iran. Retrieved from https://goo.gl/yt12xH.

[50] Ronald Kainda, Ivan Flechais, and Andrew William Roscoe. 2009. Usability and security of out-of-band channels in secure device pairing protocols. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS'09)*.

[51] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. 2015. Sound-proof: Usable two-factor authentication based on ambient sound. In *Proceedings of the USENIX Security Symposium*.

[52] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. 2011. A comparative usability evaluation of traditional password managers. In *Proceedings of the Information Security and Cryptology Conference (ICISC'11)*.

[53] Jonathan Katz, Philip D. MacKenzie, Gelareh Taban, and Virgil D. Gligor. 2005. Two-server password-only authenticated key exchange. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'05)*. 1–16.

[54] Swati Khandelwal. 2017. Real-world SS7 Attack. Retrieved from https://thehackernews.com/2017/05/ss7-vulnerability-bank-hacking.html.

[55] Hugo Krawczyk. 2005. HMQV: A high-performance secure Diffie-Hellman protocol. In *Proceedings of the Annual International Cryptology Conference*. 546–566.

[56] Arun Kumar, Nitesh Saxena, Gene Tsudik, and Ersin Uzun. 2009. Caveat emptor: A comparative study of secure device pairing methods. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*.

[57] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. 2016. Security keys: Practical cryptographic second factors for the modern web. In *International Conference on Financial Cryptography and Data Security*. Springer, 422–440.

[58] Chia-Chi Lin, Hongyang Li, Xiao-yong Zhou, and XiaoFeng Wang. 2014. Screenmilker: How to milk your Android screen for secrets. In *Proceedings of the Network & Distributed System Security Symposium*.
[59] Philip MacKenzie, Thomas Shrimpton, and Markus Jakobsson. 2002. Threshold password-authenticated key exchange. In *Proceedings of the Advances in Cryptology Conference (CRYPTO'02)*.
[60] Daniel McCarney, David Barrera, Jeremy Clark, Sonia Chiasson, and Paul C. van Oorschot. 2012. Tapas: Design, implementation, and usability evaluation of a password manager. In *Proceedings of the Annual Computer Security Applications Conference*.
[61] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. 2005. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 110–124.
[62] D. M'raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. 2005. *Hotp: An HMAC-based One-time Password Algorithm*. Technical Report.
[63] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. 2011. *Totp: Time-based One-time Password Algorithm*. Technical Report. Retrieved from https://goo.gl/9Ba5hv.
[64] David Pointcheval and Sébastien Zimmer. 2008. Multi-factor authenticated key exchange. In *Proceedings of the Conference on Applied Cryptography and Network Security*.
[65] Nitesh Saxena, Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. 2006. Secure device pairing based on a visual channel. In *Proceedings of the IEEE Symposium on Security and Privacy*.
[66] Svenja Schröder, Markus Huber, David Wind, and Christoph Rottermanner. 2016. When SIGNAL hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *Proceedings of the European Workshop on Usable Security (EuroUSEC)*.
[67] Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. 2014. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'14)*.
[68] Maliheh Shirvanian, Nitesh Saxena, and Jesvin James George. 2017. On the pitfalls of end-to-end encrypted communications: A study of remote key-fingerprint verification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 499–511.
[69] Victor Shoup. 2004. ISO 18033-2: An Emerging Standard for Public-Key Encryption. Final Committee Draft.
[70] Bradley Neal Suggs. 2013. Pairing a device based on a visual code. U.S. Patent App. 13/194,267.
[71] Ersin Uzun, Kristiina Karvonen, and Nadarajah Asokan. 2007. Usability analysis of secure pairing methods. In *Proceedings of the International Conference on Financial Cryptography and Data Security*.
[72] Serge Vaudenay. 2005. Secure communications over insecure channels based on short authenticated strings. In *Proceedings of the Advances in Cryptology Conference (CRYPTO'05)*.
[73] Ding Wang and Ping Wang. 2014. On the usability of two-factor authentication. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*. Springer, 141–150.
[74] Catherine S. Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack. 2009. User perceptions of security, convenience and usability for ebanking authentication tokens. *Comput. Secur.* 28, 1–2 (2009), 47–62.
[75] Catherine S. Weir, Gary Douglas, Tim Richardson, and Mervyn Jack. 2009. Usable security: User preferences for authentication methods in eBanking and the effects of experience. *Interact. Comput.* 22, 3 (2009), 153–164.
[76] Devriş İşler and Alptekin Küpçü. 2018. Distributed Single Password Protocol Framework. Cryptology ePrint Archive, Report 2018/976. Retrieved from https://eprint.iacr.org/2018/976.
[77] Devriş İşler and Alptekin Küpçü. 2018. Threshold Single Password Authentication. Cryptology ePrint Archive, Report 2018/977. Retrieved from https://eprint.iacr.org/2018/977.

Received May 2019; revised August 2020; accepted January 2021