Two-Factor Authentication with End-to-End Password Security

Stanislaw Jarecki*, Hugo Krawczyk**, Maliheh Shirvanian***, and Nitesh Saxena†

Abstract. We present a secure two-factor authentication (TFA) scheme 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 can attack all parts of the system, including all communication links and any subset of parties (servers, devices, client terminals), can learn users' passwords, and perform active and passive attacks, online and offline. In all cases the scheme provides the highest attainable security bounds given the set of compromised components. Our solution builds a TFA scheme using any Device-Enhanced PAKE, defined by Jarecki et al., and any Short Authenticated String (SAS) Message Authentication, defined by Vaudenay. We show an efficient instantiation of this modular construction which utilizes any password-based client-server authentication method, with or without reliance on public-key infrastructure. The security of the proposed scheme is proven in a formal model that we formulate as an extension of the traditional PAKE model.

We also report on a prototype implementation of our schemes, including TLS-based and PKI-free variants, as well as several instantiations of the SAS mechanism, all demonstrating the practicality of our approach.

1 Introduction

Passwords provide the dominant mechanism for electronic authentication, protecting a plethora of sensitive information. However, passwords are vulnerable to both *online* and *offline* attacks.[1] A network adversary can test password guesses in online interactions with the server while an attacker who compromises the authentication data stored by the server (i.e., a database of salted password hashes) can mount an *offline dictionary attack* by testing each user's authentication information against a dictionary of likely password choices. Offline dictionary attacks are a major threat, routinely experienced by commercial vendors, and they lead to the compromise of *billions* of user accounts [7, 6, 15, 20, 17, 12]. Moreover, because users often re-use their passwords across multiple services, compromising one service typically also compromises user accounts at other services.

Two-factor password authentication (TFA), where user U authenticates to server S by "proving possession" of an auxiliary personal device D (e.g. a smartphone or a USB token) in addition to knowing her password, forms a common

* U. California Irvine. Email: sjarecki@uci.edu.
** IBM Research. Email: hugo@ee.technion.ac.il.
*** U. Alabama at Birmingham. Email: maliheh@uab.edu
† U. Alabama at Birmingham. Email: saxena@cis.uab.edu
1 This is a full version of a paper which appeared in PKC'18 [38].

defense against *online* password attacks as well as a second line of defense in case of password leakage. A TFA scheme which uses a device that is not directly connected to U's client terminal C typically works as follows: D displays a short one-time secret PIN, either received from S (e.g. using an SMS message) or computed by D based on a key shared with S, and the user manually types the PIN into client C in addition to her password. Examples of systems that are based on such one-time PINs include SMS-based PINs, TOTP [10], HOTP [14], Google Authenticator [4], FIDO U2F [2], and schemes in the literature such as [48].

Vulnerabilities of traditional TFA schemes. Existing TFA schemes, both PIN-based and those that do not rely on PINs, e.g. [8, 1], combine password authentication and 2nd-factor authentication as separate authentication mechanisms leading to several limitations. Chief among these is that such TFA solutions remain vulnerable to *offline dictionary attacks* upon server compromise in the same way as non-TFA password authentication schemes (i.e. via exposure of users' salted hashes), thus perpetuating the main source of password leakage. Moreover, existing TFA's have several vulnerabilities against *online* attacks: (1) The read-and-copy PIN-transfer is subject to a variety of *eavesdropping attacks*, including SMS hijacking [2], shoulder-surfing, PIN recording, client-side or device-side attacks via keyloggers or screen scrapers, e.g. [43], and PIN *phishing* [16]. (2) The read-and-copy PIN-transfer allows only *limited PIN entropy* and while, say, a 6-digit PIN is hard to guess, PIN guessing can be used in a large-scale online attack against accounts whose passwords the attacker already collected, e.g. [15, 20, 17, 12]. For example, if the attacker obtains password information for a large set of accounts, PINs are 6-digit long, and the attacker can try 10 PIN guesses per account, one expects a successful impersonation per 100,000 users. (3) Current PIN-based TFAs perform *sequential authentication* using the password and the PIN, i.e. C sends the password to S (over TLS), S confirms whether pwd is correct, and only then C sends to S the PIN retrieved from D. This enables online password attacks without requiring PIN guessing or interaction with a device, thus voiding the effects of PIN on password-guessing or password-confirmation online attacks.

Our Contributions. In this paper we aim to address the vulnerabilities of the currently deployed TFA schemes by (1) introducing a precise security model for TFA schemes capturing well-defined *maximally-attainable* security bounds, (2) exhibiting a practical TFA scheme which we prove to achieve the strong security guaranteed by our formal model, and (3) prototyping several methods for validating user's possession of the secondary authentication factor. We expand on each of these aspects next.

TFA Security Model with End-to-End Security. We introduce a *Two-Factor Authenticated Key Exchange (TFA-KE)* model in which a user authenticates to server S by (1) entering a password into client terminal C and (2) proving

[2] E.g., SIM card swap attacks [18] and SMS re-direction where PINs are diverted to the attacker's phone exploiting SS7 vulnerabilities [21]. The latter led to NIST's recent decision to deprecate SMS PINs as a TFA mechanism [19].

possession of a personal device D which forms the second authenticator factor. In the TFA-KE model, possession of D is proved by the user confirming in the device equality of a t-bit *checksum* displayed by D with a *checksum* displayed by C. Following [51] (see below), this implements a t-bit C-to-D *user-authenticated channel*, which confirms that the same person is in control of client C and device D. This channel authentication requirement is weaker than the *private* channel required by current PIN-based TFAs and, as we show, it allows TFA schemes to be both more secure *and* easier to use.

The TFA-KE model, that we define as an extension of the standard Password-Authenticated Key Exchange (PAKE) [24] and the Device-Enhanced PAKE (DE-PAKE) [37] models, captures what we call *end-to-end security* by allowing the adversary to *control all communication channels and compromise any protocol party.* For each subset of compromised parties, the model specifies *best-possible security bounds*, leaving inevitable (but costly) exhaustive online guessing attacks as the only feasible attack option. In particular, in the common case that D and S are uncorrupted, the only feasible attack is an active *simultaneous online* attack against *both* S and D that also requires guessing the password *and* the t-bit checksum. Compromising server S allows the attacker to impersonate S, but does not help in impersonating the user to S, and in particular does not enable an offline-dictionary attack against the user's password. Compromising device D makes the authentication effectively password-only, hence offering best possible bounds in the PAKE model (in particular, the offline dictionary attack is possible only if D and S are both compromised). Finally, compromising client C leaks the password, but even then impersonating the user to the server requires an active attack on D. We prove our protocols in this strong security model.

Practical TFA with End-to-End Security. Our main result is a TFA scheme, GenTFA that achieves end-to-end security as formalized in our TFA-KE model and is based on two general tools. The first is a Device-Enhanced Password Authenticated Key Exchange (DE-PAKE) scheme as introduced by Jarecki et. al [37]. Such a scheme assumes the availability of a user's auxiliary device, as in our setting, and utilizes the device to protect against offline dictionary attacks in case of server compromise. However, DE-PAKE schemes provide no protection in case that the client machine C is compromised and, moreover, security completely breaks down if the user's password is leaked. Thus, our approach for achieving TFA-KE security is to start with a DE-PAKE scheme and armor it against client compromise (and password leakage) using our second tool, namely, a SAS-MA (Short-Authentication-String Message Authentication) as defined by Vaudenay [51]. In our application, a SAS-MA scheme utilizes a t-bit user-authenticated channel, called a *SAS channel*, to authenticate data sent from C to D. More specifically, the SAS channel is implemented by having the user verify and confirm the equality of two t-bit strings, called *checksums*, displayed by both C and D. It follows from [51] that if the displayed checksums coincide then the information received by D from C is correct except for a 2^{-t} probability of authentication error. We then show how to combine a DE-PAKE scheme with such a SAS channel to obtain a scheme, GenTFA, for which we can prove TFA-

KE security, hence provably avoiding the shortcomings of PIN-based schemes. Moreover, the use of the SAS channel relaxes the required user's actions from a read-and-copy action in traditional schemes to a simpler compare-and-confirm which also serves as a proof of physical possession of the device by the user (see more below).

We show a concrete *practical* instantiation of our general scheme GenTFA, named OpTFA, that inherits from GenTFA its TFA-KE security. Protocol OpTFA is modular with respect to the (asymmetric) password protocol run between client and server, thus it can utilize protocols that assume PKI as the traditional password-over-TLS, or those that do not require any form of secure channels, as in the (PKI-free) asymmetric PAKE schemes [25, 32]. In the PKI case, OpTFA can run over TLS, offering a ready replacement of current TFA schemes in the PKI setting. In the PKI-free case one gets the advantages of the TFA-KE setting without relying on PKI, thus obtaining a strict strengthening of (password-only) PAKE security [24, 45] as defined by the TFA-KE model.

The cost of OpTFA is two communication rounds between D and C, with 4 exponentiations by C and 3 by D, plus the cost of a password authentication protocol between C and S. In the PKI setting the latter is the cost of establishing a server-authenticated TLS channel, while in the PKI-free case one can use an asymmetric PAKE (e.g., [27, 36]) with cost (some of it computable offline) of 3 exponentiations for C, 2 for S, and one multi-exponentiation for each.

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

Related Works. We discuss related works in greater detail in Section 7. The main observations are: First, multiple methods have been proposed in the crypto literature for strengthening password authentication against offline dictionary attacks in case of server compromise by introducing an additional party in the protocol (e.g., *password-hardened* or *device-enhanced* authentication [31, 27, 23, 37] and Threshold-PAKE or 2-PAKE, e.g. [44, 28, 40]), but these schemes offer no security against an active attacker in case of password leakage or client compromise, hence they are not TFAs. Second, many TFA schemes offer alternatives to PIN-based TFAs, but *none of them offer protection against offline attacks upon server compromise* except for the scheme of [48] (see Section 7). Moreover, if these schemes consider D as an independent entity (rather than a local component of client C) then they either have on-line security vulnerabilities or they require a pre-set secure full-bandwidth C-D channel. In our case, we do with just a SAS channel that as we show in Section 6 has several practical implementations. Third, we are not aware of any attempt to model security of TFA schemes where D and C are not co-located, nor do we know any PKI-free TFA schemes proposed for this setting.

Road-Map In Section 2 we present TFA-KE security model. In Section 3 we describe our protocol building blocks, including the SAS-MA protocol of [51]. In Section 4 we present a practical TFA-KE protocol OpTFA, and we provide informal rationale for its design choices. In Section 5 we show a more general TFA-KE protocol GenTFA, of which OpTFA is an instance, together with its formal security proof. In Section 6 we report on the implementation and testing of protocol OpTFA, and we describe several SAS channel designs. In Section 7 we include a more detailed discussion of related works.

2 TFA-KE Security

We introduce the *Two-Factor Authenticated Key Exchange (TFA-KE)* security model that defines the assumed environment and participants in our protocols as well as the attacker's capabilities and the model's security guarantees. Our starting point is the *Device-Enhanced PAKE (DE-PAKE)* model, introduced in [37], which extends the well-known two-party *Password-Authenticated Key Exchange (PAKE)* model [24] to a multi-party setting that includes users U, communicating from client machines C, servers S to which users log in, and auxiliary *devices* D, e.g. a smartphone. A DE-PAKE scheme has the security properties of a two-server PAKE (2-PAKE) [28, 40] where D plays the role of the 2nd server. Namely, a compromise of either S or D (but not both) essentially does not help the attacker, and in particular leaks no information about the user's password. However, whereas 2-PAKE might be insecure in case of a compromise of *both* S and D, in a DE-PAKE the adversary who compromises S and D must stage an offline dictionary attack to learn anything about the password. We recall the standard PAKE model of [24] and its Device-Enhanced PAKE (DE-PAKE) extension of [37] in Appendix A.

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

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

To that end our TFA-KE model augments the communication model by an authentication abstraction on the client-to-device channel, but it does so without requiring the client to store any long-term keys (other than the user's password). Namely, we assume a uni-directional C-to-D "Short Authenticated String" (SAS) channel, introduced by Vaudenay [51], which allows C to communicate t bits to D that cannot be changed by the attacker. The t-bit C-to-D SAS channel abstraction comes down to a requirement that the user compares a t-bit *checksum* displayed by both C and D, and approves (or denies) their equality by choosing the corresponding option on device D.

As is standard, we quantify security by attacker's resources that include the computation time and the number of instances of each protocol party the adversary interacts with. We denote these as q_D, q_S, q_C, q'_C, where the first two count the number of active sessions between the attacker and D and S, resp., while q_C (resp. q'_C) counts the number of sessions where the attacker poses to C as S (resp. as D). Security is further quantified by the password entropy d (we assume the password is chosen from a dictionary of size 2^d known to the attacker), and parameter t, which is called the SAS channel *capacity*. As we explain in Section 3, a C-to-D SAS channel allows for establishing a D-authenticated secure channel between D and C, except for the 2^{-t} probability of error [51], which explains 2^{-t} factors in the TFA-KE security bounds stated below.

TFA Security Definition. We consider a communication model of open channels plus the t-bit SAS-channel between C and D, and a man-in-the-middle adversary that interacts with q_D, q_S, q_C, q'_C sessions of D, S, C, as described above. The adversary can also corrupt any party, S, D, or C, learning its stored secrets and the internal state as that party executes its protocol, which in the case of C implies learning the user's password. All other adversarial capabilities as well as the test session experiment defining the adversary's goal are as in DE-PAKE (and PAKE) models – see Appendix A. In particular, the adversary's advantage is, as in DE-PAKE and PAKE, an advantage in distinguishing between a random string and a key computed by S or C on a test session.

The security requirements set by Definition 1 below are the *strictest* one can hope for given the communication and party corruption model. That is, wherever we require the attacker's advantage to be no more than a given bound with a set of corrupted parties, then there is an (unavoidable) attack - in the form of exhaustive guessing attack - that achieves this bound under the given compromised parties. Importantly, and *in contrast to typical two-factor authentication solutions*, the TFA-KE model requires that the second authentication factor D not only provides security in case of client and/or password compromise, but that *it also strengthens online and offline security (by 2^t factors) even when the password has not been learned by the attacker.*

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

1. *If* S*,* D*, and* C *are all uncorrupted:*
$$\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} \leq \min\{q_C + q_S/2^t, q_C' + q_D/2^t\}/2^d + \epsilon$$
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 leaks 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 not* C*), and* $\overline{q}_S$ *and* $\overline{q}_D$ *count* A*'s offline operations performed based on resp.* S*'s and* D*'s state:* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} \leq \min\{\overline{q}_S, \overline{q}_D\}/2^d$

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

In case of client corruption (and password leakage), the adversary's probability of impersonating the user to the server is at most $\min(q_S, q_D)/2^t$, which is the best possible bound when the attacker holds the user's password. In case of device corruption, the adversary's advantage is at most $(q_C+q_S)/2^d$, which matches the optimal PAKE probability, namely, when a device is not available. Finally, upon server corruption, the adversary's probability of success in impersonating the user to any uncorrupted server session is (assuming $q'_C = 0$ for simplicity) at most $q_D/2^{t+d}$. In other words, learning server's private information necessarily allows the adversary to authenticate as the server to the client, but it does not help to impersonate as the client to the server. In contrast, widely deployed PIN-based TFA schemes that transmit passwords and PINs over a TLS channel are subject to an offline dictionary attack in this case.

A possible strengthening. Note that when C and D are corrupted, there is no security to be offered (the attacker has possession of the device and the password). However, in the case that both C and S are corrupted one can hope that the attacker could not authenticate to sessions in S that the attacker does not actively control. This is indeed the case and the above model can be extended to include this case with a bound of $q_D/2^t$. Our protocols as described in Figures 3 and 4 do not achieve this bound. However the following small modification does (refer to the figures). S is initialized with a public key of D and before sending the value zid to D (via C), S encrypts it under D's public key.

3 Building Blocks

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

SAS-MA Scheme of Vaudenay [51]. The Short Authentication String Message Authentication (SAS-MA) scheme allows the transmission of a message from a sender to a receiver so that the receiver can check the integrity of the received message. A SAS-MA scheme considers two communication channels. One that allows the transmission of messages of arbitrary length and is controlled by an active man-in-the-middle, and another that allows sending up to t bits that cannot be changed by the attacker (neither channel is assumed to provide secrecy). We refer to these as the *open channel* and the *SAS channel,* respectively, and call the parameter t the *SAS channel capacity.* A SAS-MA scheme is called *secure* if the probability that the receiver accepts a message modified by a (computationally bounded) attacker on the open channel is no more than 2^{-t} (plus a negligible fraction). In Figure 1 we show a secure SAS-MA implementation of [51] for a sender C and a receiver D. The SAS channel is abstracted as a comparison of two t-bit strings $\mathsf{checksum}_C$ and $\mathsf{checksum}_D$ computed by sender and receiver, respectively. As shown in [51], the probability that an active man-in-the-middle attacker between D and C succeeds in changing message $\mathsf{M_C}$ while D and C compute the same checksum is at most 2^{-t}. Note that this level of security is achieved without any keying material (secret or public) pre-shared between the parties. Also, importantly, there is no requirement for checksums to be secret. (In Section 5 we present a formal SAS-MA security definition.)

Thus, the SAS-MA protocol reduces integrity verification of a received message $\mathsf{M_C}$ to verifying the equality of two strings (checksums) assumed to be transmitted “out-of-band", namely, away from adversarial control. In our application, the checksums will be values displayed by device D and client C whose equality the user verifies and confirms via a physical action, e.g. a click, a QR snapshot, or an audio read-out (see Section 6). In the TFA-KE application this user-confirmation of checksum equality serves as evidence for the physical control of the terminal C and device D by the same user, and a confirmation of user's possession of the 2nd authentication factor implemented as device D.

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

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

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

SAS-MA Protocol:

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

Fig. 1: SAS Message Authentication (SAS-MA) [51]

SAS-SMT. One can use a SAS-MA mechanism from C to D to bootstrap a *confidential channel* from D to C. The transformation is standard: To send a message m securely from D to C (in our application m is a one-time key and D's PTR response, see below), C picks a CCA-secure public key encryption key pair $(\mathsf{sk}, \mathsf{pk})$ (e.g., pair (x, g^x)) for an encryption scheme $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$, sends pk to D, and then C and D execute the SAS-MA protocol on $\mathsf{M_C} = \mathsf{pk}$. If D accepts, it sends m encrypted under pk to C, who decrypts it using sk. The security of SAS-MA and the public-key encryption imply that an attacker can intercept m (or modify it to some related message) only by supplying its own key pk' instead of C's key, and causing D to accept in the SAS-MA authentication of pk' which by SAS-MA security can happen with probability at most 2^{-t}. The resulting protocol has 4 messages, and the cost of a plain Diffie-Hellman exchange if implemented using ECIES [22] encryption. We refer to this scheme as SAS-SMT (SMT for “secure message transmission").

aPAKE. Informally, an aPAKE (for asymmetric or augmented PAKE) is a password protocol secure against server compromise [25, 32], namely, one where the server stores a one-way function of the user's password so that an attacker who breaks into the server can only learn information on the password through an exhaustive offline dictionary attack. While the aPAKE terminology is typically used in the context of password-only protocols that do not rely on public keys, we extend it here (following [37]) to the standard PKI-based password-over-

TLS protocol. This enables the use of our techniques in the context of TLS, a major benefit of our TFA schemes. Note that this standard protocol, while secure against server compromise is not strictly an aPAKE as it allows an attacker to learn plaintext passwords (decrypted by TLS) for users that authenticate while the attacker is in control of the server. As shown in [37], dealing with this property requires a tweak in the DE-PAKE protocol (C needs to authenticate the value b sent by D in the PTR protocol described below - see also Sec. 6).

DE-PAKE. A Device-Enhanced PAKE (DE-PAKE) [37] is an extension of the asymmetric PAKE model by an auxiliary device, which strengthens aPAKE protocols by eliminating offline dictionary attacks upon server compromise. We discuss DE-PAKE in more detail in Section 2 and recall its formal model in Appendix A. We use DE-PAKE protocols as a main module in our general construction of TFA-KE, and our practical instantiation of this construction, protocol OpTFA, uses the DE-PAKE scheme of [37] which combines an asymmetric aPAKE with a password hardening procedure PTR described next.

Password-to-Random Scheme PTR. A PTR is a password hardening procedure that allows client C to translate with the help of device D (which stores a key k) a user's *master password* pwd into independent pseudorandom passwords (denoted rwd) for each user account. The PTR instantiation from [37] is based on the Ford-Kaliski's Blind Hashed Diffie-Hellman technique [31]: Let G be a group of prime order q, let H' and H be hash functions which map onto, respectively, elements of G and κ-bit strings, where κ is a security parameter. Define $F_k(x) = H(x, (H'(x))^k)$, where the key k is chosen at random in $\mathbb{Z}_q$. In PTR this function is computed jointly between C and D where D inputs key k and C inputs $x = \mathsf{pwd}$ as the argument, and the output, denoted $\mathsf{rwd} = F_k(\mathsf{pwd})$, is learned by C only. The protocol is simple: C sends $a = (H'(\mathsf{pwd}))^r$ for r random in $\mathbb{Z}_q$, D responds with $b = a^k$, and C computes $\mathsf{rwd} = H(x, b^{1/r})$. Under the One-More (Gap) Diffie-Hellman (OM-DH) assumption in the Random Oracle Model (ROM), this scheme realizes a universally composable oblivious PRF (OPRF) [36], which in particular implies that $x = \mathsf{pwd}$ is hidden from all observers and function $F_k(\cdot)$ remains pseudorandom on all inputs which are not queried to D.

4 OpTFA: A Practical Secure TFA-KE Protocol

In Section 5 we present and prove a general design, GenTFA, of a TFA-KE protocol based on two generic components, namely, a SAS-MA and DE-PAKE protocols. But first, in this section, we show a practical instantiation of GenTFA using the specific building blocks presented in Section 3, namely, the SAS-MA scheme from Fig. 1 and the DE-PAKE scheme from [37] (that uses the DH-based PTR scheme described in that section composed with any asymmetric PAKE). This concrete instantiation serves as the basis of our implementation work (Section 6) and helps explaining the rationale of our general construction. OpTFA is presented in Figure 3. A schematic representation is shown in Figure 2.

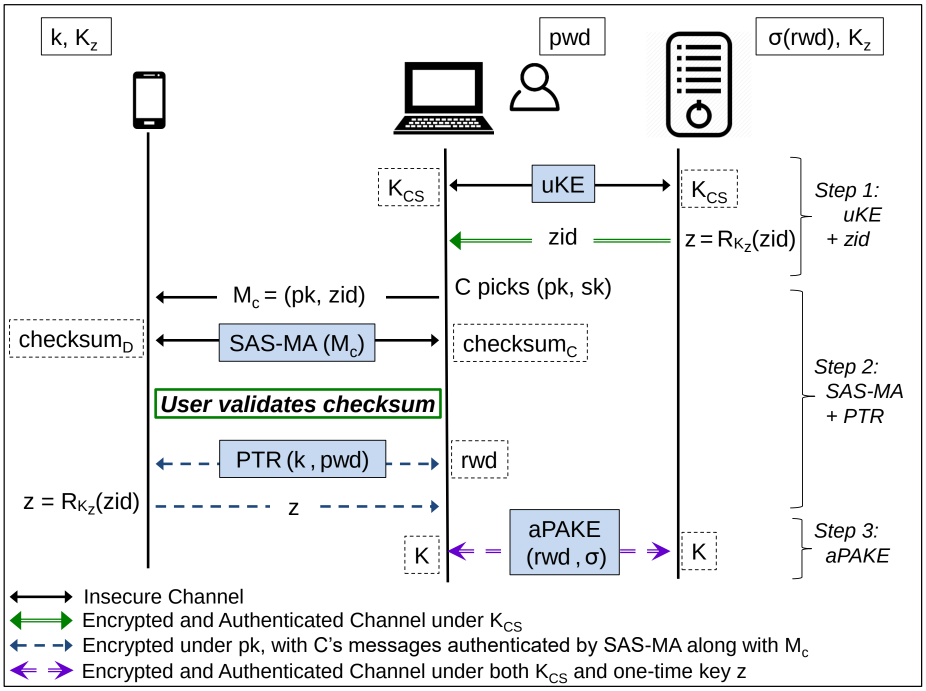

Fig. 2: Schematic Representation of Protocol OpTFA of Fig. 3

Enhanced TFA via SAS. Before going into the specifics of OpTFA, we describe a *general technique* for designing TFA schemes using a SAS channel. In traditional TFA schemes, a PIN is displayed to the user who copies it into a login screen to prove access to that PIN. As discussed in the introduction, this mechanism suffers of significant weaknesses mainly due to the low entropy of PINs (and inconvenience of copying them). We suggest automating the transmission of the PIN over a *confidential channel* from device D to client C. To implement such channel, we use the SAS-SMT scheme from Sec. 3 where security boils down to having D and C display t-bit strings (checksums) that the user checks for equality. In this way, low-entropy PINs can be replaced with full-entropy values (we refer to them as *one-time keys (OTK)*) that are immune to eavesdropping and bound active attacks to a success probability of 2^{-t}. These active attacks are impractical even for $t = 20$ (more a denial-of-service than an impersonation threat) and with larger t's as illustrated in Sec. 6 they are just infeasible. Note that this approach works with any form of generation of OTK's, e.g., time-based mechanisms, challenge-response between device and server, etc.

4.1 OpTFA Explained

Protocol OpTFA (Fig. 3) requires several mechanisms that are necessary to obtain the strong security bounds of the TFA-KE model. To provide rationale for

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

Initialization:

1. On input the user's password pwd, pick random k in $\mathbb{Z}_q$ and set $\mathsf{rwd} = F_k(\mathsf{pwd}) = H(\mathsf{pwd}, (H'(\mathsf{pwd}))^k)$;
2. Initialize the asymmetric PAKE scheme aPAKE on input rwd and let σ denote the user's state at the server.
3. Choose random key K_z for PRF R, and set zidSet to the empty set;
4. Give $(k, K_z, \mathsf{zidSet})$ 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 which establishes session key K_{CS} between them;
2. S generates random κ-bit nonce zid, computes $z \leftarrow \mathsf{R}(K_z, zid)$, and sends zid to C authenticated under key K_{CS}.

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

1. C generates PKE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}$, t-bit random value R_C, κ-bit random value d, and random r in Z_q. C then computes $a \leftarrow H'(\mathsf{pwd})^r$, $\mathsf{M_C} \leftarrow (\mathsf{pk}, zid, a)$, $\mathsf{Com} \leftarrow H_{\mathsf{com}}(\mathsf{M_C}, R_C, d)$, 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. Otherwise 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 $\mathsf{rwd} \leftarrow H(\mathsf{pwd}, b^{1/r})$ [$= F_k(\mathsf{pwd})$], and aborts if Dec outputs $\perp$.

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

1. C and S run protocol aPAKE on resp. inputs rwd and σ with all aPAKE messages authenticated by keys z and K_{CS} (each key is used to compute a MAC on each aPAKE message).

   Each party aborts and sets local output to $\perp$ if any of the MAC verifications fails.
2. The final output of C and S equals their outputs in the aPAKE instance: either a session key K or a rejection sign $\perp$.

Fig. 3: OpTFA: Efficient TFA-KE Protocol with Optimal Security Bounds

the need of these mechanisms we show how the protocol is built bottom-up to deliver the required security properties. We stress that while the design is involved the resultant protocol is efficient and practical. The presentation and discussion of security properties here is informal but the intuition can be formalized as we do via the TFA-KE model (Sec. 2), the generic protocol GenTFA in next section and the proof of Theorem 1.

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

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

• OpTFA 0.1. We enhance password-over-TLS with the OTK-over-SAS mechanism described above. First, C transmits the user's password to S over TLS and if the password verifies at S, S sends a nonce zid to C who relays it to D. On the basis of zid (which also acts as session identifier in our analysis), D computes a OTK $z = \mathsf{R}_{K_z}(zid)$ where R is a PRF and K_z a key shared between D and S. D transmits z to C over the SAS-SMT channel and C relays it to S over TLS. The user is authenticated only if the received value z is the same as the one computed by S.

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

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

• OpTFA 0.3. We add defense against offline dictionary attacks upon server compromise by resorting to the DE-PAKE construction of [37] and, in particular, to the password-to-random hardening procedure PTR from Sec. 3. For this, we now assume that the user has a master password pwd that PTR converts into randomized passwords rwd for each user account. By registering rwd with server S and using PTR for the conversion, DE-PAKE security ensures that offline dictionary attacks are infeasible even if the server is compromised (case (3) in Def. 1). Note that the PTR procedure runs between D and C following the establishment of the SAS-SMT channel.

• OpTFA 0.4. We change the run of PTR between D and C so that the value a computed by C as part of PTR is transmitted over the SAS-authenticated channel from C to D. Without this authentication the strict bound of case (3) in Def. 1 (simplified for $q'_C = 0$), namely, $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}} \leq q_D/2^{d+t} + \epsilon$ upon server compromise, would not be met. Indeed, when the attacker compromises server S, it learns the key K_z used to compute the OTK z so the defense provided by OTK is lost. So, how can we still ensure the 2^t denominator in the above bound expression? The answer is that by authenticating the PTR value a under SAS-MA, the attacker is forced to run (expected) 2^t sessions to be able to inject its own value a over that channel. Such injection is necessary for testing a password guess even when K_z is known. When considering a password dictionary of size 2^d this ensures the denominator 2^{d+t} in the security bound.

• OpTFA 0.5. We add the following mechanism to OpTFA: Upon initialization of an authentication session (for a given user), C and S run an *unauthenticated* (a.k.a. anonymous) key exchange uKE (e.g., a plain Diffie-Hellman protocol) to establish a shared key K_{CS} that they use as a MAC key applied to all subsequent OpTFA messages. To see the need for uKE assume it is omitted. For simplicity, consider the case where attacker A knows the user's password. In this case, all A needs for impersonating the user is to learn one value of z which it can attempt by acting as a man-in-the-middle on the C-D channel. After q_D such attempts, A has probability of $q_D/2^t$ to learn z which together with the user's password allows A to authenticate to S. In contrast, the bound required by Def. 1 in this case is the stricter $\min\{q_S, q_D\}/2^t$. This requires that for *each* attempt at learning z in the C-D channel, not only A needs to try to break SAS-MA authentication but it also needs to establish a new session with S. For this we resort to the uKE channel. It ensures that a response z to a value zid sent by S over a uKE session will only be accepted by S if this response comes back on the *same* uKE session (i.e., authenticated with the same keys used by S to send the challenge zid). It means that both zid and z are exchanged with the same party. If zid was sent to the legitimate user then the attacker, even if it learns the corresponding z, cannot use it to authenticate back to S. We note that uKE is also needed in the case that the attacker does not know the password. Without it, the success probability for this case is about a factor $2^d/q_S$ higher than acceptable by Def. 1.
Note. When all communication between C and S goes over TLS, there is no need to establish a dedicated uKE channel; TLS serves as such.

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

• OpTFA 0.7. Finally, we generalize OpTFA so that the password protocol run as the last stage of OpTFA (after PTR generates rwd) can be implemented with *any* asymmetric aPAKE protocol, with or without assuming PKI, using the server-specific user's password rwd. As shown in [37], running any aPAKE protocol on a password rwd produced by PTR results in a DE-PAKE scheme, a property that we use in an essential way in our analysis.

We need one last mechanism for C to prove knowledge of z to S, namely, we specify that both C and S use z as a MAC key to authenticate the messages sent by protocol aPAKE (this is in addition to the authentication of these messages with key K_{CS}). Without this, an attack is possible where in case that OpTFA fails the attacker learns if the reason for it was an aPAKE failure or a wrong z. This allows the attacker to mount an online attack on the password without the attacker having to learn the OTK. (When the aPAKE is password-over-TLS the above MAC mechanism is not needed, the same authentication effect is achieved by encrypting rwd and z under the same CCA-secure ciphertext [33].)

• OpTFA. Version 0.7 constitutes the full specification of the OpTFA protocol, described in Fig. 3, with generic aPAKE.

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

OpTFA Security. Security of OpTFA follows from that of protocol GenTFA because OpTFA is its instantiation. See Theorem 1 in Section 5 and Corollary 1.

5 The Generic GenTFA Protocol

In Figure 4 we show protocol GenTFA which is a generalization of protocol OpTFA shown in Fig. 3 in Section 4. Protocol GenTFA is a compiler which converts *any* secure DE-PAKE and SAS-MA schemes into a secure TFA-KE. It uses the same uKE and CCA-PKE tools as protocol OpTFA, but it also generalizes two other mechanisms used in OpTFA as, resp. a generic symmetric *Key Encapsulation Mechanism* (KEM) scheme and an *Authenticated Channel* (AC) scheme.

A Key Encapsulation Mechanism, denoted $(\mathsf{KemE}, \mathsf{KemD})$ (see e.g. [49]), 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 OpTFA of Figure 3, KEM is implemented using PRF R: zid is a random κ-bit string and $z = R(K_z, zid)$. We also generalize the usage of the MAC function in OpTFA as an Authenticated Channel, defined by a pair ACSend, ACRec, which implements bi-directional authenticated communication between two parties sharing a symmetric key K [29, 34]. Algorithm ACSend takes inputs key K and message m and outputs m with authentication tag computed with key K, while the receiver procedure, $\mathsf{ACRec}(K, \cdot)$, outputs either a message or the rejection symbol $\perp$. We assume that the AC scheme is stateful and provides authenticity and protection against replay.

The security of GenTFA is stated in the following theorem:

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

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

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

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

1. C generates a PKE key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}$, sends $\mathsf{M_C} = (\mathsf{pk}, zid)$ to D, and C and D run SAS-MA to authenticate $\mathsf{M_C}$ using the t-bit C-to-D SAS channel.
2. D aborts if $zid \in \mathsf{zidSet}$ 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 DE-PAKE on resp. inputs pwd, k, and σ, modified as follows:

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

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

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

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

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

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

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

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

As a corollary we obtain a proof of TFA-KE security for protocol OpTFA from Fig. 3 which uses specific secure instantiations of GenTFA components. The corollary follows by applying the result of Vaudenay [51], which implies in particular that the SAS-MA scheme used in OpTFA is secure in ROM, and the result of [37], which implies that the DE-PAKE used in OpTFA is secure under the OM-DH assumption if the underlying aPAKE is a secure asymmetric PAKE.

We note that protocol OpTFA optimizes GenTFA instantiated with the DE-PAKE of [37] by piggybacking the C-D round of communication in that protocol, $a = H'(\mathsf{pwd})^r$ and $b = a^k$, onto resp. C's message $\mathsf{M_C}$ and the plaintext in D's ciphertext e_D. The security proof extends to this round-optimized case because SAS-MA authentication of $\mathsf{M_C}$ and CCA-security of PKE bind DE-PAKE messages a, b to this session just as the $\mathsf{ACSend}(K_{CD}, \cdot)$ mechanism does in (non-optimized) protocol GenTFA.

Corollary 1. *Assuming that* aPAKE *is a secure asymmetric PAKE,* uKE *is secure Key Exchange,* (KG, Enc, Dec) *is a CCA-secure PKE,* R *is a secure PRF, and MAC is a secure message authentication code, protocol* OpTFA *is a secure TFA-KE scheme under the OM-DH assumption in ROM.*

Security definition of SAS authentication. For the purpose of the proof below we state the security property assumed of a SAS-MA scheme which was informally described in Section 3. While [51] defines the security of SAS-MA using a game-based formulation, here we do it via the following (universally composable) functionality $\mathsf{F_{SAS[t]}}$: On input a message $[\mathsf{SAS.SEND}, sid, P', m]$ from an honest party P, functionality $\mathsf{F_{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, P', m]$ to A, and then, if A's response is $[\mathsf{SAS.CONNECT}, sid]$, then $\mathsf{F_{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, m]$ to P', if A's response is $[\mathsf{SAS.ABORT}, sid]$, then $\mathsf{F_{SAS[t]}}$ sends $[\mathsf{SAS.SEND}, sid, P, \perp]$ to P', and if A's response is $[\mathsf{SAS.ATTACK}, sid, m']$ then $\mathsf{F_{SAS[t]}}$ throws a coin ρ which comes out 1 with probability 2^{-t} and 0 with probability $1 - 2^{-t}$, and if $\rho = 1$ then $\mathsf{F_{SAS[t]}}$ sends succ to A and $[\mathsf{SAS.SEND}, sid, P, m']$ to P', and if $\rho = 0$ then $\mathsf{F_{SAS[t]}}$ sends fail to A and $[\mathsf{SAS.SEND}, sid, P, \perp]$ to P'.

In our main instantiation of the generic protocol GenTFA of Figure 4, i.e. in protocol OpTFA of Figure 3, we instantiate SAS-MA with the scheme of [51], but even though the original security argument given for it in [51] used the game-based security notion, it is straightforward to adopt this argument to see that this scheme securely realizes the above (universally composable) functionality.

Proof of Theorem 1. Let A be an adversary limited by time T playing the TFA-KE security game, which we will denote G_0, instantiated with the TFA-KE scheme GenTFA. Let the security advantage defined in Definition 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 GenTFA protocol sessions in which A plays only a passive eavesdropper role except that we allow A to abort any of these protocol executions at any step. Let $n_S = q_S + q_{HbC}$, $n_D = q_D + q_{HbC}$, $n_C = q_C + q'_C + q_{HbC}$, and note that these are the ranges of indexes i, j, l for instances Π_i^{S}, Π_j^{C}, and Π_l^{D}. We will use $[n]$ to denote range $\{1, ..., n\}$.

The security proof goes by cases depending on the type of corrupt queries A makes. In all cases the proof starts from the security-experiment game G_0 and proceeds via a series of game changes, G_1, G_2, etc, until a modified game G_i allows us to reduce an attack on the DE-PAKE with the same corruption pattern (except in the case of corrupt client C) to the attack on G_i. In the case of the corrupt client the argument is different because it does not rely on the underlying DE-PAKE (note that DE-PAKE does not provide any security properties in the case of client corruption). In some game changes we will consider a modified adversary algorithm, for example an algorithm constructed from the original adversary A interacting with a simulator of some higher-level procedure, e.g. the SAS-MA simulator. Wlog, we use A_i for an adversary algorithm in game G_i.

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

Case 1: No party is compromised. This is the case when A makes no corrupt queries, i.e. it's the default "network adversary" case. Below we describe only the game changes in the proof, and we state what we claim about the effects of that game change and what assumption we use. The full details of the proof are included in Appendix B.

Game G_1*:* Let Z be a random function which maps onto κ-bit strings. If (zid_i, z_i) dentes the KEM (ciphertext,key) pair generated by Π_i^{S} then in G_1 we set $z_i = Z(zid_i)$ instead of using KemE, and we abort if there is ever a collision in z_i values. Security of KEM implies that $p_1 \leq p_0 + \epsilon^{\mathsf{KEM}}(n_S) + n_S^2/2^\kappa$.

Game G_2*:* Here we replace the SAS-MA procedure with the simulator $\mathsf{SIM}_{\mathsf{SAS}}$ implied by the UC security of the SAS-MA scheme of [51]. In other words, whenever Π_j^{C} and Π_l^{D} execute the SAS-MA sub-protocol, we replace this execution with a simulator $\mathsf{SIM}_{\mathsf{SAS}}$ interacting with A and the ideal SAS-MA functionality $\mathsf{F}_{\mathsf{SAS[t]}}$. For example, Π_j^{C}, instead of sending $\mathsf{M_C} = (\mathsf{pk}, zid)$ to A_1 and starting a SAS-MA instance to authenticate $\mathsf{M_C}$ to D, will send $[\mathsf{SAS.SEND}, sid, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$ to $\mathsf{F}_{\mathsf{SAS[t]}}$, which triggers $\mathsf{SIM}_{\mathsf{SAS}}$ to start simulating to A the SAS-MA protocol on input $\mathsf{M_C}$ between Π_j^{C} and Π_l^{D}. The rules of $\mathsf{F}_{\mathsf{SAS[t]}}$ imply that $\mathcal{A}$ can make this connection either succeed, abort, or, if it attacks it then Π_l^{D} will abort with probability $1 - 2^{-t}$, but with probability 2^{-t} it will accept $\mathcal{A}$'s message $\mathsf{M_C}^*$ instead of $\mathsf{M_C}$. Security of SAS-MA implies that $p_2 \leq p_1 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{SAS}}$.

Game G_3*:* Here we re-name entities involved in game G_2. Note that adversary A_2 interacts with G_2 which internally runs algorithms $\mathsf{SIM}_{\mathsf{SAS}}$ and $\mathsf{F}_{\mathsf{SAS[t]}}$, and that $\mathsf{SIM}_{\mathsf{SAS}}$ interacts only with $\mathsf{F}_{\mathsf{SAS[t]}}$ on one end and A_2 on the other. We can therefore draw the boundaries between the adversarial algorithm and the security game slightly differently, by considering an adversary A_3 which executes the steps of A_2 and $\mathsf{SIM}_{\mathsf{SAS}}$, and a security game G_3 which executes the rest of game G_2, including the operation of functionality $\mathsf{F}_{\mathsf{SAS[t]}}$. In other words, G_3 interacts with A_3 using the $\mathsf{F}_{\mathsf{SAS[t]}}$ interface to $\mathsf{SIM}_{\mathsf{SAS}}$, i.e. G_3 sends to A_3 messages of the type $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M}_{\mathsf{C}}]$, and A_3's response must be one of $[\mathsf{SAS.CONNECT}, sid]$, $[\mathsf{SAS.ABORT}, sid]$, and $[\mathsf{SAS.ATTACK}, sid, \mathsf{M}_{\mathsf{C}}{}^*]$. 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*:* Here we change game G_3 s.t. if A sends $[\mathsf{SAS.CONNECT}, sid]$ to let the SAS-MA instance go through between Π_j^{C} and Π_l^{D} with M_{C} containing Π_j^{C}'s key pk, then we replace the ciphertext e_D subsequently sent by Π_l^{D} by encrypting a constant string instead of $\mathsf{Enc}(\mathsf{pk}, (z, K_{CD}))$, and if A passes this e_D to Π_j^{C} then it decrypts it as (z, K_{CD}) generated by Π_l^{D}. In other words, we replace the encryption under SAS-authenticated key pk by a "magic" delivery of the encrypted plaintext. The CCA security of PKE implies that $p_4 \leq p_3 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{PKE}}$.

Game G_5*:* Here we abort if, assuming that key pk and ciphertext e_D were exchanged between Π_j^{C} and Π_l^{D} correctly, any party accepts wrong messages in the subsequent DE-PAKE execution authenticated by K_{CD} created by Π_l^{D}. The authentic channel security implies that $p_5 \leq p_4 + \min(n_C, n_D) \cdot \epsilon^{\mathsf{AC}}$.

Game G_6*:* We perform some necessary cleaning-up, and abort if the SAS-MA instance between Π_j^{C} and Π_l^{D}) sent M_{C} correctly, but adversary did not deliver Π_l^{D}'s response e_D back to Π_j^{C} and yet Π_l^{D} did not abort in subsequent DE-PAKE. Since this way Π_j^{C} has no information about key K_{CD} we get $p_6 \leq p_5 + q_D \cdot \epsilon^{\mathsf{AC}}$.

Game G_7*:* We replace the keys created by uKE for every Π_i^{S}-Π_j^{C} session in step I.1 *on which* A *was only an eavesdropper*, with random keys. Security of uKE implies that $p_7 \leq p_6 + \min(n_C, n_S) \cdot \epsilon^{\mathsf{uKE}}$.

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 to be passive on the C-S link in the DE-PAKE in step III. Also, if A does not attack the SAS-MA and delivers D's response to C then A is forced to be passive on the C-D link in the DE-PAKE in step III (and if A does not deliver D's response to C then this D instance will abort too). The remaining cases are either (1) active attacks on the key exchange in step I or (2) when A attacks the SAS-MA sub-protocol and gets D to accept $\mathsf{M}_{\mathsf{C}}* \neq \mathsf{M}_{\mathsf{C}}$ or (3) A sends $e_D^* \neq e_D$ to C. In handling these cases the crucial issue is what A does with the zid 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 $\mathsf{M}_{\mathsf{C}} = (\mathsf{pk}, zid_i)$ 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 we 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*:* Let Π_i^{S} and Π_j^{C} be rogue sessions s.t. A sends zid_i to Π_j^{C} in step I.2, but then stop Π_j^{C} from getting the corresponding z_i by either attacking SAS-MA or misdelivering D's response e_D. In that case neither Π_j^{C} nor A have any information about z_i, and therefore Π_i^{S} should reject. Namely, if in G_9 we set Π_i^{S}'s output to $\perp$ in such cases then $p_9 \leq p_8 + q_S \cdot \epsilon^{\mathsf{AC}}$.

Game G_{10}*:* Let Π_i^{S} and Π_j^{C} be rogue sessions and A send zid_i to Π_j^{C} as above, but now consider the case that A lets Π_j^{C} learn z_i but A does not learn z_i itself, i.e. A lets SAS-MA and e_D go through. In this case we will abort if in DE-PAKE communication in Step III between Π_i^{S} and Π_j^{C} either party accepts a message not sent by the other party. Since A has no information about z_i the authenticated channel security implies that $p_{10} \leq p_9 + \min(q_C, q_S) \cdot \epsilon^{\mathsf{AC}}$.

Note that at this point if A 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} by sending $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid_i)]$ for any l then A is forced to be a passive eavesdropper on the DE-PAKE protocol in step III. Note that this holds 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 $\mathsf{M_C}$ sent by some Π_j^{C}.

Game G_{11}*:* Finally consider the case when A itself sends zid_i to D, i.e. when $L(i) = l$ s.t. A sends $[\mathsf{SAS.ATTACK}, sid, \mathsf{M_C}^* = (\mathsf{pk}^*, zid_i)]$ in response to $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, \mathsf{M_C}]$, but the $\mathsf{F}_{\mathsf{SAS[t]}}$ coin-toss comes out $\rho_l = 0$, i.e. A fails in this SAS-MA attack. In that case we can let Π_i^{S} abort in step III because if $\rho_l = 0$ then A has no information about $z_i = Z(zid_i)$, hence $p_{11} \leq p_{10} + q_S \cdot \epsilon^{\mathsf{AC}}$.

After these game changes, we finally make a reduction from an attack on underlying DE-PAKE to an attack on TFA-KE. Namely, we construct A^* which achieves advantage $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}} = 2 \cdot (p_{11} - 1/2)$ against DE-PAKE, 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 $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{TFA}}$.

The reduction goes through because after the above game-changes A can either essentially let a DE-PAKE instance go through undisturbed, or it can attempt to actively attack the underlying DE-PAKE instance either via a rogue C session or via rogue sessions with device S and server D. However, each rogue D session is bound to a unique rogue S session, because of the uKE and (zid, z) mechanism, and for each such D, S session *pair*, the probability that an active attack is not aborted is only 2^{-t}. This implies that the (q_S, q_D, q_C) parameters characterizing the TFA-KE attacker A scale-down to $(q_S/2^t, q_D/2^t, q_C)$ parameters for the resulting DE-PAKE attacker A^*, which leads to the claimed security bounds by the security of DE-PAKE. The details of construction for A^* and the above argument are included in Appendix B

Case 2: Party corruptions. In the forthcoming revision of this paper we will include formal security proofs for the cases of client corruption and of device and/or server corruption, showing that our scheme achieves all 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.

6 System Development & Testing

Here we report on an experimental prototype of protocol OpTFA from Figure 3 on page 12 and present novel designs for the SAS channel implementation. We experiment with OpTFA using two different instantiations of the password protocol between C and S. One is PKI-based that runs OpTFA over a server-authenticated TLS connection; in particular, it uses this connection in lieu of the uKE in step I and implements step III by simply transmitting the concatenation of password rwd and the value z under the TLS authenticated encryption. The second protocol we experimented with is a PKI-free asymmetric PAKE borrowed from [36, 27]. Roughly, it runs the same PTR protocol as described in Section 3 but this time between C and S. C's input is rwd and the result $F_k(\mathsf{rwd})$ serves as a user's private key for the execution of an authenticated key-exchange between C and S. We implement the latter with HMQV [41] (as an optimization, the DH exchange used to implement uKE in step I of OpTFA is "reused" in HMQV).

In Table 1 we provide execution times for the various protocol components, including times for the TLS-based protocol and the PKI-free one with some elements borrowed from the implementation work from [37]. We build on the following platform. 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 in Java for Android phones.

Table 1: Average execution time of OpTFA and its components (10,000 iterations)

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

All DH-based operations (PTR, key exchange and SAS-SMT encryption) use elliptic curve NIST P-256, and hashing and PRF use HMAC-SHA256. Hashing into the curve is implemented with simple iterated hashing till an abscissa x on the curve is found (it will be replaced with a secure mechanism such as [26]).

Communication between C and S uses a regular internet connection between the browser C and web server S. Communication between C and D (except for checksum comparison) goes over the internet using a bidirectional Google Cloud Messaging (GCM) [5], in which D acts as the GCM server and C acts as the GCM client. GCM involves a registration phase during which GCM client (here C) registers with the GCM generated client ID to the GCM server (here D), to assure that D only responds to the registered clients. In case that the PAKE protocol in OpTFA is implemented with password-over-TLS, [37] specifies the need for D to authenticate the PTR value b sent to C (see Sec. 3). In this case, during the GCM registration we install at C a signature public key of D.

6.1 Checksum Validation Design

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

Manual Checksum Validation. In the simplest approach, the user compares the checksums displayed on D and C and taps the Confirm button on D in case the two match [50]. Although, this type of code comparison has recently been deployed in TFA systems, e.g., [8], it carries the danger of neglectful users pressing the confirm button without comparing the checksum strings. Another common solution for checksum validation is "Copy-Confirm" [50] where the user types the checksum displayed on C into D, and only if this matches D's checksum does D proceeds with the protocol. We implemented this scheme using a 6 digit number. We stress that in spite of the similarity between this mechanism and PIN copying in traditional TFA schemes, there is an essential security difference: Stealing the PIN in traditional schemes suffices to authenticate instead of the user (for an attacker that holds the user's password) while stealing the checksum value entered by the user in OpTFA is worthless to the attacker (the checksum is a validation code, not the OTK value needed for authentication).

The above methods using human visual examination and/or copying limit the SAS channel capacity (typically to 4-6 digits) and may degrade usability [47]. As an alternative we consider the following designs (however one may fallback to the manual schemes when the more secure schemes below cannot be used, e.g., missing camera or noisy environments).

QR Code Checksum Validation. In this checksum validation model, we encode the full, 256-bit checksum computed in protocol OpTFA into a hexstring and show it as a 230×230 pixel QR Code on the web-page. We used ZXing library to encode the QR code and display it on the web page and read and decode it 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. With the larger checksum ($t = 256$) active attacks on SAS-SMT turn infeasible and the expressions 2^{-t} in Definition 1) negligible.

Voice-based Checksum Validation. We implement a voice-based checksum validation approach that assumes a microphone-equipped device (typically a smartphone) where the user speaks a numerical checksum displayed by the client into the device. The device D receives this audio, recognizes and transcribes it using a speech recognition tool, and then compares the result with the checksum computed by D itself. The client side uses a Chrome extension as in the manual checksum validation case while on the device we developed a transcriber application using Android.Speech API. The user clicks on a "Speak" button added to the app and speaks out loud the displayed number (6-digit in our implementation). The transcriber application in D recognizes the speech and convert it to text that is then compared to D's checksum. To further improve the usability of this approach one can incorporate a text-to-speech tool that would speak the checksum automatically (i.e., replacing the user). The transcription approach would perhaps be easy for the users to employ compared to the QR-based approach, but would only be suitable if the user is in an environment that is non-noisy and allows her to speak out-loud. We note that the QR-code and audio-based approaches do not require a browser plugin or add-on and can be deployed on any browser with HTML5 support.

Performance Evaluation. As preliminary information, we report on 30 checksum validation 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 checksum comparison is negligible. Our preliminary testing of these two channels shows virtually-0 error rate.

7 Discussion of Related Work

Device-enhanced password-authentication with security against offline dictionary attacks (ODA). There are several proposals in cryptographic literature for password authentication schemes that utilize an auxiliary computing component to protect against ODA in case of server compromise. This was a context of the *Password Hardening* proposal of Ford-Kaliski [31], which was generalized as *Hidden Credential Retrieval* by Boyen [27], and then formalized as *(Cloud) Single Password Authentication* (SPA) by Acar et al. [23] and as a *Device-Enhanced PAKE* (DE-PAKE) by Jarecki et al. [37]. These schemes are functionally similar to a TFA scheme if the role of the auxiliary component is played by the user's device D, but they are insecure in case of password leakage e.g. via client compromise.[3] The threat of an ODA attack on compromise of an authentication server also motivated the notion of *Threshold* Password Authenticated Key Exchange (T-PAKE) [44], i.e. a PAKE in which the password-holding server is replaced by n servers so that a corruption of up to $t < n$ of them leaks no information about the password. In addition to general T-PAKE's, several solutions were also given for the specific case of $n = 2$ servers tolerating $t = 1$ corruption, known as *2-PAKE* [28, 40], and every 2-PAKE, with the user's device D playing the role of the second server, is a password authentication scheme that protects against ODA in case of server compromise. However, as in the case of [31, 27, 23, 37], if a password is leaked then 2-PAKE offers no security against an active attacker who engages with a single 2-PAKE session.

TFA with ODA security. Shirvanian et al. [48] proposed a TFA scheme which extends the security of traditional PIN-based TFAs against ODA in case of server compromise. However, OpTFA offers several advantages compared to [48]: First, [48] relies on PKI (the client sends the password and the one-time key, OTK, to the PKI-authenticated server) while OpTFA has both a PKI-model and a PKI-free instantiation. Second, [48] assumes full security of the t-bit D-C channel for OTK transmission while we reduce this assumption to a t-bit *authenticated* channel between C and D. Consequently, we improve user experience by replacing the *read-and-copy* action with simpler and easier *compare-and-confirm*. On the other hand, [48] can use *only* the t-bit secure D-C link while OpTFA requires transmission of full-entropy values between D and C.

[3] We note that [23] also show a *Mobile Device* SPA, which provides client-compromise resistance, but it requires the user to type the password onto the device D, and to copy a high-entropy key from D to C, thus increasing manually transmitted data even in comparison to traditional TFAs. By contrast, OpTFA dispenses entirely with manual transmission of information to and from D.

TFA with the 2nd factor as a *local* cryptographic component. Some Two-Factor Authentication schemes consider a scenario where the 2nd factor is a device D capable of storing cryptographic keys and performing cryptographic algorithms, but unlike in our model, D is connected directly to client C, i.e. it effectively communicates with C over secure links. (However, security must hold assuming the adversary can stage a lunch-time attack on device D, so D cannot simply hand off its private keys to C.) The primary example is a USB stick, like YubiKey [13], implementing e.g. the FIDO U2F authentication protocol [2, 42]. A generalized version of this problem, including biometric authentication, was formalized by Pointcheval and Zimmer as *Multi-Factor Authentication* [46], but the difference between that model and our TFA-KE notion is that we consider device D which has *no pre-set secure channel with client* C. Moreover, to the best of our knowledge, all existing MFA/TFA schemes even in the secure-channel D-C model are still insecure against ODA on server compromise, except for the aforementioned TFA of Shirvanian et al. [48].

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

PhoneAuth [30] and Authy [11] replace PINs with S-to-D challenge-response communication channeled by C, but they require a pre-paired Bluetooth connection to secure the C-D channel. A full-bandwidth secure C-D channel reduces the three-party TFA notion to a two-party setting, where device D is a local component of client C, but requiring an establishment of such secure connection between a browser C and a cell phone D makes a TFA scheme harder to use. TFA schemes like SlickLogin (acquired by Google) [3], Sound-Login [9], and Sound-Proof [39] in essence attempt to implement such secure C-to-D channel using physical security assumptions on physical media e.g. near-ultrasounds [3], audible sounds [9], or ambient sounds detecting proximity of D to C [39], but they are subject to eavesdropping attacks and co-located attackers.

Several TFA proposals, including Google Prompt [8] and Duo [1], follow a *one-click* approach to minimize user's involvement if D is a data-connected device like a smartphone. In [8, 1] S communicates directly over data-network to D, which prompts the user to approve (or deny) an authentication session, where the approve action prompts D to respond in an entity authentication protocol with S, e.g. following the U2F standard [2]. This takes even less user's involvement than the compare-and-confirm action of our TFA-KE, but it does not establish a strong binding between the C-S login session and the D-S interaction. E.g., if the adversary knows the user's password, and hence the TFA security depends entirely on D-S interaction, a man-in-the-middle adversary who detects C's attempt to establish a session with S, and succeeds in establishing a session with S before C does, will authenticate as that user to S because the honest user's approval on D's prompt will result in S authenticating the adversarial session.

References

1. Duo Security Two-Factor Authentication. https://goo.gl/wT3ur9.
2. FIDO Universal 2nd Factor. https://www.yubico.com/.
3. Google acquires slicklogin, the sound-based password alternative. https://goo.gl/V9J8rv.
4. Google Authenticator Android app. https://goo.gl/Q4LU7k.
5. Google Cloud Messaging. https://goo.gl/EFvXt9.
6. LinkedIn Confirms Account Passwords Hacked. http://goo.gl/UBWuY0.
7. RSA breach leaks data for hacking securid tokens. http://goo.gl/tcEoS.
8. Sign in faster with 2-Step Verification phone prompts. https://goo.gl/3vjngW.
9. Sound Login Two Factor Authentication. https://goo.gl/LJFkvT.
10. TOTP: Time-Based One-Time Password Algorithm. https://goo.gl/9Ba5hv.
11. Two-factor authentication - authy. https://www.authy.com/.
12. Yahoo Says 1 Billion User Accounts Were Hacked. https://goo.gl/q4WZi9.
13. YubiKeys: Your key to two-factor authentication. https://goo.gl/LLACvP.
14. RFC 4226 HOTP: An HMAC-based One-Time Password Algorithm, 2005. https://goo.gl/wxHBvT.
15. Russian Hackers Amass Over a Billion Internet Passwords, 2014. https://goo.gl/KCrFjS.
16. London Calling: Two-Factor Authentication Phishing From Iran, 2015. https://goo.gl/w6RD67.
17. Hack Brief: Yahoo Breach Hits Half a Billion Users, 2016. https://goo.gl/nz4uJG.
18. SIM swap fraud: The multi-million pound security issue that UK banks won't talk about, 2016. http://www.ibtimes.co.uk/sim-swap-fraud-multi-million-pound-security-issue-that-uk-banks-wont-talk-about-1553035.
19. SMS Deprecated, 2016. https://github.com/usnistgov/800-63-3/issues/168.
20. Over 560 Million Passwords Discovered in Anonymous Online Database, 2017. https://goo.gl/upDqzt.
21. Real-World SS7 Attack - Hackers Are Stealing Money From Bank Accounts, 2017. https://thehackernews.com/2017/05/ss7-vulnerability-bank-hacking.html.
22. M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *Topics in Cryptology - CT-RSA '01*, volume 2020 of *Lecture Notes in Computer Science*. Springer, 2001.
23. T. Acar, M. Belenkiy, and A. Küpçü. Single password authentication. *Computer Networks*, 57(13), 2013.
24. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – Eurocrypt*, 2000.
25. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security*, 1993.
26. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. 2013.
27. X. Boyen. Hidden credential retrieval from a reusable password. In *Proc. of ASIACCS*, 2009.
28. J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. A new two-server approach for authentication with short secrets. In *12th USENIX Security Symp*, 2003.
29. 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*, pages 453–474, 2001.

30. 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*, 2012.
31. W. Ford and B. S. K. Jr. Server-assisted generation of a strong secret from a password. In *WETICE*, pages 176–180, 2000.
32. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology*. 2006.
33. S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.*, 2(3):230–268, Aug. 1999.
34. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO*, pages 273–293, 2012. Also Cryptology ePrint Archive, Report 2011/219.
35. 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*, pages 233–253. Springer, 2014.
36. 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.
37. S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-enhanced password protocols with optimal online-offline protection. In *ASIACCS 2016*, 2016. http://eprint.iacr.org/2015/1099.
38. S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Two-factor authentication with end-to-end password security. In *International Conference on Practice and Theory of Public Key Cryptography (PKC)*, 2018.
39. 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.
40. J. Katz, P. D. MacKenzie, G. Taban, and V. D. Gligor. Two-server password-only authenticated key exchange. In *ACNS*, pages 1–16, 2005.
41. H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *Annual International Cryptology Conference*, pages 546–566, 2005.
42. J. Lang, A. Czeskis, D. Balfanz, M. Schilder, and S. Srinivas. Security keys: Practical cryptographic second factors for the modern web, 2016.
43. 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.
44. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology – CRYPTO*. 2002.
45. P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology - CRYPTO 2002, International Cryptology Conference*, 2002.
46. D. Pointcheval and S. Zimmer. Multi-factor authenticated key exchange. In *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings*, pages 277–295, 2008.
47. N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *Security and Privacy, IEEE Symposium on*, 2006.
48. 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.
49. V. Shoup. ISO 18033-2: An emerging standard for public-key encryption, Dec. 2004. Final Committee Draft.

50. E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Financial Cryptography and Data Security*. 2007.
51. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology - CRYPTO*, number 3621 in Lecture Notes in Computer Science, pages 309 – 326. Springer Verlag, 2005.

A PAKE and DE-PAKE Security Models

We recall the *Device-Enhanced PAKE (DE-PAKE)* security model of [37], which forms a basis of our TFA model, and which extends the PAKE model (also reviewed below) to the case where the user controls an auxiliary device which constitutes the user's second authentication token in addition to the password.

A.1 PAKE Security Model [24]

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 [35] 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 [24], 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 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 corrupt(U), corrupt(S), compromise(S, U), reveal(P, i) or 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_C, 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* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{PAKE}} \leq \frac{q_C + q_S}{|\mathsf{Dict}|} + \epsilon$ *where* q_C *is the number of rogue* send *queries having the target user* U *as recipient*[4] *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.

A.2 DE-PAKE Security Model [37]

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.

[4] The subscript C in q_C indicates rogue queries to the *client* machine used by U; this is in line with the notation used in the TFA case.

The following security definition captures the maximal-attainable online and offline security from a DE-PAKE protocol as informally discussed in the introduction. The attacker's goal is as in PAKE, i.e. to win the test experiment at a user or server instance. The correctness property is also unchanged. However, to the attacker resources we add the number of rogue client sessions (see Section A.1) with the device as the sender, denoted q'_C, and the number of *rogue* device sessions with the client as the sender, denoted q_D. We refer to this more powerful adversary as a DE-PAKE attacker. 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 C or S session and ends its run by outputing bit b'. We say that A *wins* if $b' = b$ where b is the bit chosen by the test session. We define the *advantage of* A *against* DEPAKE as $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} = 2 \cdot Pr[\mathsf{A} \text{ wins against } \mathsf{DEPAKE}] - 1$.

Definition 3. *A DE-PAKE protocol is called $(q_S, q_C, q'_C, q_D, T, \epsilon)$-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_C, q'_C, q_D as defined above)*

1. *If* S *and* D *are uncorrupted, the following bound holds:*
$$\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq \frac{\min\{q_C + q_S, q'_C + q_D\}}{2^d} + \epsilon. \tag{1}$$
2. *If* D *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq (q_C + q_S)/2^d + \epsilon$.
3. *If* S *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathsf{DEPAKE}} \leq (q'_C + q_D)/2^d + \epsilon$.
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.*

Note. The original definition in [37] uses notation q_U instead of q_C to denote the number of rogue queries to the user. Here we use q_C which refers to queries to the *client* machine and is in line with the notation in the TFA context.

Strong KCI Resistance: Discussion. DE-PAKE is intended to provide stronger notion of security in case of server compromise than PAKE. In PAKE the adversary can authenticate to S in case of U-compromise through an offline dictionary attack, but in DE-PAKE this is prohibited. To formalize this requirement we follow the treatment of KCI resistance from [41] and we strengthen the attacker capabilities through a more liberal notion of fresh sessions at a server S. This is why all sessions considered *fresh* in the PAKE model are also considered fresh in the DE-PAKE model, but in addition, in the DE-PAKE model a session Π_i^{S} at server S with peer U is considered fresh *even if queries* $\mathsf{corrupt(S)}$ *or* $\mathsf{compromise(S, U)}$ *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) ensures that this is the case for DE-PAKE secure protocols even when *unbounded* offline attacks against S are allowed.

B Proof of Theorem 1 (Sec. 5): Details of Game Changes

The security proof for Theorem 1 included in Section 5 contains only the high-level description of the game-changes and the claims we make about them. Here we provide all the missing details.

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

Game G_3*:* Note that in the above security game adversary A_2 interacts with game G_2 which internally runs interactive algorithms $\mathsf{SIM}_{\mathsf{SAS}}$ and $\mathsf{F}_{\mathsf{SAS[t]}}$. Note also that the $\mathsf{SIM}_{\mathsf{SAS}}$ algorithm interacts only with $\mathsf{F}_{\mathsf{SAS[t]}}$ on one end and A_2 on the other. We can, therefore, draw the boundaries between 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 $\mathsf{SIM}_{\mathsf{SAS}}$, and a security game G_3 which executes the rest of game G_2, including the operation of functionality $\mathsf{F}_{\mathsf{SAS[t]}}$. Note that G_3 does not execute the SAS-MA protocol, but interacts with A_3 using the $\mathsf{F}_{\mathsf{SAS[t]}}$ interface to $\mathsf{SIM}_{\mathsf{SAS}}$, 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 A_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 $\mathsf{E}_{\mathsf{ACbreak(CD)}}$ 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 $\mathsf{E}_{\mathsf{ACbreak(CD)}}$ 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 $\mathsf{E}_{\mathsf{ACbreak(CD')}}$ 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 $\mathsf{E}_{\mathsf{ACbreak(CD')}}$ 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 $\mathsf{E}_{\mathsf{ACbreak(CS)}}$ 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 $\mathsf{E}_{\mathsf{ACbreak(CS)}}$ 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-MA sub-protocol and delivers D's ciphertext to C in step II then A is forced, by game G_5, to be passive on the C-D link in the DE-PAKE in step III (and if A does not deliver D's ciphertext to C then this D instance will not respond to any further messages, by game G_6). The remaining cases are thus active attacks on the key exchange in step I and the case when A either attacks the SAS-MA sub-protocol and gets D to accept $\mathsf{M_C}* \neq \mathsf{M_C}$ or sends $e_D^* \neq e_D$ to C.

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

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

Game G_{10}*:* Consider the same case of a rogue session Π_i^{S} and a rogue session Π_j^{C} to which the adversary sends zid_i in step I.2, but now consider the possibility that the adversary lets Π_j^{C} get the corresponding z_i but does not learn z_i itself.

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

Note that by the changes done by games G_9 and G_{10}, if the adversary interferes with the KE in step I.1 with session Π_i^{S}, sends zid_i to some Π_j^{C} and does not send it to some Π_l^{D} in a $[\mathsf{SAS.ATTACK}, sid, (\mathsf{pk}^*, zid_i)]$ message for any l then the adversary is forced to be a passive eavesdropper on the DE-PAKE protocol in step III, or otherwise Π_i^{S} will output $\perp$. Note that this is the case when $L(i) = l$ s.t. the game issues $[\mathsf{SAS.SEND}, sid, \Pi_j^{\mathsf{C}}, \Pi_l^{\mathsf{D}}, (\mathsf{pk}, zid_i)]$ 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 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 GenTFA *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 GenTFA *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 GenTFA *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 GenTFA without aborting. A^* performs this emulation by implementing the Authenticated Channel layer as in step III of GenTFA using the corresponding state computed above, i.e. $K_{CS,i}^{\mathsf{S}}, z_i^{\mathsf{S}}$ 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 external 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 $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}}$ 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, $\mathsf{Adv}_{\mathsf{A}^*}^{\mathsf{DEPAKE}}$ 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 GenTFA 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.