Two-Factor Authentication with End-to-End Password Security and Reduced User Involvement

Stanislaw Jarecki*, Hugo Krawczyk†, Maliheh Shirvanian‡ and Nitesh Saxena‡
*University of California at Irvine Email: stasio@ics.uci.edu †IBM Research Email:hugo@ee.technion.ac.il
‡University of Alabama at Birmingham Email: maliheh@uab.edu , saxena@uab.edu

***Abstract*—We present a secure two-factor authentication (TFA) system based on the possession by the user of a password and a crypto-capable device. Security is "end-to-end" in the sense that the attacker is allowed attacks on all parts of the system: control of all communication links between all parties (passive and active man-in-the-middle attacks), full compromise of parties (server, device, client machine), and the ability to learn users' passwords. In all cases the system provides the highest security attainable given the set of compromised parties and/or passwords. Our solution carefully builds on the DE-PAKE (Device-Enhanced PAKE) scheme from Jarecki et al. and the Short Authenticated String (SAS) message authentication scheme of Vaudenay. Starting from these components we develop a full TFA solution that not only protects against an attacker that learns the user's password but also improves security even in the cases that the attacker has not obtained the user's password. We provide a modular cryptographic design that can utilize any standard password-based client-server authentication scheme, with or without reliance on public-key infrastructure. We prove the security of our construction based on a formal model that we formulate.**

In addition to achieving end-to-end security, the use of a SAS mechanism also improves user experience by reducing the human involvement relative to typical TFA schemes. Particularly, we dispense with the need of the user copying strings from device to the client machine and we replace it with a simple string comparison (visually by the user or through automated means such as a QR code). Furthermore, we show that the benefits of using SAS can be enjoyed by other TFA mechanisms that enhance password authentication with one-time codes (or PINs) produced by a crypto-capable device and transmitted to the user's client machine. We show that the integration of SAS with such traditional TFA schemes improves security against eavesdropping and active attacks on the client-device link by virtually eliminating the effectiveness of eavesdropping attacks (that target the one-time code returned by the device) and making active attacks impractical even when all the user does is to compare a 6-digit string computed by client and device.

We validate the practicality of the proposed protocols by implementing and testing several of our TFA protocol variants. In particular, we introduce various physical designs of the authenticated transmission of the SAS string, including automatically capturing a QR code encoding SAS by device's camera, or recording and decoding user's speaking of the SAS by device's microphone. These methods can improve the security of the TFA system by increasing the channel capacity and enhance the user experience by automating the SAS comparison task.

I. Introduction

Passwords provide the dominant authentication mechanism for web security protecting a plethora of sensitive information stored by the Internet services. 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 (typically, a database of salted password hashes) can mount an *offline dictionary attack* against each user by testing all password against a dictionary of likely password choices. Offline dictionary attacks are a major threat (see e.g. [10], [11]), being routinely experienced by major commercial vendors [1], [8], [2], [7], [6]. Such attacks are particularly effective and responsible for the compromise of billions of user accounts [12], and because users often re-use their passwords across multiple services, compromising one service may compromise user accounts at other services.

A common line of defense against *online* password attacks and to counter the leakage of a user's password is the use of *two-factor password authentication* (TFA). A TFA scheme authenticates a user U to a server S provided that the user knows a password and has access to an auxiliary personal device D (e.g., a smartphone, a USB token, etc.). If D is not directly connected to the user's client C (i.e., the user's machine or terminal), D typically displays a short one-time *secret* PIN, either received from the server (e.g. using an SMS message) or computed by D based on a key shared with the server. The user then manually types the PIN into the client C in addition to supplying her password. These mechanisms require a read-and-copy action by the user in order to implement the transmission of the PIN as well as to prevent an attacker from learning it (which would allow an attacker in possession of the password to impersonate the user in that session, possibly after disrupting the user's session with the server). The read-and-copy technique limits the entropy of the PIN which helps guessing attacks and opens to a variety of eavesdropping attacks, including shoulder-surfing and PIN recording (from camera snapshots to electromagnetic leakage), and client-side and device-side attacks via keyloggers, screen scrapers and mobile malware (e.g., [27]). Very significantly, the vulnerability to PIN leakage prevents the use of higher-capacity channels for transmitting the PIN such as over WiFi or Bluetooth.

Examples of systems that are based on one-time PINs include TOTP [13], HOTP [9], Google Authenticator [4], FIDO U2F [3], and state-of-the-art schemes in the literature such as [29].

Our Work In a Nut-Shell: In this paper we first address the above shortcomings of TFA systems by presenting a mechanism that armors TFA schemes against PIN eavesdropping and man-in-the-middle attacks while at the same time relaxing the required work from the human user. Assuming a device capable of performing cryptographic operations, this mechanism enables the use of high-entropy PINs and high-entropy channels like WiFi, Bluetooth and specialized channels such as Google Cloud Messaging in our implementation. It can also support a setting where the client-device communication is routed through the server and over the internet. We then use this same mechanism as a basis to the design of a complete TFA system that offers end-to-end security against an attacker that controls all communication channels, can corrupt parties at will and can also learn users' passwords.

Our focus is on a setting where the device generates a *one-time code (OTC)* (which we often denote by z) that is transmitted to the client's machine through some communication channel so that authentication to the server by a user requires a password as well as the OTC. While traditional TFA schemes refer to the OTC as a PIN, we drop this terminology which has connotations of a short, human copyable string while we allow for arbitrary-size values.

Our Detailed Contributions: The technical contributions of our work are four-fold, including (1) our use of the SAS notion for TFA hardening, (2) our TFA end-to-end security protocol design, (3) formal modeling and (4) prototype implementation and testing.

1. The Use of SAS Notion in TFA Security: A crucial component in our solutions is the Short-Authentication-String (SAS) scheme of Vaudenay [33] that allows for the exchange of authenticated information between two parties without the need to preset any keys, secret or public, between the two. Authentication is achieved by each party computing a non-secret validation string, called a *checksum*, and verifying that they both computed the same value. It is shown in [33] that if the checksum is of length t, then the probability that an attacker tampered with the communication and yet the parties' checksums coincide is at most 2^{-t}. In our setting, the parties to establish the authenticated channel are C and D, and the verification of checksum equality is done by the user, e.g., D and C display the checksum and the user verifies and confirms equality in the device (our usage of SAS is uni-directional hence device-only confirmation suffices). Since an authenticated channel can be used to bootstrap a confidential channel (e.g., run Diffie-Hellman over the authenticated channel), then we can use SAS to create a confidential channel between D and C over which to send the one-time code OTC.

By following this approach we achieve all of the following benefits: the OTC is not limited anymore to a copy operation by the user, hence allowing for any communication medium between C and D; the OTC can now be full fledge (e.g. 128 bits) instead of the ≈ 20 bits of human-transferred communication; the channel over which OTC is sent is encrypted; the user experience is improved by relaxing a copy operation to easier tasks such as a visual comparison and by dispensing with any involvement of the user with secret information (there is no secrecy requirement on the checksum value). More fundamentally we have achieved two essential security requirements. First, the user confirmation operation serves to verify equality but also prove possession of the device at the time of authentication. Second, we achieve full cryptographic security of the TFA scheme against OTC guessing and passive eavesdropping attacks on the C-D link. Lastly, OTC is secured even against active attacks. The only option to learn the OTC by such an attacker (other than compromising the device or breaking into the client during the OTC transmission), is to break SAS authentication which has probability of at most 2^{-t} for checksums of length t. Even for $t = 20$ (e.g., if the user visually compares a 6-string) the attack becomes essentially impractical (see Section III); and if comparison uses more automated means, e.g., capturing a QR code displayed by C with D's camera or the user reading the string into the device (as we implement), then t is large, say 128, and the attack is fully infeasible.

2. TFA with End-to-End Password Security: The SAS-based mechanism presented above serves as a general tool applicable to a large variety of TFA schemes that depend on the transmission of a one-time code (OTC) from device to client. Here we use this tool to build our strongest solution, namely, a two-factor authentication protocol that provides *end-to-end password security* in the sense that it contemplates an adversary A that is capable of controlling and compromising all components of the system. A has full control of all communication links between all parties (passive and active man-in-the-middle attacks), can compromise any party in the system (server, device, client machine), and has the ability to learn users' passwords. In all cases the goal is to ensure the highest security attainable under these attack capabilities given the set of compromised parties and/or password and leaving unavoidable online attacks as the only exploitable attack venue (except when both device and server are compromised in which case an offline dictionary attack is also unavoidable). This strong security notion is formally captured by our security model, to which we refer as TFA-KE (see below), and our solution is formally proved to satisfy this model.

The starting point for our solution is the Device-Enhanced Password-Authenticated Key Exchange (DE-PAKE) protocol from [24] which considers the same set of parties $(\mathsf{U}, \mathsf{S}, \mathsf{D}, \mathsf{C})$ and same communication and adversarial models as we do *except that [24] does not allow for client compromise nor it protects against password disclosure.* Indeed, the solution in [24] breaks upon the leakage of a user's password to the attacker. Our goal is to close this gap in achieving end-to-end security by armoring the scheme with a two-factor authentication (TFA) capability. By doing so not only we provide security upon password leakage but we actually *strengthen the security achieved by [24] even in the cases that the attacker has not obtained the user's password.*

Our main approach is to augment the DE-PAKE protocol with a TFA capability that exploits the presence in DE-PAKE of a crypto-capable device D. In our solution, the device is initialized with a PRF key shared with the server S which is used by both parties to generate a OTC z unique to each authentication session (z is the output of the PRF computed on a server-generated nonce). Authentication by the user (through its client machine C) requires both the user's password and the possession of the session-unique OTC z, where the latter is transmitted from D to C. Since we allow for full control of the D-C link by the attacker, we need to protect the transmission of z for which we use the SAS mechanism as discussed above to establish a confidential channel from C to D. While this addition provides defense against an attacker that has learned the password, it is not enough to provide the full ("highest attainable") security that we target. Two subtle but crucial points that we uncover are the need for establishing a binding channel between S and C (implemented through an ephemeral unauthenticated key exchange) and the need to carefully compose the OTC z with the password authentication component (PAKE) of the DE-PAKE protocol (we use z as a key that authenticates the PAKE part of the DE-PAKE scheme). We show that omitting any of these mechanisms allows for attacks that violate the security definition.

Once we put all these elements together we obtain our TFA-KE solution that we formally prove to satisfy the stringent requirements of our security model. One important property that we inherit from the DE-PAKE protocol is its modularity that allows client-server interaction based on any password protocol (secure against server compromise). In particular, this can be a PKI-based protocol as the standard password-over-TLS scheme or it can be a PAKE protocol that works without any reliance on PKI. In the former case, the password-over-TLS protocol is simply augmented by concatenating the OTC z to the transported password.

3. TFA-KE Model: Formalizing End-to-End Password Security: We prove the security of our TFA-KE solution based on a formal TFA-KE security model that we introduce to capture the adversarial capabilities and strong security guarantees as described above. We base this model on the *Device-Enhanced PAKE (DE-PAKE)* model of Jarecki et al. [24]. The latter extends the standard PAKE model [16] by considering a setting where a user U authenticates to a server S using a password pwd and an auxiliary device D, where user interaction is carried through a client machine C that communicates with both server and device. The security model allows the attacker to control all communication channels between servers, clients and devices and also considers security upon corruption of servers and devices *but does not allow for the corruption of clients or learning of users' passwords.* The security definition is "maximalist" in that it requires that the only feasible attacks are those based on unavoidable brute-force online dictionary attacks. Offline attacks should be infeasible even upon server compromise in a strong sense: as long as the attacker does not get hold of the user's device, the difficulty of an impersonation is the same with or without the information stored at the server. Similarly, compromise of the device only should not allow for an offline attack. Offline attacks are possible (and unavoidable in this model) only if both device and server are compromised.

While defense against offline attacks is as strong as one can hope, the DE-PAKE model does not consider client corruption or other forms of password leakage and, as said, the solution from [24] breaks upon the leakage of a user's password to the attacker. Here we enhance the DE-PAKE model by adding to the attacker's capabilities the possible compromise of client machines and passwords. The resultant model, that we denote by *TFA-KE*, extends DE-PAKE with a TFA capability as discussed earlier where user authentication needs to combine possession of the password with a verified "human-authenticated" access to the device. Specifically, we augment the DE-PAKE communication model with a SAS channel between client and device and set the security requirements in terms of maximal success probability for the attacker as a function of both the password entropy d (i.e., passwords drawn at random from a dictionary of size 2^d) and the SAS channel capacity t.

For example, while in the PAKE and DE-PAKE models each online authentication attempt by the attacker has 2^{-d} probability of success, in our case this bound must be at most $2^{-(d+t)}$. It means that the only feasible strategy for the attacker is to simultaneously guess the password *and* the SAS-channel checksum as well as to perform an online interaction with the server, client and device (acting as a man-in-the-middle between the latter two for breaking the SAS authentication). Since the TFA-KE model allows for such interaction of the attacker with the parties, this bound represents the highest security one can achieve. In the case that the password is leaked (for which the PAKE and DE-PAKE models provide no security), TFA-KE sets the attacker's success probability to at most 2^{-t}, namely, it succeeds only if it acts as a man-in-the-middle between client and device and is lucky enough to guess the parties' checksum. If the server is compromised the above bounds should still hold while if the device is compromised then the impersonation probability should be no more than 2^{-d} (falling back to the DE-PAKE case). Only if both server and device are compromised, an offline attack is possible. Refer to Section IV for the complete details of the model and the bounds in all compromise settings. We stress that similarly to the PAKE and DE-PAKE models, TFA-KE ensures the security of keys exchanged between user and server and not just the security of authentication.

4. TFA System Design, Implementation and Testing: We provide the full, server-side (JavaScripts and PHP Scripts), client-side (Chrome browser extensions) and device-side (Android app), design and implementation of a concrete instantiation of the TFA-KE protocol, and report on its performance. We implement the underlying D-C communication channels, using Google Cloud Messaging. Over this channel, we implement and run the PTR protocol from [24] as well as SAS authentication [33]. We implemented TFA with two PAKE instantiations, a PKI-free and a PKI-based one. All the implementations are based on well-recognized security libraries (e.g., Java Security, BouncyCastle, and Stanford Java Script Libraries) and none of our instantiations requires modification to the browser.

A main component of our TFA protocol is the authenticated communication channel from C to D, over which the checksum implementing SAS authentication is transferred/validated. We provide the design and implementation of two practical checksum validation channels, and demonstrate their feasibility. The first design is based on a full bandwidth QR Code, in which the checksum is encoded in a QR code and is displayed on the client's terminal. The checksum is recognized by a QR Code decoder (developed using Zebra Crossing library) and validated on the smartphone app. The second design is an audio-based channel, implemented on top of the advanced human speech transcription technology. In this model, the user speaks the checksum (displayed on the terminal) and a transcriber tool (developed using Android.Speech API) recognizes and validates the checksum.

II. Related work.

Shirvanian et al. [29] proposed TFA schemes which, similarly to us, protect against both the server and the client compromise by extending security of TFA to resistance against server compromise. Some of the advantages of our schemes compared to [29] include the following: First, [29] is fully dependent on PKI (the client sends the password and the OTC to the PKI-authenticated server) while the modularity of our approach allows for PKI-based and non-PKI PAKE schemes; Second, [29] assumes physical security of the secret OTC transmission while in our case equality verification of a non-secret string suffices; Third, we improve user experience by replacing the OTC copy action with a simpler/easier equality verification. Lastly, we formulate a more general model of TFA-KE security, with optimal security bounds, which applies to TFA protocols where all three parties C,S,D communicate over public channels. (On the other hand, the protocols of [29] applied in the setting where *all* D-C communication is contained in the t-bit OTC message, while our protocols several hundred bits of D-C and C-D communication.)

The work of [14] showed device-enhanced password authentication schemes which, similarly to the DE-PAKE scheme of [24], protect against the server compromise. However, the "cloud service" version of their scheme, similarly as the DE-PAKE scheme of [24], does not protect against client compromise. The "mobile device" version of their scheme does provide client-compromise resistance but it requires the user to type the password onto the device D, and to copy a cryptographic key from D to C, thus increasing the demand for manually transmitted information compared to today's standard TFA's, in contrast to our work which shows that one can achieve TFA with resistance to client, device, and server compromise, while entirely dispensing with the need for manual tranmission of information to and from D.

PhoneAuth [18] is an academic software token TFA scheme that leverages Bluetooth communication between the browser and the phone to eliminate user-phone interaction. The Bluetooth channel enables the server (through the browser) and the phone to run a challenge-response protocol which provides second authentication factor. Unlike our work, this scheme relies on a pre-paired Bluetooth connection to thwart active and passive attacks on the client-device channel. Authy [15] is another approach that allows seamless TFA using Bluetooth communication between the computer and the phone. However, Authy also requires a pre-paired Bluetooth connection. Another recent work is U2F [3], an open TFA standard with long one time codes that enables users to access multiple services, with one single device. However, this approach, similar to traditional TFA, may not remain secure under server and/or device compromise.

Sound-Proof [25] is a minimal effort TFA system that leverages ambient sounds to detect the proximity of the second factor (phone) and the browser. However, this approach is insecure against co-located attackers as well as remote attackers who can predict the user's acoustic environment. Moreover, this approach has also been shown insecure against the remote attacker who can make the second factor device, i.e., the phone, create some predictable or already known sounds, as demonstrated in [31]. Similar to Sound-Proof, our work aims to reduce the human involvement in the authentication process but aims to do so by providing strong end-to-end security guarantees in a provably secure way.

III. SAS Authentication and TFA Security

In this paper we consider TFA schemes (both existing ones as well as our own from Section VI) where password-based authentication of a user U to a server S is complemented by the client C sending to S a session-specific one-time code (OTC) z generated by an auxiliary device D possessed by U.[1] The purpose is to prevent an attacker that has learned U's password but does not possess the device from impersonating U to S. Thus, the security of the TFA scheme fully depends on the secrecy of z. So how does one guarantee a confidential channel from D to C? In typical TFA schemes, the lack of such channel forces the schemes to assume some form of physical security, typically letting the (human) user himself implement this channel through a manual copy into C of an OTC displayed by D.

Here we present a solution for D-C channel confidentiality based on the *SAS Message Authentication* scheme of Vaudenay [33]. This scheme, to which we will refer just as SAS, allows the transmission of a message from a sender to a receiver so that the receiver can check the integrity of the received message. Specifically, the abstract SAS model assumes two communication channels. One that allows the transmission of messages of arbitrary length and is controlled by an attacker (that can change traffic at will), and another channel that allows sending up to t bits that cannot be changed by the attacker (neither channel is assumed to provide secrecy). We will refer to these channels as the *open channel* and the *SAS channel*, respectively, and will call the parameter t the *SAS channel capacity*. An instantiation of the SAS model is called *secure* if the probability of a (computationally bounded) attacker to change a message sent over the open channel is no more than 2^{-t} (plus a negligible fraction).

A secure SAS implementation was provided in [33] and is presented in Figure 1 (see also Figure 2) where sender and receiver are denoted by C and D, respectively, the parties in our application. The SAS channel in this case reduces to the comparison of two strings of length t, $\mathsf{checksum}_C$ and $\mathsf{checksum}_D$, computed by sender and receiver, respectively.

[1]Our presentation here is not concerned with the specific method of generating z which is typically computed by D as the output of a PRF under a key shared with S on a synchronized timestamp [13] or counter [9], or on a challenge received from S as in Section VI.

Parties. C (sender), D (receiver).

Message authentication. Message M_C sent from C to D who verifies integrity of the received message via checksum received over the C-to-D SAS channel.

Parameters. Security parameter: κ; SAS Channel capacity: t.

Commitment. Implemented via a hash function H_{com} with κ-bit output.

SAS Protocol.

1) C chooses random strings R_C, d of length t and κ, resp. Computes commitment $\mathsf{Com} \leftarrow H_{\mathsf{com}}(\mathsf{M}_\mathsf{C}, R_C, d)$ and sends $(\mathsf{M}_\mathsf{C}, \mathsf{Com})$ to D.
2) D chooses random string R_D of length t and sends it to C.
3) C sends (R_C, d) to D and enters $\mathsf{checksum}_C \leftarrow R_C \oplus R_D$ into the C-to-D SAS channel.
4) D checks that $\mathsf{checksum}_D \leftarrow R_C \oplus R_D$ is the same as the value $\mathsf{checksum}_C$ received on the C-to-D SAS channel. If so and also $\mathsf{Com} = H_{\mathsf{com}}(\mathsf{M}_\mathsf{C}, R_C, d)$, D accepts M_C, else it rejects.

Fig. 1: SAS Message Authentication Protocol [33]

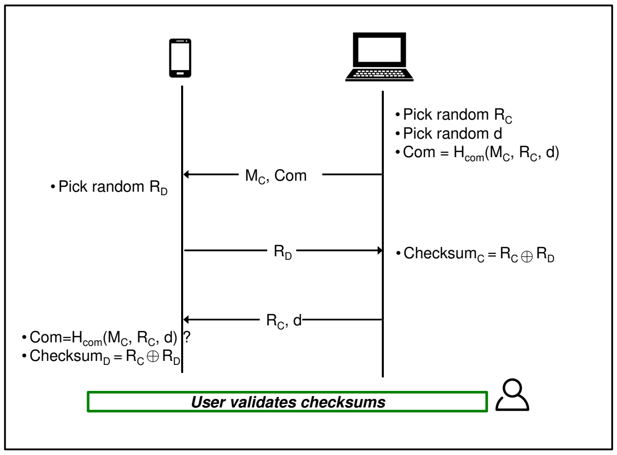
• Pick random R_C
• Pick random d
• Com = $H_{com}(M_C, R_C, d)$
M_C, Com
• Pick random R_D
R_D
• $Checksum_C = R_C \oplus R_D$
R_C, d
• Com=$H_{com}(M_C, R_C, d)$?
• $Checksum_D = R_C \oplus R_D$
User validates checksums

Fig. 2: Schematic Representation of the SAS Protocol (Fig. 1)

It is proved in [33] that the probability that an active attacker acting between D and C succeeds in changing the message 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 parties. Also, importantly, there is no requirement for checksums to be secret!

In our TFA application the protocol will be run between client C and device D where the SAS channel is implemented by each party displaying the locally computed checksum and using the (human) user to confirm equality (see more on this below), hence ensuring authentication of the exchanged message. Recall, however, that what is needed in the TFA application is a confidential channel from D to C for the transmission of the OTC z. This is implemented by using the SAS message authentication scheme for C to send an ephemeral public key to D who uses this key to encrypt z. That is, C chooses a pair of secret-public key pair $(\mathsf{sk}, \mathsf{pk})$ and sets $\mathsf{M}_\mathsf{C} = \mathsf{pk}$ in the SAS protocol. Only after the SAS channel is verified with the help of the user will D encrypt z under pk and send to C who can decrypt using sk.

Note that the user assistance is essential here to guarantee authentication which in turn is required for bootstrapping the D-to-C confidential channel. Very importantly, this user intervention serves as proof of possession of the device by the user during the authentication session. In other words, the simultaneous possession of password and device is required for a successful authentication of the user to the server.

How does the user confirm equality? There is a wide range of possible implementations, from simple visual inspection of the displayed checksums (e.g., a 6-digit number) confirmed by the user on the device app to more automated approaches such as the user using the device's camera to take a snapshot of a QR code displayed by the client. We expand on these techniques and their implementation in Section VIII. Note that in traditional TFA schemes that require the user to copy the OTC z from D to C, the length of z (a PIN in these cases) is restricted to a few characters. Here, we improve on all counts with respect to these traditional schemes:

- The *user experience is improved* by relaxing a copy operation to a visual comparison (or other easy tasks), and by dispensing with any involvement of the user with secret information (as said, there is no secrecy requirement on the checksum value).
- The value z can be of arbitrary length and encrypted *hence making guessing and eavesdropping attacks infeasible*.
- The checksum may be restricted in length depending on implementation but it is only needed against harder-to-mount active attacks on the D-C link and even a 20-bit checksum *makes active attacks essentially impractical* [2]; when using higher-bandwidth checksum (e.g., via QR codes) *active attacks become as infeasible as eavesdropping ones.*
- Even in cases where a traditional TFA uses a high bandwidth medium to transmit z (say, via QR), this transmission requires physical secrecy protection. In our case, thanks to our built-in protection of the D-C channel, one *can use any tappable communication channel* for transmitting z, such as WiFi, Bluetooth, the internet, or specialized channels such as Google's Cloud Messaging as in our implementation of Section VIII.

In the following sections we will use SAS authentication in essential ways to achieve TFA-KE protocols that achieve our strong notions of security. We first introduce our security model in the next section.

IV. TFA-KE Security

In this section we introduce the *two-factor authenticated key exchange (TFA-KE)* security model that defines the assumed

[2]It takes the attacker an expected number of 1 million active man-in-the-middle disruptions of sessions initiated by the client with the device to learn a single value z usable in a single login session and with each unsuccessful attempt being detected by the honest parties (this is more DoS than impersonation).

environment and participants in our protocols as well as the attacker's capabilities and the model's security guarantees. The starting point is the *Device-Enhanced PAKE (DE-PAKE)* model, introduced in [24], which extends the well-known two-party PAKE (password-authenticated key exchange) model [16] to a multi-party setting that includes *users* (communicating from *client machines*), *servers* to which users log in, and auxiliary *devices* (e.g., a smartphone, a watch, etc.) that users utilize during the password authentication process to strengthen security (these parties are denoted by U,C, S, D, respectively). Since our model and presentation is based on the DE-PAKE model we recall this model in Appendix A and we refer the reader to [24] for more details.

User devices are used in the DE-PAKE setting to provide resilience to offline attacks upon server compromise (which the two-party S-C PAKE model cannot possibly defend against). However, the DE-PAKE model does not provide security upon client compromise (or password leakage). Indeed, since the DE-PAKE's communication model assumes unprotected channels between all the parties, learning the user's password allows the attacker to act as the user itself (communicating with device and server). Thus, to extend the model to deal with security upon the attacker learning a password, one has to the very least assume some form of authentication in the C to D communication implemented through an active action of the user that requires physical possession of the device).

Consequently, our TFA-KE model, intended to add to DE-PAKE security protection upon password leakage, augments the DE-PAKE model with a *minimal* authentication abstraction on the client-to-device communication channel (and without assuming the storage of long-term keying information, such as PKI, by either client or user other than the user's password). For this we adopt the SAS channel abstraction of [33] (see Section III) where the communication from C to D is carried over two different channels. One is the regular unrestricted channel controlled by the attacker A, the other is a "t-capacity SAS-channel" where C can send up to t bits to D that A cannot change. Since these t bits can be used to authenticate the information sent over the regular channel this is equivalent to assuming a regular channel where the attacker succeeds in injecting its own messages with probability 2^{-t} [33]. Clearly, the larger t, the better security one can get, thus we will formulate the security definitions as a function of the parameter t (allowing for meaningful security bounds even with small values of t, e.g., $t = 20$).

We further quantify security in terms of the attacker's resources that include, in addition to computation time, parameters q_D, q_S, q_C, q'_C. The first two count the number of active sessions between the attacker and the device and server, resp., while q_C (resp. q'_C) counts the number of sessions where the attacker poses to C as the server (resp. D)[3]. We note that in the case that C communicates with S over a server-authenticated channel, e.g., TLS, $q_C = q'_C = 0$. As in the PAKE and DE-PAKE cases, the attacker's success is measured as the probability of successful impersonation (technically, this is quantified as the attacker's *advantage*, namely, by how much the attacker's probability to distinguish a key computed between S and C from random exceeds 1/2). In addition, security depends on the entropy of passwords, which we denote by d (assuming the password is chosen from a dictionary of size 2^d known to the attacker), as well as by the *capacity* t of the SAS channel.

The security requirements set by the TFA-KE model are the *strictest* one can hope for given the communication model (unprotected channels between the parties other than the SAS-channel between C and D). That is, wherever we require the attacker's advantage to be no more than a given bound, then there is an avoidable attack that achieves this bound in the given compromise setting.

Very importantly, and *in contrast to typical two-factor authentication solutions,* the TFA-KE model requires that the availability of the second factor D, not only provides security in case of client and/or password compromise, but it significantly strengthens online and offline security (by 2^t factors) *even when the password has not been learned by the attacker.*

A. TFA Security Definition

We consider the augmented communication model as described above and add to the attacker's capabilities the corruption of clients C in which case it learns C's internal state and also the user's password. All other adversarial capabilities as well as the test session experiment defining the adversary's goal are as in DE-PAKE model. We denote the adversary's advantage by $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}}$.

Definition 1. *A TFA-KE protocol* TFA *is* (T, ϵ)*-secure if it is correct, and for any password dictionary* Dict *of size* 2^d*, any t-bit SAS channel and any attacker* A *that runs in time* T*,* A*'s advantage* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}}$ *in distinguishing the session key output by the protocol from random is bounded as follows (for* q_S, q_C, q'_C, q_D, t *as defined above):*

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

Explaining the bounds. The security of the TFA scheme relative to the DE-PAKE model can be seen by comparing the above bounds to those in Definition 3 in Appendix A (recall that our notation q_C corresponds to the notation q_U in that definition). Here we explain the meaning of some of these bounds.

[3]In the DE-PAKE terminology [24] such adversarial sessions are called *rogue*. We note that the term q_C replaces the term q_U used in [24]; we prefer the subscript C as indicating interactions with the Client rather than directly with the User.

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 $(q_C+q_S)/2^d$. Note that in the PKI model $q_C = q'_C = 0$, in which case the bound reduces to $\min(q_S, q_D)/2^{t+d}$. The interpretation of this last bound, and similarly for the other bounds in this model, is that in order to have a probability $q/2^{t+d}$ to successfully impersonate the user, the attacker needs to run q online sessions with S *and also* q online sessions with D (in each of these sessions 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 authenticated over the SAS channel which happens with probability 2^{-t}). This is the best one can achieve in the TFA adversarial setting since an adversary who guesses both the user's password and the t-bit SAS-channel message can successfully authenticate as the user to the server and vice versa.

In case of client corruption (hence also password leakage), DE-PAKE offers no protection but in TFA-KE 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 when the attacker holds the user's password. Finally, in case of device corruption, the adversary's advantage is at most $(q_C+q_S)/2^d$ that matches the optimal PAKE probability (namely, when a device is not available), while upon server corruption, the adversary's probability of success (namely, impersonating the user to any uncorrupted server session) is at most $\min(q_S, q_D)/2^{d+t}$. In other words, learning server's private information necessarily allows the adversary to authenticate as the server to the client, but it *does not help in any way* to impersonate to any session at the server. Note that a DE-PAKE protocol also provides such resilience to server compromise but without the 2^{-t} factor in item 3 of Definition 1. In contrast, widely deployed PIN-based TFA schemes that transmit passwords and PINs over a TLS channel provide no protection at all in this case (an offline dictionary attack of complexity 2^d is sufficient to find the password in these cases).

TFA-KE Security in the PKI Setting. If we can rely on the PKI, i.e. if the user U can hold an authentic public key of the server S, then the TFA-KE security properties are strengthened because the adversary loses the ability to send rogue messages on behalf of S to C. In other words, under the assumption of a trustworthy PKI terms q_C and q'_C can be set to zero in all the security bounds above.

V. Motivating our design: Protocol TFA-PAKE

We present protocol TFA-PAKE which implements a full TFA-KE scheme based on any (generic) password-only protocol PAKE. The presentation is informal and intended to introduce the ideas and techniques underlying our main protocol, PTFA, which we define in Section VI on the basis of TFA-PAKE and the DE-PAKE work from [24], and which we prove to satisfy our strong notion of security in Section VII.

Figure 3 shows a schematic presentation of TFA-PAKE which combines any given password protocol PAKE with the SAS mechanism from Section III. It starts with the user

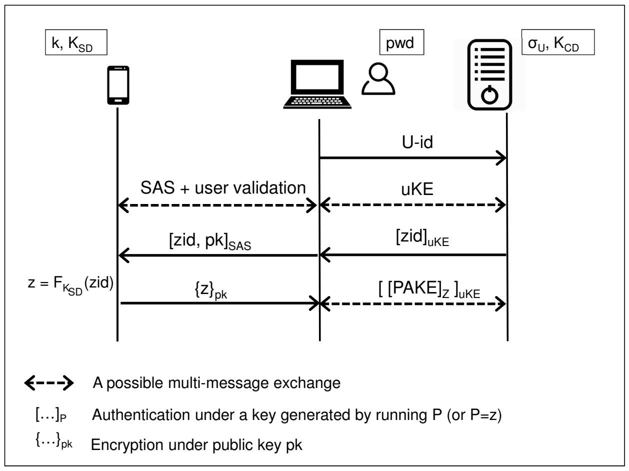
k, K_SD
pwd
σ_U, K_CD
U-id
SAS + user validation
uKE
[zid, pk]_SAS
[zid]_uKE
z = F_KSD(zid)
{z}_pk
[[PAKE]_Z]_uKE
A possible multi-message exchange
[...]_P Authentication under a key generated by running P (or P=z)
{...}_pk Encryption under public key pk

Fig. 3: Schematic representation of TFA-PAKE

U sending (via C) its userid information to the server S whom has stored user-specific information σ_U as well as a PRF key K_{SD} shared with the device D (both generated at the time of user's and device's registration). This initiates the run of an ephemeral session-specific key exchange (KE) protocol between S and C to agree on a shared key. Protocol KE does not have to be authenticated hence we refer to it as unauthenticated KE (uKE). All subsequent session traffic between C and S is authenticated using the key shared through uKE. We note that while we do not assume any pre-shared long-term keys, public or secret, between C and S, in a PKI setting where C holds a public key of S, the uKE protocol can be implemented using TLS.

In parallel with the establishment of uKE between C and S, client C runs an instance of the SAS protocol from Figure 1 with D to establish an authenticated channel between C and D where authenticity is validated via the user verification of checksums equality. C uses this authenticated channel to send a public key pk to D (used below to encrypt z).

While in the TFA protocol discussed in Section III we did not specify the input to the PRF used to compute the one-time code z, here we define the input to this computation as a value (challenge) sent by S and denoted zid (zid is implemented as a nonce and also serves as a unique identifier for the current session of protocol TFA-PAKE). This value is sent from S to C authenticated under the key derived by uKE and relayed by C to D authenticated under the SAS-established shared key. D responds with the value z sending it to C *encrypted* under the public key pk received from C (z is the only value in the protocol that requires encryption except if the specific PAKE protocol requires it).

Once C obtains the value z, it initiates a run of the PAKE protocol with S using U's password. To achieve the benefits of TFA and the optimal bounds in our security definition, PAKE is run needs to be carefully combined with the value z. For this we specify that z be used to authenticate the PAKE messages exchanged between C and S (e.g., under MAC keys derived from z and used by C and S to authenticate their

PAKE messages). Note that since we specified that all traffic between C and S runs over uKE, the MACed PAKE messages are sent over this uKE channel (this "double authentication" is necessary).

In the rest of this section we exemplify some of the subtleties surrounding the design of TFA-PAKE which also apply to our main protocol PTFA. It shows how the removal of some of the design elements leads to the violation of the security requirements set by our definition.

The need for careful bundling of PAKE and z. To gain intuition into the design of TFA-PAKE, assume the protocol changes the execution order and starts by running PAKE between C and S under the user's password to establish a key K (this ordering is common in deployed TFA systems); then it runs SAS as in Figure 3 so C can obtain z, and in the last flow C transmits z under the PAKE key K. This *decoupling* of PAKE and z provides much weaker security than we aim for. Most serious is the fact that the protocol is open to online guessing attacks against the password without need for the attacker A to possess the user's device or learn z. This obvious practical weakness is also reflected in the formal probability bounds for the attacker. After q_D attempts, the attacker has probability $q_D/2^d$ to learn the password after which it can impersonate the user in a session for which it learns z. Assuming a non-PKI case where the server does not authenticate via a public key, A learns z acting as man in the middle between S and C as follows. When U initiates a session with S through C, A intercepts this communication and takes the role of S. At the same time, A starts a session with S using the user's password. When S sends zid, A uses it as the zid in the open session with C. When A gets back from C the value z computed by D it uses it to complete authentication with S. The probability of success is $q_D/2^d$, violating the much stricter $\min\{q_S, q_D\}/2^{d+t}$ required by our definition.

In the PKI case, A cannot impersonate the server to C, so after learning the password it needs to learn z through breaking the SAS authentication. The latter has probability $\min\{q_D, q_S\}/2^t$ where in each attempt A acts as the client with S and with D, having probability of 2^{-t} to get D to answer the value zid input by A (it takes a break of the SAS channel authentication for D to accept a value zid coming from A rather from U). While the attack is harder to mount than in the non-PKI case, the success probability is still significantly larger than the optimal bound $\min\{q_S, q_D\}/2^{d+t}$ required by our definition (and achieved by our PTFA of Sec. VI in the PKI and non-PKI cases!).

The reason for the above weakness is that PAKE and z were decoupked from each other. It turns out that even if PAKE runs after receiving z (as in TFA-PAKE), the exact way PAKE and z are coupled is essential for security. For example, one could specify that the parties complete the PAKE run and only then C proves to S that it has the correct value z (say, by sending z encrypted under the session key computed by the PAKE run). However, if the PAKE protocol discloses the fact that S completed the computation of the session key (which is typical in PAKE protocols), one would learn that the password used in this PAKE session was correct. As before, this allows the attacker to try password guesses and learn the correctness of the guess without having to possess the device or having the correct z. Instead, by coupling PAKE and z via an authentication operation as specified in TFA-PAKE, the protocol would reach successful completion only when using the correct z.

The need for uKE. To illustrate a different aspect of the protocol design, we now consider a variant of TFA-PAKE where the uKE exchange 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, our required bound in this case is the stricter $\min\{q_S, q_D\}/2^t$ bound. What this bound requires is that for *each* attempt at learning z in the C-D channel, not only A needs to try to break SAS authentication but it also needs to establish a new session with S. To enforce this we require 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 flows, zid and z, are exchanged with the same party. If zid was sent to the legitmate client 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 our definition.

On single response to zid values. To achieve optimal bounds as we target, it is imperative that D never respond 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). Otherwise, the attacker gets multiple attempts at learning z for a single challenge zid without having to pair such attempt with a session with S as our $\min\{q_S, q_D\}/2^{d+t}$ bound requires.

Finally, we stress that while TFA-PAKE achieves some of our stingent bounds, it does achieve all of them (e.g., full security against server compromise). For this we need to incorporate into the protocol the PTR component of DE-PAKE protocols as well as an asymmetric PAKE [24] as presented next.

VI. PTFA: A SECURE TFA-KE PROTOCOL

We present our main protocol, PTFA, an instantiation of a more general TFA-KE protocol presented and proven secure in Section VII. As a corollary of that proof we obtain that PTFA satisfies our strong security definition from Section IV. The design of PTFA combines the TFA-PAKE protocol from the previous section with the DE-PAKE protocol from [24], denoted here by DEPAKE. We start by recalling DEPAKE and its main component PTR that we use in the specification of PTFA.

Initialization: On input the user's password pwd, pick random k in $\mathbb{Z}_q$ and set $\mathsf{rwd} = F_k(\mathsf{pwd}) = H_{\mathsf{oprf}}(\mathsf{pwd}, (H_{\mathsf{grp}}(\mathsf{pwd}))^k)$.
Initialize the asymmetric PAKE scheme aPAKE on input rwd and let σ be the resulting server's state.
Let K_z be a random key for PRF R, and set zidSet to the empty set.
Give $(k, K_z, \mathsf{zidSet})$ 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 to establish a key K_{CS} between them;
2) stores $z \leftarrow \mathsf{R}(K_z, zid)$, computes $e_S \leftarrow \mathsf{ACSend}(K_{CS}, zid)$, and sends e_S to C;
   C computes $zid \leftarrow \mathsf{ACRec}(K_{CS}, e_S)$, or aborts if decryption fails.

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

1) C generates PKE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}$, t-bit random value R_C, a (2κ)-bit random value d, and random r in Z_q;
   computes $a \leftarrow H_{\mathsf{grp}}(\mathsf{pwd})^r$, $\mathsf{M_C} \leftarrow (\mathsf{pk}, zid, a)$, $\mathsf{Com} \leftarrow H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$, and sends $(\mathsf{M_C}, \mathsf{Com})$ to D;
2) D on $(\mathsf{pk}, zid, a, \mathsf{Com})$, aborts if $zid \in \mathsf{zidSet}$, otherwise it adds zid to zidSet and sends random t-bit value R_D to C.
3) C receives R_D, computes $\mathsf{checksum}_C \leftarrow R_C \oplus R_D$, sends (R_C, d) to D, and inputs $\mathsf{checksum}_C$ into the C-to-D SAS channel.
4) D computes $\mathsf{checksum}_D \leftarrow R_C \oplus R_D$ and upon receiving $\mathsf{checksum}_C$ on the C-to-D SAS channel,
   it checks if $\mathsf{checksum}_C = \mathsf{checksum}_D$ and $\mathsf{Com} = H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$ and aborts if not.
   D computes $b \leftarrow a^k$ and $z \leftarrow \mathsf{R}(K_z, zid)$, and sends $e_D \leftarrow \mathsf{Enc}(\mathsf{pk}, (z, b))$ to C.
5) C computes $(z, b) \leftarrow \mathsf{Dec}(\mathsf{sk}, e_D)$ and $\mathsf{rwd} \leftarrow H_{\mathsf{oprf}}(\mathsf{pwd}, b^{1/r})$, 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 messages authenticated by keys z and 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 $\perp$.
2) The final output of C and S equals their outputs in the DE-PAKE instance: either a session key K or a rejection sign $\perp$.

Fig. 4: TFA-KE Instantiation: Protocol PTFA

A. DE-PAKE protocol DEPAKE [24]

Protocol DEPAKE was proposed in [24] and shown to satisfy the DE-PAKE security definition (recalled in Appendix A-B). The DE-PAKE setting is similar to ours, involving the same set of parties $(\mathsf{U}, \mathsf{S}, \mathsf{C}, \mathsf{D})$ and intended to satisfy strong security against a full man-in-the-middle attacker that can corrupt servers and devices but, in contrast to our model, does not guarantee any security upon password leakage and/or client compromise. Indeed, the DEPAKE protocol breaks upon the user's password disclosure. The PTFA protocol presented here arms DEPAKE against such vulnerability.

Protocol DEPAKE builds on two primitives: An asymmetric PAKE protocol aPAKE run between server and client (on behalf of a user), and a PTR (Password-to-Random) protocol PTR run between client C and device D that translates, using a key stored at D, a user's *master password* into random independent passwords for each user account. *Asymmetric PAKE* (aPAKE) refers to a password protocol secure against server compromise [20], [23], [24], namely, one where the server stores a one-way function of the user's password such that an attacker that breaks into a server cannot learn the user's password without running an offline dictionary attack. We note that protocol DEPAKE accepts any aPAKE including those that build on PKI (i.e., assume the client has a server's public key), e.g., the standard password-over-TLS scheme, or those that are independent of PKI (no key other than the user's password is needed). Protocol DEPAKE uses these two components as follows: First, a session between C (on behalf of U) and S is established by C, then C runs the PTR protocol with D on U's password pwd, and third, C runs an aPAKE session with S using the output rwd of PTR as her password.

PTR *protocol [24].* Here we describe the specific PTR protocol instantiated in [24] which is based on the Ford-Kaliski's Blind Hashed Diffie-Hellman technique [19] (sometimes referred to as an *Oblivious PRF (OPRF)*). First, we define a keyed function F using the following elements: A group G of prime order q, a hash function H_{grp} that maps arbitrary strings into elements of G and hash function H_{oprf} mapping arbitrary strings into $\{0,1\}^\kappa$ (where κ is a security parameter). We define $F_k(x) = H_{\mathsf{oprf}}(x, (H_{\mathsf{grp}}(x))^k)$ where the key k is chosen at random in $\mathbb{Z}_q$. In PTR this function is computed jointly between D and C where D inputs key k and C inputs the user's password pwd as the x value, and the output, denoted rwd, is learned by C only. Specifically, the computation of $F_k(\mathsf{pwd})$ proceeds by C sending $a = H_{\mathsf{grp}}(\mathsf{pwd})^r$ where C chooses r at random in $\mathbb{Z}_q$, D responding with $b = a^k$, and C computing $\mathsf{rwd} = b^{1/r}$. In this way, D learns nothing about pwd and no one (C or an attacker) can distinguish from random the result of F_k on inputs they did not query from D (this properties are proved in [24] based on the One-More (Gap) Diffie-Hellman (OM-DH) assumption over G in the ROM model).

B. *The* PTFA *protocol*

Our main TFA-KE instantiation, PTFA, uses as component any asymmetric PAKE protocol aPAKE, the above protocol PTR, in addition to the following standard cryptographic primitives: A pseudorandom function R used to map zid challenges into z outputs under a key K_{DS} shared between D and S; a semantically-secure public key encryption scheme $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$; an unauthenticated key exchange protocol, denoted uKE, run between C and S to generate shared session-specific keys K_{CS} that are used to authenticate all subsequent session traffic between C and S (uKE can be as simple as a plain unauthenticated Diffie-Hellman or a standardized protocol like TLS if running on a PKI setting); an *Authenticated Channel (AC)* scheme, defined by a pair ACSend, ACRec, which implement bi-directional authenticated communication between two parties sharing a symmetric key K [17], [21]. We assume that the AC scheme is stateful and that it provides authenticity and protection against replay. The sender algorithm ACSend takes inputs key K and message m, and outputs m with additional authentication information computed with key K (e.g., using a MAC function). The receiver procedure, $\mathsf{ACRec}(K, c)$ outputs either a message or the rejection symbol $\perp$.

We are now ready to define PTFA as described in Fig. 4. The protocol is a composition of protocol TFA-PAKE from Figure 3 with the DEPAKE protocol of [24] and its two components aPAKE and PTR described in Section VI-A. ("PTFA" is a mnemonic contraction of PTR and TFA.) Protocol aPAKE is used as the PAKE component of TFA-PAKE (shown as the last C-S interaction in Fig. 3) while the PTR protocol is run between C and D over the SAS-protected communication between these parties. Specifically, the value a from PTR is sent together with the value zid in the flow from C to D, and the value b is sent from D to C together with z (the latter is encrypted by D under a public key that C sends also with zid). Also, note the management of values zid via the set zidSet in Fig. 4 which implements the requirement, that we discussed at the end of Section V, that D never responds twice to the same zid (except for possible re-transmission upon network errors; see Section VIII-B). The PTFA protocol with all its details is shown in Figure 4 (see also the illustration in Fig. 6 in Section VIII).

C. *TFA-KE Security of Protocol* PTFA

In Section VII we present a generalization of PTFA that works with *any* DE-PAKE protocol, not necessarily the specific DEPAKE protocol from [24], and prove it to be secure under the security definition from Section IV. As a corollary we obtain a proof that also PTFA satisfies this very strong definition of TFA-KE security.

The intuition for the protocol's security and the motivation for its design has been covered to a large degree in discussing the TFA-PAKE protocol of Section V. However, we note that the addition of the PTR protocol is essential for achieving the stringent bounds of our definition, particularly upon server compromise. Indeed, note that our bounds require that even in case of server compromise the attacker should have only $q_D/2^{d+t}$ probability of impersonation (for simplicity we illustrate the case $q'_C = 0$). However, how can one enforce the 2^t factor when the attacker learns the key K_{DS} shared between S and D upon server compromise and therefore can compute the z values? The answer is that PTFA runs the PTR protocol under SAS protection which forces the attacker to run (expected) 2^t sessions to be able to inject its own PTR value a over that channel, and such injection is necessary for testing a guess for a user's password even when K_{DS} is known. When considering a password dictionary of size 2^d this ensures the denominator 2^{d+t} in the security bound.

VII. Cryptographic Analysis

Here we prove that protocol PTFA satisfies the strong security definitions from Section IV. We do so by proving the security of a generic protocol TFAgen, presented in Figure 5, which generalizes PTFA, hence allowing for further instantiations. In particular, protocol TFAgen compiles *any* secure DE-PAKE and aPAKE schemes into a secure TFA scheme. In addition to the building blocks SAS, uKE, AC, DE-PAKE, and PKE introduced in Sections III, V, and VI, protocol TFAgen also uses a generic symmetric Key Encapsulation Mechanism (KEM), denoted $(\mathsf{KemE}, \mathsf{KemD})$ (see e.g. [30]), which 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 $z \leftarrow \mathsf{KemD}(K_z, zid)$. A KEM is secure if key z corresponding to $zid \notin \{zid_1, ..., zid_q\}$ is pseudorandom even given the keys z_i corresponding to all zid_i's. In protocol PTFA of Figure 4 such KEM is implemented using a PRF R: zid is a random κ-bit string and $z = R(K_z, zid)$.

Theorem 1. *Assuming security of the building blocks DE-PAKE, SAS, uKE, PKE, KEM, and AC, protocol* TFAgen *is a* (T, ϵ)*-secure TFA-KE scheme for* ϵ *upper bounded by*

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

for $n = q_{HbC} + \max(q_S, q_D, q_C, q'_C)$ *where* q_{HbC} *denotes the number of* TFAgen *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 the protocol building block* P.

The proof of the theorem is lengthy and it is included in Appendix B. As a corollary, we obtain the security claim for protocol PTFA in Fig. 4, under standard assumptions in ROM.

Corollary 2. *Assuming that* aPAKE *is a secure asymmetric PAKE,* uKE *is secure Key Exchange,* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *is a semantically-secure PKE,* R *is a secure PRF, and* $(\mathsf{ACSend}, \mathsf{ACRec})$ *is a secure Authenticated Channel scheme, protocol* PTFA *is a secure TFA-KE scheme under the OM-DH assumption in ROM.*

VIII. System Development & Testing

A. *Protocol Instantiation*

We first summarize the concrete instantiation of the protocol that we implemented and evaluated as part of our TFA system. In this protocol, the user U, who owns a smartphone D, tries

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

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

1) S and C run a (non-authenticated) key exchange uKE to establish a key K_{CS} between them;
2) S generates $(zid, z) \leftarrow \mathsf{KemE}(K_z)$, computes $e_S \leftarrow \mathsf{ACSend}(K_{CS}, zid)$, 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 + **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 a SAS scheme to authenticate $\mathsf{M_C}$ using the t-bit C-to-D SAS channel.
2) D aborts if $zid \in \mathsf{zidSet}$ 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 the DE-PAKE protocol 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. 5: Generic TFA-KE Scheme: Protocol TFAgen

to log in to a webserver S through a client terminal C. The implemented protocol follows the specification from Figure 4.

The parties involved in the protocol run a one time initialization phase. During initialization D is initialized with: 1) k, the random key for the PTR function to generate the randomized password from the user's master password pwd, 2) K_z, the random secret for the PRF function R shared between S and D to generate the one time code z, 3) zidSet, an empty list of previously received nonce, and 4) $History_D$, an empty history of recent transaction with C. C is initialized with $History_C$, an empty history of recent transaction with D. The server S is initialized with K_z and σ, the user-specific PAKE state, typically a one-way mapping of the user's site-specific password rwd (e.g., $H(\mathsf{rwd}, Salt)$).

In the authentication phase, U authenticates to S using the password rwd and the OTC z. During this phase, rwd is reconstructed by running an PTR protocol between C and D, on input of pwd provided by U and k coming from D. z is generated at D and S from a nonce zid (picked and transferred by S to D via C). z is transferred from D to C encrypted under a client's public key pk where pk is transmitted from C to D over the authenticated SAS channel.

Finally, to accomplish the second-factor authentication, C proves knowledge of z by binding it (via authentication) to the PAKE protocol messages run between C and S. In our implementation κ (i.e., the security parameter) is 128 bits.

Figure 6 shows the high level view of the steps taken by each party in the protocol. The PTFA authentication protocol starts by running a key exchange protocol (e.g., TLS) between C and S in the initial stage, upon opening the website . Next S sends the nonce zid to D via C. This actions can be triggered by entering the username in the website. Freshness of the nonce zid is verified by D by comparing it against the list of recently received nonce (zidSet). A repeated nonce is accepted only if it is re-sent as the result of restarting an interrupted transaction between D and C (e.g., due to a network failure). D detects such situation through $History_D$ and client resends the interrupted messages as recorded in $History_C$ (e.g., zid is stored in $History_C$)

After receiving the first message (zid, pk), D and C run SAS protocol (e.g., [33]), which results in a short string *checksum* in C and D. To ensure that the SAS protocol is not prone to the man-in-the-middle attack, U is required to validate the two checksum strings being produced on D and C. Only after the checksum is validated and the channel gets authenticated, C and D complete the PTR protocol to generate $\mathsf{rwd} = F_k(\mathsf{pwd})$, and D sends $z = \mathsf{R}_{K_z}(zid)$ to C over the resulting channel secured by the SAS protocol.

Finally, S and C run the PAKE protocol on input rwd and σ (e.g., in the PKI model C bundles rwd and z together and send to S over TLS; S can then authenticate the user by matching the password and OTC against σ and $\mathsf{R}_{K_z}(zid)$).

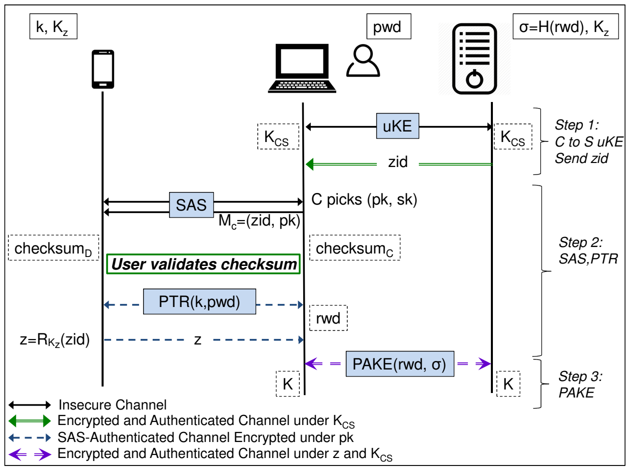

Fig. 6: High-level overview of the implemented instantiation of PTFA protocol.

B. Implementation of the Protocol

1) Parties and Applications: In our TFA system implementation, the webserver S is a Virtual Machine running Debian

8.0 with 2 Intel Xeon 3.20GHz and 3.87GB of memory. Client terminal C is a MacBook Air with 1.3GHz Intel Core i5 and 4GB 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 100Mbps 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 is developed in Java for Android phones.

2) Key Components: Here we define the main components of the implementation:

PTR Function: The PTR functions runs between C and D to blind the password. We implemented the PTR function as $F_k(\mathsf{pwd}) = H_{\mathsf{oprf}}(\mathsf{pwd}|domain, H_{\mathsf{grp}}(\mathsf{pwd}|domain)^k)$ where the algebraic operations are performed over the NIST P-256 elliptic curve, k is the PTR key chosen as a random value in Z_q (q is the order of the curve) and H_{grp} hashes strings into this curve (see below). To run the PTR function, the extension picks a random number $\rho \in Z_q$ and raises the hash value (H_{grp}) of the input to the power ρ, and sends it to D (we call this value a). In response, the extension receives $b = a^k$ from D only after D confirms the group membership of a. The rest of the computation happens on the client-side. In particular, C checks the group membership of b, reconstructs the randomized password by raising the received value b to the power of $\rho^{-1} \in Z_q$ and then computes SHA-256 hash of the calculated value. ρ is stored in $History_C$ to be reused in case the transaction between C and D needs to be repeated.

We implemented the Elliptic Curve point computation using SJCL library on the client. We developed the cryptographic function on the smart phone using SpongyCastle and the Java Security API.

Hash-into-Elliptic-Curve: The above-described PTR uses a hash function H_{grp} that maps arbitrary strings into points in the PTR's elliptic curve. Once the user types her password pwd, the extension receive pwd and inputs it to the H_{grp} function. To add resistance against phishing attacks, password is concatenated with the domain name of the website and the concatenated string is input into H_{grp}. The implementation of H_{grp} is in line with [24], although we used different set of libraries (SJCL).

SAS Protocol: We implemented the SAS protocol introduced in [33]. Figure 2 depicts the SAS protocol as implemented in our system. To authenticate $\mathsf{M_C} = (\mathsf{pk}, zid)$, in the first round of the protocol, C generates a random 128-bit number R_C and sends a commitment $C = \mathsf{Com}(\mathsf{M_C}, R_C, d)$ to D. d is a random 256-bit number used as the decommitment value, and R_C is used in the generation of a checksum to be verified by the user. After receiving C, D picks a random number R_D and sends it to C in the second round of the protocol. In the last round, C sends R_C and the decommitment value d to D. To protect against a man-in-the-middle (MitM) attacker on the protocol, both C and D generate a checksum $= R_C \oplus R_D$, which should match at both sides, as an indication of "MitM attack-free" session. We will describe different mechanisms to realize the checksum matching task later in Section VIII-C. R_C, and d are stored in $History_C$.

One Time Code Generation: The OTC z is generated as a function of the nonce zid on the device, and is verified on the server. zid is picked by S and transferred to D via C (C stores zid in $History_C$). We implemented the OTC generation function by computing the keyed-hash message authentication code (HMAC) of the nonce. Our implementation of HMAC is based on SHA-256 with the key K_z shared between S and D during the initialization phase. In traditional TFA approaches, the OTC value is usually truncated and encoded to a human-readable 6 digit code [13]. However, since in our protocol, the OTC is transferred on the automated channels and human user is not involved in transferring it, we can use the result of SHA-256 as the OTC without truncating it. The OTC generator function is adopted from Google Authenticator one-time passcode generator for Android developed in Java using Java Security library [4] and the OTC verification at the server-side is developed in PHP.

PAKE Protocol: In the PKI model, the communication between S and C takes place over an encrypted TLS channel. At the final stage of the protocol, C bundles the OTC z and the password rwd together and sends them to S over the TLS channel, and S then verifies these values and authenticates the user.

Our PKI-free PAKE model is in line with [24], adapted from the threshold PAKE (TPAKE) protocol of [22], [23]. The PAKE protocol also involves a key exchange adapted from [26].

We developed the PKI-free PAKE protocol on the Chrome browser extension using CryptoJS library, and we designed the server as a Java application using Java Security and BouncyCastle API. The server application resides on the same server as the web-server and communicates with it internally.

3) Communication Channels: In our protocol, C and S communicate over the Internet. The server identifies U by her username, and retrieves the state of U stored on the server (σ) to run the protocol. The state of the user includes $H_{\mathsf{oprf}}(\mathsf{rwd})$, PRF key ($K_z$), and PAKE parameters. C identifies the server by the IP address or domain name as is the case in locating any website. Except the checksum comparison task, C and D communicate over a bidirectional Google Cloud Messaging (GCM) [5], in which D acts as the GCM server and C acts as the GCM client. The channel gets authenticated and secured after running the SAS protocol. The communication between D and C goes over an Elliptic Curve Integrated Encryption Scheme (ECIES) encryption under C's one-time public key pk. We developed the ECIES key generation and decryption for the Chrome extension in JavaScript and developed the ECIES encryption mechanism for Android using SpoungyCastle library. The communication between D and S is proxied via C.

C. Checksum Validation Design

An essential component of the SAS protocol (described above; see Figure 2) is the verification of the checksum.

TABLE I: The average execution time of each of the components of the PTFA protocol and the total execution time.

Protocol	**Purpose**	**Parties**	**Average Time (ms)** (std. dev.)
SAS (excluding 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	Password Authenticated Key Exchange	C and S	182.27 (3.67)
– PTR			60.67 (1.65)
– Key Exchange			106.48 (4.09)
PKI PKAE (TLS)	Encrypt C-S Communication	C and S	32.54 (1.38)
OTC Generation and Verification	Second Factor Authentication	D and S	0.29 (0.00)
Overall in PKI-free Model		C, D and S	**410.77 ms**
Overall in PKI Model		C, D and S	**263.27 ms**

Since the protocol is run over an insecure channel, a man-in-the-middle (MitM) attacker might alter the messages passed between C and D. The SAS protocol generates a checksum of the protocol on D and C, and the two checksums need to be compared in order to detect the MitM attack (non matching checksum values indicate the presence of the attack). The checksums should be validated over an out-of-band channel, since the attacker has full control of the D-C channel (i.e., the checksums cannot be transferred on the same data channel as other messages of the SAS protocol).

A common approach for checksum validation is called "Compare-Confirm", in which the user compares the checksums displayed on D and C, and taps the Confirm button on D and/or C in case the two match [32]. Even though such comparison suffices for our purposes, this approach gets compromised when a neglectful or rushing user presses the confirm button without comparing the checksum strings.

Another approach for checksum validation is "Copy-Confirm" [32]. Here, the user enters the checksum displayed on C ($\mathsf{checksum}_C$) into D, and D confirms if the entered value matches with its own checksum value ($\mathsf{checksum}_D$), and, only if the two match, proceeds with the protocol (e.g., submits OTC over the authenticated channel). This approach requires the user to type the checksum on D.

However, as per the previous studies [28], users might make mistakes in entering the checksum, which would eventually degrade usability and security.

Although security of the SAS protocol benefits from the use of long checksums, in both of the above approaches, the checksum values can not be more than 4-6 digits due to the manual burden imposed on the user. Even when the checksums are short, users may still make mistakes in comparing or transferring them.

In our design, we show how we can improve upon the security of the above approaches by using longer checksums through the use of QR-based and voice-based approaches. Our designs make the transfer of checksum values relatively easy, without relying upon the user to make the checksum comparison decision. 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. In case QR-based or voice-based approach is not feasible to be used (e.g., missing camera or noisy environments), we still offer the option to use the manual Copy-Confirm approach.

1) Manual Checksum Validation: In the manual checksum Copy-Confirm approach, we map the checksum into a 6 digit number (i.e., 19 bits) by truncating 19 bits of the $checksum = R_D \oplus R_C$ starting from an offset defined by the last byte of the checksum. We then display it as a 6 digit number in a browser alert box and ask the user to enter it in her smartphone app. We mapped the checksum to the 6 digit number in the browser extension using the SJCL library and on the device using Java Security library.

2) QR Code Checksum Validation: In this checksum validation model, we encode the full, 256-bit checksum, 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 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. In this setting, the user does not need to enter the checksum but only needs to hold her phone and capture a picture of the browser's screen. The larger checksum (i.e., larger value of t) in this case leads to a major security improvement: It reduces the attacker's advantage from Definition 1 to negligible values (making the corresponding attacks completely infeasible) except in the case of device compromise.

3) Voice-based Checksum Validation: With the voice-based checksum validation approach, same as the manual checksum entry, we encode the checksum in 6 digits and display it on the client terminal. However, rather than entering the checksum in the mobile device or capturing the QR code, we ask the user to speak the checksum that is displayed on the terminal into her smart phone. The smart phone receives this audio, recognizes and transcribes it using a speech recognition tool, then compares it with the $\mathsf{checksum}_D$ and proceeds with the protocol if the match validates.

The developed code for our Chrome extension is similar to the manual checksum validation approach. On the device, however, we developed a transcriber application using Android.Speech API. The user clicks on a "Speak" button we added to the app and speaks out loud the 6-digit number. The transcriber application recognizes the speech and convert it to the text that can be compared against $\mathsf{checksum}_C$

To further improve the usability of this approach, we 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.

D. Performance Evaluation

To evaluate the feasibility of our PTFA protocol, we measured its computational performance. The computational time of the different cryptographic components was calculated over 10,000 iterations, under the settings described earlier in this section (Table I provides the results). The overall computational time of the protocol (excluding the manual checksum validation) is about 410ms in the PKI-Free PAKE model and about 260ms in the PKI Model. None of the protocol components took more than 182ms to execute. This demonstrates that the protocol and all its cryptographic subcomponents are computationally-efficient.

Since the checksum validation protocol is assisted by the human user, we report on 30 iterations performed by one experimenter. The time taken by manual checksum validation was 8.50s on average (standard deviation 2.84s). The time taken by QR-Coded validation was 4.87s on average for capturing the code (standard deviation 1.32s) and 0.02s on average for decoding the code (standard deviation 0.00s). The time taken by audio-based validation was 4.08s on average for speaking the checksum (standard deviation 0.34s) and 1.18s on average for transcribing the spoken checksum (standard deviation 0.42s). The average time for these tasks may vary between different users. The time taken by the device to perform the comparison of $\mathsf{checksum}_D$ and $\mathsf{checksum}_C$ is negligible. In our preliminary testing of these two channels shows virtually-0 error rate. Both of the approaches are compatible with any browser with HTML5 support.

As expected, the manual checksum entry took longer than the QR-Code and Audio-based validation, which justifies the need for these new models we had introduced. The QR-code and Audio-based validation each takes about the same time (4-5 seconds) to complete. Both seem efficient and could in fact complement each other, depending upon the type of surrounding environment at the time of login. In case neither of the two are workable (e.g., loud environment and broken phone camera), it is always possible to fall-back to the manual approach (which still takes less than 10s). While a formal usability study might be needed to evaluate the checksum models in the future, our experiments serve to establish their promising feasibility. Given that traditional two-factor authentication schemes (e.g., Google's two-step verification system) may take about 25s to login on average [25], our TFA protocol and models could offer a significant improvement in efficiency, while relaxing the requirements imposed on the human-assisted channel.

IX. Conclusion

In this paper, we designed a TFA system that can offer end-to-end security by protecting against a powerful attacker that controls all communication channels between parties (i.e., in the sense of a full active man-in-the-middle) and can compromise parties at will. In particular, protection is provided upon server compromise, device compromise and password leakage (e.g., learned upon a client compromise). Our system leverages on the notion of short authenticated strings from [33] to add TFA security against eavesdropping and man-in-the-middle attacks on the channel between the TFA device and the client machine. We formulated a rigorous security model for this system and presented a protocol that provably satisfies the strong security requirements set by this model. We also prototyped a TFA implementation of this system based on device to client channels that require reduced user involvement compared to traditionally deployed TFA systems today.

References

[1] "Anonymous hackers claim to leak 28,000 PayPal passwords on global protest day," Available at: http://goo.gl/oPv2h.
[2] "Blizzard servers hacked; emails, hashed passwords stolen," Available at: http://goo.gl/OTNWJC.
[3] "FIDO Universal 2nd Factor," https://www.yubico.com/.
[4] "Google Authenticator Android app," Available at: https://github.com/google/google-authenticator-android/.
[5] "Google Cloud Messaging," Available at: https://developers.google.com/cloud-messaging/.
[6] "Hack Brief: Yahoo Breach Hits Half a Billion Users," Available at: https://www.wired.com/2016/09/hack-brief-yahoo-looks-set-confirm-big-old-data-breach/.
[7] "Hackers compromised nearly 5M Gmail passwords," Available at: http://goo.gl/IRu07u.
[8] "LinkedIn Confirms Account Passwords Hacked," Available at: http://goo.gl/UBWuY0.
[9] "RFC 4226 HOTP: An HMAC-based One-Time Password Algorithm," Available at: https://www.ietf.org/rfc/rfc4226.txt.
[10] "RSA breach leaks data for hacking securid tokens," Available at: http://goo.gl/tcEoS.
[11] "RSA SecurID software token cloning: a new how-to," Available at: http://goo.gl/qkSFY.
[12] "Russian Hackers Amass Over a Billion Internet Passwords," Available at: http://goo.gl/aXzqj8.
[13] "TOTP: Time-Based One-Time Password Algorithm," Available at: https://tools.ietf.org/html/rfc6238.
[14] T. Acar, M. Belenkiy, and A. Küpçü, "Single password authentication," *Computer Networks*, vol. 57, no. 13, 2013.
[15] Authy, "Two-factor authentication - authy," available online on https://www.authy.com/.
[16] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology – Eurocrypt*, 2000.
[17] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 2001, pp. 453–474.
[18] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz, "Strengthening user authentication through opportunistic cryptographic identity assertions," in *Proceedings of ACM conference on Computer and communications security*. ACM, 2012.
[19] W. Ford and B. S. K. Jr., "Server-assisted generation of a strong secret from a password," in *WETICE*, 2000, pp. 176–180.
[20] C. Gentry, P. MacKenzie, and Z. Ramzan, "A method for making password-based key exchange resilient to server compromise," in *Advances in Cryptology-CRYPTO*, 2006.
[21] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *CRYPTO*, 2012, pp. 273–293, also Cryptology ePrint Archive, Report 2011/219.
[22] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and t-pake in the password-only model," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 233–253.
[23] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "Highly Efficient and Composable Password-Protected Secret Sharing," in *1st IEEE European Symposium on Security and Privacy (EuroS&P)*, 2015.

[24] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Device-enhanced password protocols with optimal online-offline protection," 2015, to appear at ASIACCS 2016.

[25] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: usable two-factor authentication based on ambient sound," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[26] H. Krawczyk, "HMQV: A high-performance secure diffie-hellman protocol," in *Annual International Cryptology Conference*, 2005, pp. 546–566.

[27] C.-C. Lin, H. Li, X.-y. Zhou, and X. Wang, "Screenmilker: How to milk your android screen for secrets." in *Network & Distributed System Security Symposium*, 2014.

[28] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Secure device pairing based on a visual channel," in *Security and Privacy, 2006 IEEE Symposium on*, 2006.

[29] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan, "Two-factor authentication resilient to server compromise using mix-bandwidth devices," in *Network & Distributed System Security Symposium*, 2014.

[30] V. Shoup, "ISO 18033-2: An emerging standard for public-key encryption," http://shoup.net/iso/std6.pdf, Dec. 2004, final Committee Draft.

[31] B. Shrestha, M. Shirvanian, P. Shrestha, and N. Saxena, "The sounds of the phones: Dangers of zero-effort second factor login based on ambient audio." in *Conference on Computer and Communications Security*, 2016.

[32] E. Uzun, K. Karvonen, and N. Asokan, "Usability analysis of secure pairing methods," in *Financial Cryptography and Data Security*, 2007.

[33] S. Vaudenay, "Secure communications over insecure channels based on short authenticated strings," in *Advances in Cryptology - CRYPTO*, ser. Lecture Notes in Computer Science, no. 3621. Springer Verlag, 2005, pp. 309 – 326.

Appendix A
PAKE and DEPAKE Security Model

We recall the *Device-Enhanced PAKE (DE-PAKE)* security model from [24] that extends the regular PAKE model (also reviewed below) for the case that the password protocol is aided by a device, and which we use as a basis for our TFA model.

A. PAKE Security Model [16]

Protocol participants. There are two types PAKE protocol participants, users and servers. Each user U is associated with a unique server S while servers may be associated with multiple users.

Protocol execution. A PAKE protocol has two phases: initialization and key exchange. In the initialization phase each user U chooses a random password pwd from a given dictionary Dict and interacts with its associated server S producing a user's state $\sigma_{\mathsf{S}}(\mathsf{U})$ that S stores while U only remembers its password pwd. *Initialization is assumed to be executed securely, e.g., over secure channels.* In the key exchange phase, users interact with servers over insecure (adversary-controlled) channels to establish session keys. Both users and servers may execute the protocol multiple times in a concurrent fashion. Each execution of the PAKE protocol by U or S defines a (user or server) protocol *instance*, also referred to as a protocol *session*, denoted respectively Π_i^{U} or Π_i^{S}, where integer pointer i serves to differentiates between multiple protocol instances executed by the same party. Each protocol session is associated with the following variables: a *session identifier* sid, which we equate with the message transcript observed by this instance (where both U and S order their interaction transcripts starting with U's message), a *peer identity* pid, and a *session key* sk. For a user instance the peer is always the user's server while for a server instance the peer is the user authenticated in the session. The output of an execution consists of the above three variables which can be set to $\perp$ if the party aborts the session (e.g., when authentication fails, a misformed message is received, etc.). When a session outputs $\mathsf{sk} \neq \perp$ we say that the session *accepts*.

PAKE Security. To define security we consider a probabilistic attacker A which schedules all actions in the protocol and controls all communication channels with full ability to transport, modify, inject, delay or drop messages. In addition, the attacker knows (or even chooses) the dictionaries used by users. The model defines the following queries or activations through which the adversary interacts with, and learns information from, the protocol's participants.

$\mathsf{send}(P, i, P', M)$: Delivers message M to instance Π_i^P purportedly coming from P'. In response to a send query the instance takes the actions specified by the protocol and outputs a message given to A. When a session accepts, a message indicating acceptance is given to A. A send message with a new value i (possibly with null M) creates a new instance at P with pid P'. For simplicity, we assume that the pair $\{P, P'\}$ in any send message contains a user and the server associated to that user (a non-compliant message causes the receiving instance to abort). The send query can also create a new instance of party P: If Π_i^{U} does not exist then query $\mathsf{send}(\mathsf{U}, i, \mathsf{S}, \mathsf{init})$ creates a new instance Π_i^{U} which executes with $\mathsf{pid} = \mathsf{S}$ on U's chosen password pwd. Similarly, if Π_i^{S} does not exist then $\mathsf{send}(\mathsf{S}, i, \mathsf{U}, M)$ creates a new instance Π_i^{S} which executes with $\mathsf{pid} = \mathsf{U}$ on S's input $\sigma_{\mathsf{S}}(\mathsf{U})$, with U's first message set to M. (This formalism assumes that protocol exchanges are initiated by users, which is the operational setting in PAKE.)

$\mathsf{reveal}(P, i)$: If instance Π_i^P has accepted, outputs the respective session key sk; otherwise outputs $\perp$.

$\mathsf{corrupt}(P)$: Outputs all data held by party P and A gains full control of P. We say that P is *corrupted*.

$\mathsf{compromise}(\mathsf{S}, \mathsf{U})$: Outputs state $\sigma_{\mathsf{S}}(\mathsf{U})$ at S. We say that S is U-*compromised.*

$\mathsf{test}(P, i)$: If instance Π_i^P has accepted, this query causes Π_i^P to flip a random bit b. If $b = 1$ the instance's session key sk is output and if $b = 0$ a string drawn uniformly from the space of session keys is output. A test query may be asked at any time during the execution of the protocol, but may only be asked once. We will refer to the party P against which a test query was issued and to its peer as the *target parties*.

The following notion taken from [22] is used in the security definition below to ensure that legitimate messages exchanged between honest parties do not help the attacker in online password guessing attempts (only adversarially-generated messages count towards such online attacks). It has similar motivation as the execute query in [16], but the latter fails to capture the ability of the attacker to delay and interleave messages from different sessions.

Rogue send *queries/activations:* We say that a $\mathsf{send}(P, i, P', M)$ query is *rogue* if it was not generated and/or delivered according to the specification of the protocol, i.e. message M has been changed or injected by the attacker, or the delivery order differs from what is stipulated by the protocol (delaying message delivery or interleaving messages from different sessions is not considered a rogue operation as long as internal session ordering is preserved). We also consider as rogue any $\mathsf{send}(P, i, P', M)$ query where P is uncorrupted and P' is corrupted. We refer to messages delivered through rogue send queries as *rogue activations* by A.

Matching sessions. A session in instance Π_i^P and a session in instance $\Pi_j^{P'}$ are said to be *matching* if both have the same session identifier sid (i.e., their transcripts match), the first has $\mathsf{pid} = P'$, the second has $\mathsf{pid} = P$, and both have accepted.

Fresh sessions. A session at instance Π_i^P with peer P' s.t. $\{P, P'\} = \{\mathsf{U}, \mathsf{S}\}$ is called *fresh* if none of the queries $\mathsf{corrupt}(\mathsf{U})$, $\mathsf{corrupt}(\mathsf{S})$, $\mathsf{compromise}(\mathsf{S}, \mathsf{U})$, $\mathsf{reveal}(P, i)$ or $\mathsf{reveal}(P', i')$ were issued, where $\Pi_{i'}^{P'}$ is an instance whose session matches Π_i^P (if such $\Pi_{i'}^{P'}$ exists).

Correctness. Matching sessions between uncorrupted peers output the same session key.

Attacker's advantage. Let PAKE be a PAKE protocol and A be an attacker with the above capabilities running against PAKE. Assume that A issues a single test query against a fresh session at a user or server and ends its run with an output bit b'. We say that A *wins* if $b' = b$ where b is the bit chosen internally by the test session. The *advantage of* A *against* PAKE is defined as $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{PAKE}} = 2 \cdot Pr\,[\mathsf{A} \text{ wins against } \mathsf{PAKE}] - 1$.

Definition 2. *A PAKE protocol* PAKE *is* (q_S, q_U, T, ϵ)-secure *if it is correct and for any password dictionary* Dict *and any attacker* A *that runs in time T, it holds that* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{PAKE}} \leq \frac{q_U + q_S}{|\mathsf{Dict}|} + \epsilon$ *where q_U is the number of rogue* send *queries having the target user* U *as recipient and q_S is the number of rogue* send *queries having the target* S *as recipient.*

Dictionary size 2^d. Our treatment works for any dictionary size, but for notational convenience we denote it as 2^d.

B. DE-PAKE Security Model [24]

We present the extension of the PAKE model to the DE-PAKE setting. Besides servers and users in the PAKE model, each user is associated with a device D with which it communicates over a two-way link. (We stress that the role of D can be played by any data-connected entity, including a hand-held device or an auxiliary web service.) The initialization phase of PAKE is extended to include the user-device communication that establishes the state stored at D. As before, users only remember their passwords. As in the PAKE case, initialization (including the user-device interaction) is assumed to run over secure channels. After initialization, the links between users and devices are subject to the same man-in-the-middle adversarial activity as in the links between users and servers. Device instances Π_i^{D} are created similarly to user and server instances, and are activated by A via send queries that include users and devices as senders and receivers. However, device instances do not produce output other than the outgoing messages. In particular, reveal queries do not apply to them, but corrupt queries can be issued against devices, in which case the internal state of the device is revealed to A who then controls the device. The session-related notions, including the test query, do not apply to devices.

The attacker's goal is the same as before, i.e. to win the test experiment at a user or server instance, as in the PAKE setting. Also the correctness property is unchanged. However, to the attacker resources we add the number of *rogue* send queries (see Section A-A) where the target user is the recipient and the device the sender (denoted q'_U) and the number of *rogue* send queries where the target user is the sender and the device the recipient (denoted q_D). We refer to this more powerful adversary as a DE-PAKE attacker.

Strong KCI resistance. The DE-PAKE model is intended to provide a much stronger notion of security in case of server compromise than achievable in the PAKE case. While in the latter, impersonating U to S in case of U-compromise is possible (and unavoidable) through an offline dictionary attack, in DE-PAKE protocols this is prohibited. In order to formalize this requirement we follow the treatment of KCI resistance from [26] and we strengthen the capabilities of a DE-PAKE attacker through a more liberal notion of fresh sessions at a server S. All sessions considered *fresh* in the PAKE model are also considered fresh in the DE-PAKE model; in addition, in the DE-PAKE model, a session Π_i^{S} at server S with peer U is considered fresh *even if* $\mathsf{corrupt}(\mathsf{S})$ *or* $\mathsf{compromise}(\mathsf{S}, \mathsf{U})$ *were issued* as long as all other requirements for freshness are satisfied and *the attacker* A *does not have access to the temporary state information created by session* Π_i^{S}. This relaxation of the notion of freshness captures the case where the attacker A might have corrupted S and gained access to S's secrets (including long-term ones), yet A is not actively controlling S during the generation of session Π_i^{S}. In this case we would still want to prevent A from authenticating as U to S on that session. Definition 3 (item 2) below ensures that this is the case for DE-PAKE secure protocols even when *unbounded* offline attacks against S are allowed.

The following security definition captures the maximal-attainable online and offline security from a DE-PAKE protocol as informally discussed in the introduction. Let DEPAKE be a DE-PAKE protocol and A be an attacker with the above capabilities running against DEPAKE. As in the PAKE model, we assume that A issues a single test query against some U or S session, that A output bit b', and we say that A wins if $b' = b$ where b is the bit chosen by the test session. We define $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} = 2 \cdot Pr\,[\mathsf{A} \text{ wins against } \mathsf{DEPAKE}] - 1$.

Definition 3. *A DE-PAKE protocol is called* $(q_S, q_U, q'_U, q_D, T, \epsilon)$*-secure if it is correct, and for any password dictionary* Dict *of size 2^d and any attacker that runs in time T, the following properties hold (for q_S, q_U, q'_U, q_D as defined above):*

1) *If* S *and* D *are uncorrupted, the following bound holds:*

$$\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq \frac{\min\{q_U + q_S, q'_U + q_D\}}{2^d} + \epsilon. \qquad (1)$$

2) *If* D *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq (q_U + q_S)/2^d + \epsilon$.

3) If S *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq (q'_U + q_D)/2^d + \epsilon$.

4) When both D *and* S *are corrupted, expression (1) holds but* q_D *and* q_S *are replaced by the number of offline operations performed based on* D*'s and* S*'s state, respectively.*

Notation change: In the adaptation of the DE-PAKE model to TFA-KE (Section IV) we changed the notation q_U above to q_C. Indeed, in the context of TFA-KE talking about interactions of the attacker with the client C is more accurate than with the user U.

APPENDIX B
PROOF OF THEOREM 1

Security definition of SAS authentication. For the purpose of the proof below we state the security property assumed of the SAS authentication mechanism which was informally described in Section III. While [33] defines the security of SAS channels using a game-based formulation, here we do it via the following (universally composable) functionality $\mathsf{F}_{\mathsf{SAS[t]}}$: On input a message $[\mathsf{SAS.SEND}, sid, P', m]$ from an honest party P, functionality $\mathsf{F}_{\mathsf{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, P', m]$ to A, and then, if A's response is $[\mathsf{SAS.CONNECT}, sid]$, then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, m]$ to P', if A's response is $[\mathsf{SAS.ABORT}, sid]$, then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, \perp]$ to P', and if A's response is $[\mathsf{SAS.ATTACK}, sid, m']$ 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 $[\mathsf{SAS.SEND}, sid, P, m']$ to P', and if $\rho = 0$ then $\mathsf{F}_{\mathsf{SAS[t]}}$ sends fail to A and $[\mathsf{SAS.SEND}, sid, P, \perp]$ to P'.

In our main instantiatiation of the generic protocol TFAgen of Figure 5, i.e. in protocol PTFA of Figure 4, we instantiate SAS authentication with the scheme of [33], but even though the original security argument given for it in [33] used the game-based SAS security notion, it is straightforward to adopt this argument to see that this scheme securely realizes the above (universally composable) functionality.

Proof of Theorem 1 in Section VII

Proof. 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 TFAgen. Let the security advantage defined in Definition 1 for adversary A satisfy $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} = \epsilon$. Let Π_i^{S}, Π_j^{C}, Π_l^{D} refer to respectively the i-th, j-th, and l-th 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 TFAgen 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 $n_S = q_S + q_{HbC}$, $n_D = q_D + q_{HbC}$, $n_C = q_C + q'_C + q_{HbC}$, and note that these are the ranges of indexes respectively i, j, l for instances Π_i^{S}, Π_j^{C}, and Π_l^{D}. In what follows we will use $[n]$ to denote range $\{1, ..., n\}$.

The security proof will proceed 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 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, see below) 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 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 $\mathsf{test}(P, i)$ (for P equal to either S or 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's the default "network adversary" case.

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 which 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 $z_i \leftarrow Z(zid_i)$. Note that $p_1 \leq p_0 + \epsilon^{\mathsf{KEM}}(n_S) + n_S^2/2^\kappa$ where the last term follows from the fact that zid-collision implies a z-collision, and z-collision occurs in n_S random z samples with probability at most $n_S^2/2^\kappa$.

Game G_2*:* Let $\mathsf{SIM}_{\mathsf{SAS}}$ be the simulator for the SAS 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 sub-protocol, we replace this SAS execution with a simulator $\mathsf{SIM}_{\mathsf{SAS}}$ interacting with A_1 and the ideal SAS 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 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 triggers $\mathsf{SIM}_{\mathsf{SAS}}$ to start simulating to A_1 the SAS 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 $\mathsf{SIM_{SAS}}$ and $\perp$ to Π_l^{D} and Π_l^{D} aborts, and if $\rho = 1$ then $\mathsf{F_{SAS[t]}}$ sends succ to A and $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \mathsf{M_C}^*]$ to Π_l^{D}, and then Π_l^{D} proceeds to step II.2 using message $\mathsf{M_C}^*$. Since the SAS protocol realizes the UC functionality $\mathsf{F_{SAS[t]}}$ with at most error ϵ^{SAS} (per instance), and the simulator $\mathsf{SIM_{SAS}}$ 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 adverary A_2 interacts with game G_2 which internally runs interactive algorithms $\mathsf{SIM_{SAS}}$ and $\mathsf{F_{SAS[t]}}$. Note also that the $\mathsf{SIM_{SAS}}$ algorithm interacts only with $\mathsf{F_{SAS[t]}}$ on one end and A_2 on the other. We can, therefore, draw the boundaries between 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 $\mathsf{SIM_{SAS}}$, and a security game G_3 which executes the rest of game G_2, including the operation of functionality $\mathsf{F_{SAS[t]}}$. Note that G_3 does not execute the SAS protocol, but interacts with A_3 using the $\mathsf{F_{SAS[t]}}$ interface to $\mathsf{SIM_{SAS}}$, i.e. G_3 sends to A_3 messages of the type $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$, and A_3's response must be one of $[\mathsf{SAS.CONNECT}, sid]$, $[\mathsf{SAS.ABORT}, sid]$, and $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^*]$. Since we are only re-drawing the boundaries between the adversarial algorithm and the security game, we have that $p_3 = p_2$.

Game G_4*:* Let $\mathsf{A}_4 = \mathsf{A}_3$ and let G_4 be as G_3 except that for every message $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ send by G_3 for some (j, l) pair, if Adv_4 sends $[\mathsf{SAS.CONNECT}, sid]$ in response, then we make the following changes: First, the e_D value sent by Π_l^{D} is formed as $\mathsf{Enc}(\mathsf{pk}, (0^\kappa, 0^\kappa))$ instead of $\mathsf{Enc}(\mathsf{pk}, (z, K_{CD}))$ as in G_3, for pk specified in $\mathsf{M_C} = (\mathsf{pk}, zid)$. Secondly, if A_3 passes this e_D value 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 the PKE scheme $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$, it follows that $p_4 \leq p_3 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{PKE}}$.

Game G_5*:* Let $\mathsf{E_{ACbreak(CD)}}$ be an event that there is some session pair $(\Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}})$ s.t. (a) A_4 responded with $[\mathsf{SAS.CONNECT}, sid]$ to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$, 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 $\mathsf{A}_5 = \mathsf{A}_4$ and G_5 be as G_4 except that G_5 aborts if $\mathsf{E_{ACbreak(CD)}}$ 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 $\min(n_C, n_D) \cdot \epsilon^{\mathsf{AC}}$, hence $p_5 \leq p_4 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{AC}}$.

Game G_6*:* Let $\mathsf{E_{ACbreak(CD')}}$ be an event that there is some session pair $(\Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}})$ s.t. (a) A_4 responded with $[\mathsf{SAS.CONNECT}, sid]$ to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$, (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 $\mathsf{A}_6 = \mathsf{A}_5$ and G_6 be as G_5 except that G_6 aborts if $\mathsf{E_{ACbreak(CD')}}$ 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 $q_D \cdot \epsilon^{\mathsf{AC}}$, hence $p_6 \leq p_5 + q_D \cdot \epsilon^{\mathsf{AC}}$.

Game G_7*:* Let $\mathsf{A}_7 = \mathsf{A}_6$ and G_7 be as G_6 except that for every instance of uKE executed in step I.1, e.g. between Π_i^{S} and Π_j^{C}, if the adversary is an eavesdropper on such instance then G_7 replaces key K_{CS} established by Π_i^{S} and Π_j^{C} with a random key. By the security of the key exchange scheme uKE, it follows that $p_7 \leq p_6 + \min(n_C, n_S) \cdot \epsilon^{\mathsf{uKE}}$.

Game G_8*:* Let $\mathsf{E_{ACbreak(CS)}}$ be an event that there is some session pair $(\Pi_i^{\mathsf{S}}, \Pi_j^{\mathsf{C}})$ 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 $\mathsf{A}_8 = \mathsf{A}_7$ and G_8 be as G_7 except that G_8 aborts if $\mathsf{E_{ACbreak(CS)}}$ 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 $p_8 \leq p_7 + \max(n_C, n_S) \cdot \epsilon^{\mathsf{AC}}$.

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 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 the case when A either attacks the SAS 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 $\mathsf{E_{zidOmit(i)}}$ 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 $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, (\mathsf{pk}, zid_i)]$ for some pk, i.e. if some Π_l^{D} receives value zid_i, it receives it as part of a message $\mathsf{M_C}$ originated by some client session Π_j^{C}.

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 $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^* = (\mathsf{pk}^*, zid_i)]$ for some pk^* in response to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ 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_l^{D} 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_i^{S} 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 which 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 to an attack on the TFA-KE. Specifically, we will construct an algorithm A^* which runs in time comparable to A, achieves advantage $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}} = 2 \cdot (p_{11} - 1/2)$ 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 which come out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$. Recall that $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} = 2 \cdot (p_0 - 1/2)$ and that by the game changes above we have that $|p_{11} - p_0|$ is a negligible quantity, and hence $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}}$ is negligibly close to $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}}$.

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_i^{S}, Π_j^{C}, and Π_l^{D}, A^* starts up its local state for these sessions, which we will denote $\bar{\Pi}_i^{\mathsf{S}}$, $\bar{\Pi}_j^{\mathsf{C}}$, and $\bar{\Pi}_l^{\mathsf{D}}$.

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

Emulation of Step II of TFAgen *to* A*:* A^* picks $(\mathsf{sk}, \mathsf{pk})$ as C in step II.1 and sends $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ to A for $\mathsf{M_C} = (\mathsf{pk}, zid)$ and $zid = zid_j^{\mathsf{C}}$, where l is some new index in $[n_D]$ specified by A. If A responds with $[\mathsf{SAS.CONNECT}, sid]$ and zid was not sent to D before (otherwise $\bar{\Pi}_l^{\mathsf{D}}$ aborts), A^* generates e_D as an encryption of two fixed bitstrings as in G_{11} (see G_4). If A forwards this e_D to $\bar{\Pi}_j^{\mathsf{C}}$, A^* sets $z_j^{\mathsf{C}} = Z(zid_j^{\mathsf{C}})$, picks a random key $K_{CD,j}^{\mathsf{C}}$, sets $K_{CD,l}^{\mathsf{D}} = K_{CD,j}^{\mathsf{C}}$, and denotes such $\bar{\Pi}_j^{\mathsf{C}}, \bar{\Pi}_l^{\mathsf{D}}$ instances as *paired.* If, on the other hand, A responds with $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^*]$ for $\mathsf{M_C}* = (\mathsf{pk}^*, zid^*)$ s.t. zid^* was not sent to D before (otherwise $\bar{\Pi}_l^{\mathsf{D}}$ aborts), A^* picks a coin ρ_l as in G_{11} (see G_2) and aborts $\bar{\Pi}_l^{\mathsf{D}}$ unless $\rho_l = 1$ (which happens with probability 2^{-t}). If $\bar{\Pi}_l^{\mathsf{D}}$ does not abort, A^* picks a random key $K_{CD,l}^{\mathsf{D}}$ and sends out $e_D = \mathsf{Enc}(\mathsf{pk}^*, (Z(zid^*), K_{CD,l}^{\mathsf{D}}))$. If A didn't respond with $[\mathsf{SAS.CONNECT}, sid]$ or it did but $\bar{\Pi}_j^{\mathsf{C}}$ receives e_D^* which is different from e_D sent by $\bar{\Pi}_l^{\mathsf{D}}$, A^* sets $(z_j^{\mathsf{C}}, K_{CD,j}^{\mathsf{C}}) \leftarrow \mathsf{Dec}(\mathsf{sk}, e_D^*)$.

As in G_{11}, A^* can abort some sessions at this point: (1) A^* aborts $\bar{\Pi}_l^{\mathsf{D}}$ if A responds with $[\mathsf{SAS.CONNECT}, sid]$ above but doesn't forward e_D to $\bar{\Pi}_j^{\mathsf{C}}$ (see G_6); (2) A^* aborts $\bar{\Pi}_i^{\mathsf{S}}$ 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 $\bar{\Pi}_i^{\mathsf{S}}$ in step I, (b) A either does not send $[\mathsf{SAS.ATTACK}, sid, \cdot]$ with zid_i^{S} or it does but the corresponding coin-toss ρ comes out 0, (c) A doesn't sent zid_i^{S} to any $\bar{\Pi}_j^{\mathsf{C}}$ session, or it does for some j but then either does not do $[\mathsf{SAS.CONNECT}, sid]$ or does not deliver the resulting e_D to $Clinstprimej$; (3) A^* aborts $\bar{\Pi}_i^{\mathsf{S}}$ 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_i^{S} to any $\bar{\Pi}_j^{\mathsf{C}}$ instance, sends $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid_i^{\mathsf{S}})]$ to some $\bar{\Pi}_l^{\mathsf{D}}$ but coin ρ_l comes out 0.

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

By the above rules the only Π_i^{S} instances on which A^* can be rogue are s.t. A was not passive in the key exchange with $\bar{\Pi}_i^{\mathsf{S}}$ in step I, and there is a *unique* $l \in [n_S]$ s.t. A sent $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid_i^{\mathsf{S}})]$ in response to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \cdot]$, and $\bar{\Pi}_l^{\mathsf{D}}$ did not abort which in particular implies that coin ρ_l came out 1. Note also that the only Π_l^{D} instances on which A^* can be rogue are s.t. A sent $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid^*)]$ in response to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \cdot]$, and $\bar{\Pi}_l^{\mathsf{D}}$ did not abort, implying again $\rho_l = 1$. Therefore each rogue session Π_i^{S} corresponds to a unique rogue session Π_l^{D}, hence w.l.o.g. we can assume that there is a 1-1 relation between rogue Π_i^{S} sessions and rogue Π_l^{D} 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 which 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 $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}} = 2\cdot(p_{11} - 1/2)$, and hence $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} \leq \mathsf{Adv}_{\mathsf{A}*}^{\mathsf{DEPAKE}} + 2(p_{11} - p_0)$.

Finally, we need to 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 $\mathsf{Adv}_{Adv^*}^{\mathsf{DEPAKE}}$ 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 $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}}$ 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, $\mathsf{Adv}_{Adv^*}^{\mathsf{DEPAKE}}$ 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' comparable to T (namely T plus the emulation work of A^* which takes at most a few symmetric-cipher ops per each party instance) then the TFA-KE scheme TFAgen is (T, ϵ)-secure for $\epsilon \leq \epsilon^{\mathsf{DEPAKE}} + (p_{11} - p_0) \leq n \cdot (\epsilon^{\mathsf{KEM}} + \epsilon^{\mathsf{SAS}} + \epsilon^{\mathsf{PKE}} + \epsilon^{\mathsf{uKE}} + 6\epsilon^{\mathsf{AC}}) + n^2/2^\kappa$ 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.

Case 2: Party corruptions. In the full version of the paper we give a formal proof for the bounds in case of client corruption and of device and/or server corruption, showing that our scheme achieves the bounds from Definition 1. Here we just comment on how these bounds are derived. For the case of device corruption, 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 case of device corruption. Similarly, in case of server corruption 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 PTR protocol over the SAS channel, hence reducing the probability of the attacker successfully testing a candidate password pwd' by 2^{-t}. In the case of client compromise where the attacker learns the user's password pwd, we set $d = 0$ (a dictionary of size 1) and set $q_C = q_C' = 0$ since C is corrupted and the attacker cannot choose a test session at C. Finally, when both D and S (but not C) are corrupted one gets the same security as plain DE-PAKE, namely, requiring a full offline dictionary attack to recover pwd.

□