Two-Factor Authentication Resilient to Server Compromise Using Mix-Bandwidth Devices

Maliheh Shirvanian
University of Alabama at Birmingham
maliheh@uab.edu

Stanislaw Jarecki
University of California, Irvine
stasio@ics.uci.edu

Nitesh Saxena
University of Alabama at Birmingham
saxena@cis.uab.edu

Naveen Nathan
University of California, Irvine
nnathan@uci.edu

***Abstract*—Two-factor authentication (TFA), enabled by hardware tokens and personal devices, is gaining momentum. The security of TFA schemes relies upon a human-memorable password p drawn from some implicit dictionary D and a t-bit device-generated one-time PIN z. Compared to password-only authentication, TFA reduces the probability of adversary's online guessing attack to $1/(|D| * 2^t)$ (and to $1/2^t$ if the password p is leaked). However, known TFA schemes do not improve security in the face of offline dictionary attacks, because an adversary who compromises the service and learns a (salted) password hash can still recover the password with $O(|D|)$ amount of effort. This password might be reused by the user at another site employing password-only authentication.**

We present a suite of efficient novel TFA protocols which improve upon password-only authentication by a factor of 2^t with regards to *both* the online guessing attack *and* the offline dictionary attack. To argue the security of the presented protocols, we first provide a formal treatment of TFA schemes in general. The TFA protocols we present enable utilization of devices that are connected to the client over several channel types, formed using manual PIN entry, visual QR code capture, wireless communication (Bluetooth or WiFi), and combinations thereof. Utilizing these various communication settings we design, implement, and evaluate the performance of 13 different TFA mechanisms, and we analyze them with respect to security, usability (manual effort needed beyond typing a password), and deployability (need for additional hardware or software), showing consistent advantages over known TFA schemes.

I. Introduction

User authentication is critical to many online (and offline) services. Textual passwords form the most dominant means of user authentication deployed on the Internet today. However, passwords suffer from a number of well-documented security and usability problems [22], [25], [16]. Specifically, passwords are often weak secrets, i.e. low-entropy – short and non-random, due to the human-memorability requirement. As a result, an attacker can build a relatively short dictionary D of likely passwords, which can be used to guess passwords in an online attack. Moreover, it also enables an offline dictionary attack whereby the attacker compromises the service storing (salted) one-way functions (typically hashes) of passwords and recovers these passwords with $O(|D|)$ amount of effort per each password (chosen without high-enough entropy). Such offline dictionary attack is a serious concern, especially in light of frequent attacks against major commercial vendors, such as PayPal [1], LinkedIn [9], and Blizzard [3]. The offline dictionary attacks are attractive for malicious entities because a single server break-in leads to compromising multiple user accounts. Furthermore, since many users re-use their passwords across multiple services, compromising one service typically compromises user accounts at many other services.

To improve the security of password authentication, two-factor authentication (TFA) incorporates user's personal computational device in the authentication process. This device could be a dedicated hardware token (such as RSA SecureID [13]) or a personal gadget, such as a mobile phone (running, for example, the Google Authenticator App [6]). The device creates a short (t-bit) one-time PIN that the user has to copy over to the authentication terminal in addition to providing her password. If the PIN is generated via a pseudorandom function whose key is shared by the device and the server, the probability of attacker's success in an online guessing attack is reduced from $1/|D|$ for password-only systems to $1/(|D|{*}2^t)$. However, existing TFA schemes follow the password-only schemes by storing a (salted) one-way function of the password on the server, and therefore an adversary who compromises the server and learns the password hash, can still recover the password with $O(|D|)$ effort.

In this paper, we set out to improve the security of TFA systems against *both* online guessing attacks *and* offline dictionary attacks. To this end, we design a suite of novel simple and efficient TFA protocols and mechanisms, each offering different security and usability advantages. The idea underlying all our TFA protocols is for the server to store a *randomized* hash of the password, $h = H(p, s)$, and for the device to store the corresponding random secret s. The authentication protocol then checks whether the user types the correct password p *and* owns the device which stores s. Crucially, such protocol must be secure against a lunch-time attack on the device, hence the device cannot just display its secret s to the user. However, it turns out that the secret s can be easily masked with one-time values derived e.g. by a pseudorandom function (PRF) F whose key k is shared between the server and the device. If F_k is computed on a nonce x – e.g. equal to the current time, or chosen as a challenge by the server – the device could output $z = s \oplus F_k(x)$ as its PIN, and the server can verify the (password, PIN) pair (p, z) against the hash $H(p, s)$ by recomputing s as $z \oplus F_k(x)$. Such protocol is $1/(|D| * 2^t)$-secure against online guessing even in the presence of lunch-time attacks on the device and man-in-the-middle attacks on the communication channel between the client and the device. As for an offline dictionary attack after a server corruption, the attacker needs s to verify password guesses, making the off-line dictionary attack time grow to $O(|D| * 2^t)$. Note that the above TFA protocol exposes device secret s (and hence reduces the time of the off-line dictionary attack back to $O(|D|)$) to an attacker who corrupts the server after staging a lunch-time attack on the user's device or eavesdropping on the client-device communication. This motivates our public-key version of this scheme, which forces the dictionary attack time to $O(|D| * 2^t)$ steps even in this case.

The crucial security parameter t in our protocols is bounded by the bit capacity of the device-to-client (D-to-C) channel, i.e. by the bit-length of the PIN. However, the security properties of our TFA protocols depend also on the (existence and the) capacity of the client-to-device (C-to-D) channel, which is not typically used in existing TFA schemes. This motivates exploring different implementations of the D-to-C and C-to-D channels, namely via a manual PIN entry, a visual QR code capture, wireless communication (Bluetooth or WiFi), and combinations thereof. We design, implement and evaluate the performance of the resulting low-, medium-, and full-bandwidth TFA mechanisms based on our TFA protocols, and compare them with respect to security (i.e. primarily the length t of the PIN z and of the device secret s), usability (i.e. the manual effort involved beyond typing a password), and deployability (i.e. whether additional hardware or software is needed). These mechanisms typically provide much stronger security properties than existing TFA schemes, at modest usability costs. Interestingly, one of our mechanisms provides 2^t factor improvement in security over traditional TFA mechanisms but adds *no extra cost* in usability and deployability, and is therefore ready for immediate deployment.

Our Contributions: The main contributions brought forth by our paper are summarized below:

1. TFA Formalization (Section III): We provide what we believe is the first formal treatment of two-factor authentication, modeling different forms of TFA attacks, including offline and online attacks, eavesdropping and man-in-the-middle attacks against the client-device communications, and lunch-time attacks against devices.

2. Novel TFA Protocols (Section IV): We design four novel protocols for two-factor authentication resilient to server compromise. One of these protocols is time-based (in line with existing TFA protocols). The other three are challenge-based, one involving symmetric-key encryption and the other two public-key encryption. The strength of these protocols is their security, simplicity, efficiency, and broad applicability to a wide range of devices, e.g. traditional phones, smartphones, smart watches, dedicated security tokens, and more.

3. Mix-Bandwidth Device TFA Mechanisms (Section V): Based on these protocols, we design different TFA mechanisms enabled by a wide range of mix-bandwidth communication channels that can be established between the device and the client browser. Specifically, our TFA mechanisms are based on (1) unidirectional and bidirectional low-bandwidth (t = 19 bits, i.e. 6 decimal digits) channels formed via manual PIN entry or QR codes; (2) bidirectional mid-bandwidth channel formed by QR codes (t between 19 and 128 bits); (3) bidirectional full-bandwidth Bluetooth or point-to-point WiFi channel (t = 128 or more), and combinations thereof. This results in a total of 13 TFA variants, offering different security guarantees and usability advantages. Also, to our knowledge, this is the first use of point-to-point WiFi and bidirectional QR codes as a means of proximity-based communication for the purpose of authentication.

4. Implementation and Evaluation (Sections V and VI): We provide the server-side (PHP scripts), client-side (Chrome browser extensions for the full-bandwidth model) and device-side (Android app) implementation of all of these TFA mechanisms, and estimate their performance quantitatively in terms of the login time as well as qualitatively in terms of security, user effort and deployability.

II. Related Work

The most common form of TFA employs hardware tokens, like RSA SecureID [13], which are specialized devices used solely for the purpose of authentication. Typically, a unique token is needed to authenticate to each service, so the user needs to carry n tokens with her to enable authentication to n services, which does not scale well. Moreover, due to the need for specialized tokens, provisioning of tokens might become difficult as well as costly. Although our proposed TFA mechanisms are geared for soft token deployment, they are equally suitable for hardware tokens with various communication capabilities, e.g., presence of a screen, a wireless interface, or an ability to directly connect to a personal mobile device such as a smartphone [14].

Key advantages of soft tokens over hardware tokens include scalability and flexibility (single personal device can be used with multiple services) as well as cost savings (provisioning soft tokens is logistically much simpler). Many commercial soft token TFA schemes are available, including Google Authenticator [6], Duo Security Two-Factor Authentication [5], Celestix's HOTPin [4] and Microsoft's PhoneFactor [10]. These tokens essentially use the same time-based cryptographic protocol to generate one-time passwords as the hardware tokens. As in the case of TFA's employing hardware tokens, all these schemes store hashed passwords on the server, which means that an attacker who compromises the server can recover passwords with $O(|D|)$ offline effort.

PhoneAuth [19] is a recent academic (soft token) TFA scheme. In contrast to traditional TFA, PhoneAuth does not use a low-bandwidth manual channel (to transfer one-time password between the phone and client), but rather employs a full-bandwidth Bluetooth channel over which a cryptographic authentication protocol is run between the phone and server (with client as the router). This is similar to our FBD TFA variants that use Bluetooth communication. PhoneAuth, however, provides the same weak level of resistance to offline dictionary attacks as the other TFA schemes.

Other authentication approaches have been proposed in the literature that aim to strengthen password-authentication by leveraging a personal device (but not as a second factor). MP-Auth [20] leverages a phone to improve the security of password authentication when used on a potentially malicious client terminal connected with the phone over Bluetooth. Phool Proof's [23] goal is to prevent phishing by involving the phone in the authentication process. MP-Auth and Phool Proof store hashed password on the servers and thus provide only $O(|D|)$ level of security against offline dictionary attack. PIN-Audio [24] and Tapas [21] use the phone as a mobile password manager that is used to store long and random passwords. These two schemes make the offline dictionary attacks infeasible because passwords are no longer human-memorable and a dictionary attack is not viable. However, they do not provide two-factor authentication (e.g., if the password is phished, there would be no security). A different example of using camera-equipped personel device for strengthening user security was previously given in [18] in the context of protecting password entry on an untrusted, potentially compromised, terminal.

III. Background and Models

We describe the setting of a Two-Factor Authentication scheme TFA, we formalize the model of such scheme, and we define the security properties such scheme should have.

Participants. A TFA scheme involves three computational entities, the *Server* S, the *Client* C, and the *Device* D, in addition to the human user U. Server S runs a web service accessed by users identified by user names. Client C is a web browser used by a user U, who wants to authenticate to the service under her user name. Device D is a hand-held personal device which belongs to the same user U, and which U will use as a secondary "security token" in order to authenticate her web session to S. D can be any programmable device capable of storing data, keeping a persistent clock, performing computation, and displaying characters on a screen. We also consider TFA schemes relying on additional properties of device D, namely assuming that D can take photographs *or* that it can communicate wirelessly with a laptop, e.g. over Bluetooth or WiFi. All these assumptions are met by smart phones, which are the primary targets of our TFA schemes. However, these conditions are satisfied by several other personal electronic devices, e.g. an e-book reader or a smart watch. Finally, D could also be implemented as a dedicated hardware token, similar to e.g. RSA SecurID.

Computation and Communication Models. We assume that C and S communicate over open internet, but we assume that they can rely on a Public Key Infrastructure (PKI) for establishing a secure channel with one-sided authentication of the server S by the client C. In other words, we assume that S has a public key with an SSL certificate signed by a certification authority recognized by the client browser C, and that C and S establish an SSL connection via a TLS handshake using S's certificate. We assume that the code followed by C in a TFA protocol is either an HTML page which C receives from S over this SSL connection, or C executes a code of a browser plug-in which was installed from a trusted source. Note that the PKI assumption is a standard way of securing communication between web services and their users. (At the end of Section IV we point out that the trust in the PKI can be somewhat relaxed if C is implemented as a browser plug-in.)

While C and S communicate over the internet without particular concern about protocol bandwidth, by contrast we assume that device D might have heavily restricted communication abilities. We consider four types of devices, depending on the restrictions on the C-to-D and D-to-C communication channels. Type I is a device like those used by traditional TFA schemes such as Google's authenticator smart phone app. Namely, D cannot receive any message from C during protocol execution, and it can send a single response message resp to C which must be *short*, e.g. up to 20 or 30 bits. (In addition, we assume that device of type I can maintain a clock, or other updatable state e.g. a counter.) Devices of type II,III, and IV, can receive a single challenge message ch from C and reply with a single response resp. For device of type II message ch is *medium-sized*, e.g. between 100 and 2000 bits, while resp is short; for device of type III both ch and resp are medium-sized, and for device of type IV both ch and resp can be *long*, e.g. several thousand bits would still be fine. As we explain below, some of these D-to-C and C-to-D channels are authenticated by a human in the loop to verify that the PIN which C receives is the same PIN that D sent. However, this is not the case for all device types, and so in our security security model, we allow a man-in-the-middle attack on these channels (in addition to an eavesdropping attack); and all of our protocols (Section IV) and mechanisms implementation (Section V) are secure in the presence of such attacks.

Motivating Scenarios of Four Device-Client Interaction Types. These four device types are motivated by different implementation scenarios which make different software and hardware demands on both the device D *and* on the client C, and which make different demands on the way the user U enables the communication between C and D. Device of **type I** does not need any data connectivity, except for a way to periodically synchronize its internal clock (or counter). D must only have a small screen on which it can display its message resp encoded into a short numerical or alphanumerical string, i.e. a PIN. The user U will have to read this PIN and type it into the client browser C. Therefore this D-to-C channel is low-bandwidth, it is authenticated by the human in the loop, but it can be eavesdropped upon by a "shoulder surfer".

Device of **type II** models e.g. a camera phone which can photograph the browser's screen display. In this case C could display message ch encoded using a Quick Response (QR) code, user U could position device D until this QR code is detected and photographed by D's camera, D can decode the QR code into message ch, and reply to C with a PIN resp in the same way as device of type I. While a QR code can encode even several thousand bits, ensuring good reliability with a budget camera phone probably precludes ch longer than 2Kb. We call this bandwidth *medium* because whereas it can be much longer than the PIN we call *short*, the lower it is the better reliability/usability characteristics of the system. As in the case of a PIN D-to-C channel, this channel is authenticated by the human in the loop, but subject to eavesdropping.

Device of **type III** has the same C-to-D capability as the device of type II, but the medium-sized bandwidth on the D-to-C channel can be implemented e.g. with a larger display on D on which it can display a QR-encoded response resp, and with client C which has a front-facing camera, e.g. because the browser is running on a laptop, which can be accessed from the HTML on the browser, or because C is a browser plug-in. This D-to-C channel is also medium-sized, humanly authenticated, and subject to eavesdropping. Finally, device of **type IV** is capable of higher bandwidth communication with the client C, e.g. via a WiFi or a Bluetooth channel. While such channels could be authenticated by shared keys established in the initialization process, having to bind device D with each client terminal is not user-friendly, moreover any shared key permanently residing on a browser would be subject to attacks. We will therefore assume that such channel could be subject to both eavesdropping and man-in-the-middle attacks.

TFA **Protocol Syntax and Execution.** A TFA scheme consists of an algorithm Init and a three-party protocol $\mathsf{Auth} = (\mathsf{Auth}_\mathsf{S}, \mathsf{Auth}_\mathsf{C}, \mathsf{Auth}_\mathsf{D})$ executed between parties S, C, and D. We assume that server S keeps users' authentication data in a table indexed by user names. (Since we assume PKI, S also has a public key pair and a certificate, but in the TFA model we simply assume that the C-S communication goes over a secure channel on which S is authenticated to C.) Algorithm Init is executed separately for each user U on input a security parameter 1^τ, on an additional parameter t which fixes the upper-bound on the bit-length of D's response message resp, and on password p. Algorithm $\mathsf{Init}(1^\tau, \mathsf{t}, \mathsf{p})$ outputs a pair $(\mathsf{st}_\mathsf{S}, \mathsf{st}_\mathsf{D})$, where st_S will be kept by server S under U's user name UN, and st_D is securely loaded onto device D that belongs to U. Note that we assume that the client C does not have any permanent state. In our implementations (see Section V), C is either a browser which downloads the authentication protocol code Auth_C it follows from the server S every time the authentication protocol executes, or it is a browser plug-in, but the cryptographic model assumes a state-free client.

Whenever user U wants to authenticate to S from her browser C, C and S establish a secure connection using S's public key certificate, and over this connection U specifies her user name UN to S.[1] S retrieves st_S indexed by UN and executes the interactive algorithm Auth_S on local input st_S, communicating with C which executes an interactive algorithm Auth_C on local input p. In addition, at some point in its interaction with S, algorithm Auth_C can generate a single message ch which will be received by D, and receive a single response resp from D s.t. $|\mathsf{resp}| \leq t$, computed by algorithm Auth_D on inputs $(\mathsf{ch}, \mathsf{st}_\mathsf{D})$. (For device type I C's message ch to D is fixed as a special sign $\perp$ which triggers D to compute response resp but carries no further information.) Finally, algorithm Auth_S outputs a bit b which designates whether S authenticates this user or not. The correctness requirement is that for all τ, p, t values, if $(\mathsf{st}_\mathsf{S}, \mathsf{st}_\mathsf{D}) \leftarrow \mathsf{Init}(1^\tau, \mathsf{t}, \mathsf{p})$ then bit b output by Auth_S in an execution of $(\mathsf{Auth}_\mathsf{S}, \mathsf{Auth}_\mathsf{C}, \mathsf{Auth}_\mathsf{D})$ on respective local inputs $(\mathsf{st}_\mathsf{S}, p, \mathsf{st}_\mathsf{D})$, with all messages delivered between the parties correctly, will be equal to 1 except for probability negligible in τ.

Notes on the Timing Assumption. A TFA scheme can additionally rely on a synchronized clock between device D and server S, in which case algorithms Auth_S and Auth_D take an additional time-encoding input, resp. T_S and T_D, and the correctness property guarantees that $b = 1$ only if $T_\mathsf{S} = T_\mathsf{D}$. We say that such TFA scheme is *time-based*. Alternatively to the timing assumption, a TFA scheme could assume that both D and S have an updatable storage, which in the simplest (but sufficient) case can be a strictly increasing counter. Such TFA scheme can be called *counter-based*, where the correctness requirement would guarantee authentication only if S and D execute on the synchronized counter. We will not formally model the security of counter-based TFA schemes, but it is a plausible alternative to a time-based scheme.

[1]In most of our implementations U actually does not have to send UN to S in this first message. See the note at the end of Section IV.

TFA **Adversarial Model.** The adversary Adv who attacks a (two-factor) password authentication system can have two distinct goals. The first goal is an *authentication attack* where Adv breaks into the account of some innocent user U by successfully authenticating to S under U's user name. The second goal is a *password recovery attack* where Adv learns U's password, for example in order to re-use it in some other system where U uses the same password, as many real-life users are known to do. The first attack is sometimes called an *on-line attack*, and the second an *off-line dictionary attack*, but this terminology could be misleading because in both cases the attacker has on-line access to the participating parties.

The adversarial ability to stage either attack must be considered in a scenario which models adversary's ability to access, eavesdrop on, and even learn the private data of some participating parties by a local corruption. Furthermore, we must differentiate between (1) the case of a party corruption (active or passive), where adversary (perhaps temporarily) "resides" on this protocol party, steals its local data, and this party becomes either an eavesdropping or a malicious agent of the attacker, and (2) the case when adversary corrupts a party, passively learns it's private data, and then leaves. For the second case we will consider leakage of the server's data st_S, the device's data st_D, and the user's password p. However, for the case of player corruptions we will not consider active corruptions of the server because we are in the PKI model where the client trusts the server S it authenticates via PKI certificates. In particular in all our TFA schemes if an adversary does corrupt the server then it can learn the passwords of all users who authenticate to the server while such corruption lasts. Similarly we assume that client C runs a trusted code (in practice C often downloads this code from the same PKI-authenticated trusted server S). Indeed, if C runs a corrupted code then the adversary can steal the password of C's user U. However, we do consider an active corruption of the user's device D, which we model by letting the adversary steal D's data and perform a man-in-the-middle attack on the C-D communication.

We will make two further simplifying assumptions in the security model: First, we will formally consider the password recovery attack only for the adversary that (passively) corrupts the server S. Conversely, we will formally consider an authentication attack only if the adversary does not corrupt S. However, it is easy to see that in all our TFA schemes the adversary's probability to recover the user's password without corruption of the server is the same as the probability of staging an authentication attack. Secondly, in all of our schemes an adversary who corrupts the server can stage an authentication attack either with the same probability as the adversary that does not, or it has to perform the same off-line computation as we formally argue for the password recovery attack.

In our security notions below we consider the following execution of an *authentication game*, denoted $\mathsf{Auth}_{\mathsf{TFA},\mathsf{Adv}}$, which takes as input a tuple $(\tau, t, d, D, q_{\mathsf{S}}, q_{\mathsf{C}}, q_{\mathsf{D}})$ (all w.l.o.g. known to algorithm Adv) where D is an arbitrary set of size 2^d and t is the bit-length of the device response resp in the TFA scheme, and executes as follows:

1) First p is chosen uniformly at random in D and $\mathsf{Init}(1^\tau, \mathsf{t}, \mathsf{p})$ is executed to generate $(\mathsf{st}_{\mathsf{S}}, \mathsf{st}_{\mathsf{D}})$.
2) Adv can make q_{C} *client session* queries, i.e. it can interact with q_{C} instances of the algorithm $\mathsf{Auth}_{\mathsf{C}}$. Each instance of $\mathsf{Auth}_{\mathsf{C}}$ will run on input p and locally interact with algorithm $\mathsf{Auth}_{\mathsf{S}}$ running on input st_{S} and algorithm $\mathsf{Auth}_{\mathsf{D}}$ running on input st_{D}. Adv does not see the messages passed between $\mathsf{Auth}_{\mathsf{C}}$ and $\mathsf{Auth}_{\mathsf{S}}$, but it does see the messages passed between $\mathsf{Auth}_{\mathsf{C}}$ and $\mathsf{Auth}_{\mathsf{D}}$, and it learns a bit b output by $\mathsf{Auth}_{\mathsf{S}}$, which models the fact that a network adversary can tell whether the server authenticates the user by observing the C-S traffic. Additionally, Adv has an option to replace message resp sent by $\mathsf{Auth}_{\mathsf{D}}$ to $\mathsf{Auth}_{\mathsf{C}}$ with a message of Adv's choice, in which case we call such client session *hijacked*. (Modifying the C-to-D message can be done via a device session query, see below.) If Adv does not modify message resp we call such client session *eavesdropped.*
3) Concurrently, Adv can make q_{D} *device session* queries, i.e. it can interact with q_{D} instances of the algorithm $\mathsf{Auth}_{\mathsf{D}}$ running on local input st_{D}.
4) Concurrently, Adv can make q_{S} *server session* queries, i.e. it can interact with q_{S} instances of the algorithm $\mathsf{Auth}_{\mathsf{S}}$ running on local input st_{S}. We distinguish between two types of server sessions: We call it *network-only* if *Adv* completes it without an interaction with any $\mathsf{Auth}_{\mathsf{D}}$ session, and we call it *with-device* if between the moment it is triggered and the moment that session completes *Adv* interacts with any device session $\mathsf{Auth}_{\mathsf{D}}$.
5) Adv can make *server*, *device*, and *client leakage* queries, on which it receives resp. st_{S}, st_{D}, and p.
6) Finally Adv outputs a bit-string p^* and the experiment ends. We define the following two events:
   ◦ $\mathsf{Succ}_{\mathsf{P}} = 1$ iff $p^* = p$ [a password recovery attack];
   ◦ $\mathsf{Succ}_{\mathsf{A}} = 1$ iff $\mathsf{Auth}_{\mathsf{S}}$ outputs $b = 1$ on any server session query [an authentication attack].

If the TFA scheme is time-based we assume that throughout the authentication game S and D execute $\mathsf{Auth}_{\mathsf{S}}$ and $\mathsf{Auth}_{\mathsf{D}}$ protocols on T_{S} and T_{D} values which are equal to a global time counter T, which starts from 1 and is incremented e.g. every time the adversary triggers the $\mathsf{Auth}_{\mathsf{S}}$ protocol session. If an adversary has lunch-time or viral access to D and can move its clock forward or backward, this is equivalent to Adv being able to make a "with-device" $\mathsf{Auth}_{\mathsf{S}}$ session.

Note that the above authentication game allows Adv to interact q_{S} times with the server S as a purported user client, it allows Adv up to q_{D} lunch-time accesses to the device D, and it enables Adv to witness or interact with q_{C} executions of user U running an authentication protocol on an honest client C with the server S. In such execution Adv has an eavesdropping access to the C-D communication, but Adv can also stage a man-in-the-middle attack between C and D, e.g. if it tricks the user to use a modified device which runs some other code than $\mathsf{Auth}_{\mathsf{D}}(\mathsf{st}_{\mathsf{D}})$, or if the C-D communication goes over a wireless medium, and there is either no keys established between these devices or if Adv learns these keys via a virus or a lunch-time attack on C or D.

TFA **Security Properties.** As is standard for password authentication schemes, we define security of a TFA scheme assuming that the user chooses her password p uniformly at random from some dictionary set D of size 2^d. Since users are known to pick passwords with only moderate entropy, we must assume D is medium-sized, e.g. d is no more than 20 or 30, and in particular that it is feasible, and indeed easy, for an adversary to iterate through the dictionary D. Another crucial parameter for the security of a TFA scheme turns out to be the bandwidth t on the D-to-C channel, i.e. the bit-length of D's response resp. We define the following two security notions:

Definition 1 (Authentication-Attack Resistance): We call a TFA scheme $(\mathsf{Init}, \mathsf{Auth})$ $(T, \delta_{\mathsf{N}}, \delta_{\mathsf{D}}, \delta_{\mathsf{C}})$*-authentication-attack resistant for parameters* $(\tau, t, d, q_{\mathsf{S}}, q_{\mathsf{C}}, q_{\mathsf{D}})$ if for any D of size 2^d and any algorithm Adv whose running time is bounded by T the following holds for random execution of an authentication game $\mathsf{Auth}_{\mathsf{TFA},\mathsf{Adv}}(\tau, \mathsf{t}, \mathsf{d}, \mathsf{q}_{\mathsf{S}}, \mathsf{q}_{\mathsf{C}}, \mathsf{q}_{\mathsf{D}})$:

1) $\Pr[\mathsf{Succ}_{\mathsf{A}}] \leq \mathsf{q}_{\mathsf{S}} \cdot \delta_{\mathsf{N}}$ assuming that Adv is a network adversary, i.e. it is precluded from server leakage, client leakage, device leakage, and any with-device server session queries;
2) $\Pr[\mathsf{Succ}_{\mathsf{A}}] \leq \mathsf{q}_{\mathsf{S}} \cdot \delta_{\mathsf{D}}$ assuming that Adv is precluded from server leakage and client leakage queries (but w.l.o.g. Adv makes device leakage or with-device server session queries);
3) $\Pr[\mathsf{Succ}_{\mathsf{A}}] \leq \mathsf{q}_{\mathsf{S}} \cdot \delta_{\mathsf{C}}$ assuming that Adv is precluded from server leakage, device leakage, and with-device server session queries (but w.l.o.g. Adv makes a client leakage query).

We call scheme TFA $(\delta_{\mathsf{N}}, \delta_{\mathsf{D}}, \delta_{\mathsf{C}})$*-authentication-attack resistant* for parameters (t, d) if for all polynomials $T(\tau)$, $q_{\mathsf{S}}(\tau)$, $q_{\mathsf{C}}(\tau)$, $q_{\mathsf{D}}(\tau)$ there exists a negligible function $\epsilon(\tau)$ s.t. for every τ, TFA is $(T(\tau), \delta_{\mathsf{N}} + \epsilon(\tau), \delta_{\mathsf{D}} + \epsilon(\tau), \delta_{\mathsf{C}} + \epsilon(\tau))$-authentication-attack resistant for parameters $(\tau, t, d, q_{\mathsf{S}}(\tau), q_{\mathsf{C}}(\tau), q_{\mathsf{D}}(\tau))$.

In other words, in all authentication attacks we assume the secret st_{S} stored by the server S *does not* leak to the adversary, and we upper-bound the probability of the authentication attack per each server session in the following three cases: (1) δ_{N} is the attack probability without leaks from either C or D, and while Adv can get a lunch-time access to D, and even play a man-in-the-middle adversary on the C-D channels during U's authentication sessions, we assume that Adv cannot attempt to authenticate to S during the time Adv has access to D; (2) δ_{D} is the attack probability without leaks from C, but Adv can learn D's secret and/or it can attempt to authenticate to S while it has access to device D; (3) δ_{C} is the attack probability without leaks from D, but Adv can learn U's password, e.g. because it was cashed on C to which Adv gained access, or because Adv has learned it from some other authentication system where U re-used her password.

Definition 2 (Password-Recovery Resistance): We call a TFA scheme $(\mathsf{Init}, \mathsf{Auth})$ $(T, \epsilon, \bar{T}, n_{\mathsf{N}}, n_{\mathsf{D}})$*-password-recovery* *resistant for parameters* $(\tau, t, d, q_\mathsf{S}, q_\mathsf{C}, q_\mathsf{D})$ if for any D of size 2^d and any algorithm Adv, the following holds for a random execution of an authentication game $\mathsf{Auth}_{\mathsf{TFA},\mathsf{Adv}}(\tau, \mathsf{t}, \mathsf{d}, \mathsf{D}, \mathsf{q_S}, \mathsf{q_C}, \mathsf{q_D})$, for any γ:

1) If Adv is prevented from making client or device leakage queries (but w.l.o.g. Adv makes a server leakage query), and from making any device session queries after the server leakage query, and if Adv's time is limited by $\min(T, \gamma \cdot n_\mathsf{N} \cdot \bar{T})$, then $\Pr[\mathsf{Succ_P}] \leq \gamma + \epsilon$.
2) If Adv is prevented from making client leakage query (but w.l.o.g. Adv makes server leakage and device leakage queries), and if Adv's time is limited by $\min(T, \gamma \cdot n_\mathsf{D} \cdot \bar{T})$, then $\Pr[\mathsf{Succ_P}] \leq \gamma + \epsilon$.

For any function $\bar{T}$ of the security parameter, we call scheme TFA $(\bar{T}, n_\mathsf{N}, n_\mathsf{D})$*-password-recovery resistant* for parameters (t, d) if for all polynomials $T(\tau)$, $q_\mathsf{S}(\tau)$, $q_\mathsf{C}(\tau)$, $q_\mathsf{D}(\tau)$ there exists a negligible function $\epsilon(\tau)$ s.t. for every τ, TFA is $(T(\tau), \epsilon(\tau), \bar{T}(\tau), n_\mathsf{N}, n_\mathsf{D})$-password-recovery resistant for parameters $(\tau, t, d, q_\mathsf{S}(\tau), q_\mathsf{C}(\tau), q_\mathsf{D}(\tau))$.

We call scheme TFA $(\bar{T}, n_\mathsf{N}, n_\mathsf{D})$*-password-recovery resistant without adversarial device access* if it satisfies the above under an additional restriction on Adv in item (1), namely that Adv is also prevented from any device and client session queries (recall that client session query allows Adv to eavesdrop on C-D traffic). Note that these restrictions cut Adv off from *all* queries in the authentication game, except for the server leakage query which gives $\mathsf{st_S}$ to Adv.

In other words, we define password-recovery resistance in terms of some "base time" function $\bar{T}$ of the security parameter, s.t. the only way an adversary can find a (randomly chosen) password p of an honest user, given state $\mathsf{st_S}$ leaked from the server, is to do one of the following: (1) Adv can search through γ fraction of guesses, where each guess can be tested at cost at most $\bar{T}$, thus lower-bounding adversary's time by $\gamma \cdot n \cdot \bar{T}$, and upper-bounding the probability of finding p by γ; (2) Adv can break some underlying cryptographic assumption which could speed-up his search by leaking some additional information on p or $\mathsf{st_D}$, hence the additional probability term ϵ which should be negligible (in the security parameter) for any polynomial (in the security parameter) time bound T.

The parameters n_N and n_D designate the size of different spaces for which Adv performs a brute-force attack: If Adv is a network attacker in the sense that it leaks $\mathsf{st_S}$ from server S, but does not leak $\mathsf{st_D}$ from device D (and does not perform active attacks against the device D after leaking $\mathsf{st_S}$), Adv's search space should be lower-bounded by n_N, and if Adv leaks $\mathsf{st_S}$ and either leaks $\mathsf{st_D}$ or performs an active attack against D after leaking $\mathsf{st_S}$, its search space will be n_D.

The notion of password-resistance without adversarial device access reflects a (small) weakness of two TFA schemes we present below, TFA-T and TFA-SC, where an adversary who corrupts the server can recover in $2^d \cdot \bar{T}$ time the passwords of those users for whom it either eavesdrops on their client-device sessions or otherwise gains access to their personal device D.

TFA **Security: Fundamental Limits.** The most interesting feature of a two-factor authentication scheme is how it can strengthen the security of plain password authentication. First, note that any TFA scheme can be at most $(1/2^{d+t}, 1/2^d, 1/2^t)$-authentication-attack resistant. This is easy to see: No matter what the probability distribution of message resp is in an Auth protocol instance on inputs $(\mathsf{st_S}, p, \mathsf{st_D})$, a network adversary who executes $\mathsf{Auth_C}$ on (p^*, resp^*) chosen uniformly in $D \times \{0,1\}^t$ will make $\mathsf{Auth_S}(\mathsf{st_S})$ accept with probability at least $1/(|D| \cdot 2^t) = 1/2^{d+t}$. Similarly, an adversary who corrupts D and learns $\mathsf{st_D}$, or has access to D while interacting with some $\mathsf{Auth_S}(\mathsf{st_S})$ instance, can execute $\mathsf{Auth_C}$ on p^* chosen uniformly in D, and will make $\mathsf{Auth_S}(\mathsf{st_S})$ accept if $p^* = p$ which holds with probability $1/|D| = 1/2^d$. Finally, an adversary who learns password p can always succeed with probability at least $1/2^t$ by running $\mathsf{Auth_C}$ on p and resp^* chosen uniformly in $\{0,1\}^t$.

Secondly, we argue that any TFA scheme that achieves optimal authentication-attack resistance of $(1/2^{d+t}, 1/2^d, 1/2^t)$ can have at most $(2^{d+t}, 2^d)$-password-recovery resistance. Note that for a random $\mathsf{Auth_S}$ generated by Init on a random p, there should be only one pair (p, resp) in $D \times \{0,\}^t$ s.t. $\mathsf{Auth_S}(\mathsf{st_S})$ accepts in interaction with $\mathsf{Auth_C}$ on input p and D's response resp. Otherwise Adv would successfully authenticate to S with probability higher than $1/2^{d+t}$ by running $\mathsf{Auth_C}$ on random (p^*, resp^*) in $D \times \{0,1\}^t$. Consequently, given st, adversary Adv can test a guessed password p^* by executing $\mathsf{Auth_S}(\mathsf{st_S})$ interacting with $\mathsf{Auth_C}(\mathsf{p}^*, \mathsf{resp}^*)$ on random resp^*, because this test will succeed, with high enough probability, only if $p^* = p$. Thus if $\bar{T}$ is the time to execute $(\mathsf{Auth_S}, \mathsf{Auth_C})$ as above, Adv can find the correct p (for a random p, except for negligible probability) with probability γ, for any γ, by testing a γ fraction of the $D \times \{0,1\}^t$ search space, and the time of such attack is upper-bounded by $\gamma \cdot 2^{d+t} \cdot \bar{T}$, thus n_N can be at most 2^{d+t}. Clearly, if Adv learns $\mathsf{st_D}$, it can run $\mathsf{Auth_D}(\mathsf{st_D})$ to compute resp instead of guessing it, thus reducing the search space to just the password dictionary D, hence n_D can be at most 2^d.

These parameters should be contrasted with the security bounds achievable by plain password authentication, which are $1/2^d$ for an on-line authentication attack and 2^d for password recovery given $\mathsf{st_S}$. A two-factor authentication can improve both, respectively to $1/2^{d+t}$ and 2^{d+t} given a personal device with just t bits of output bandwidth. Moreover, such scheme could and should gracefully degrade to the security of a plain password scheme if an adversary gets hold of this personal device. Note that the simplistic TFA schemes available today do achieve $1/2^{d+t}$ on-line attack resistance, but their password-recovery resistance given $\mathsf{st_S}$ is unnecessarily just 2^d. As we show, the optimal password-recovery resistance bound of 2^{d+t} can be achieved with inexpensive TFA schemes which are easy to deploy and to operate, without doing any harm to the active attack resistance property.

Finally, we note that a TFA scheme could make recovering a user's password given the leaked server's state $\mathsf{st_S}$ not only harder, i.e. requiring 2^{d+t} instead of 2^d off-line tests, but actually impossible. However this can come only at a price of weakening the $1/2^{d+t}$ bound on the on-line authentication attack. Consider for simplicity of the argument $D = \{0,1\}^d$ and $t = d$. In this case we could make $\mathsf{st_D}$ a random string s in $\{0,1\}^d$, $\mathsf{st_S}$ could be a string $\sigma = s \oplus p$, and the authentication algorithm would check if C (on input p), given access to D (on input s), can recover the same string $\sigma = p \oplus s$ which is held by S. In such scheme password p is effectively secret-shared between S and D, hence corruption of S (or D) leaks no information about p, and thus makes password recovery impossible, but the on-line authentication attack resistance of such protocol is only $1/2^d$, because Adv could authenticate by guessing the correct d-bit value σ. Trading on-line authentication attack resistance for password-recovery resistance in case of server's memory leakage does not seem to be a good deal, but it *is* a good deal to increase password-recovery resistance without damaging on-line authentication attack resistance, as is the case for the TFA schemes we present in this paper.

IV. Protocols and Security Analysis

We describe four TFA protocols, each of which can be executed given any bandwidth limit of t bits for the response message resp flowing from the device D to the client C. The first protocol, TFA-T, which stands for *time-based* TFA protocol, is applicable to all device types, including devices of type I, which can receive no input during the TFA protocol execution, but which rely on a clock synchronized with the server (or, alternatively, on a counter). The second and third protocols, TFA-SC, and TFA-PC, which stands respectively for *symmetric-key* and *public-key* TFA protocols, are applicable for devices of type II and higher, because they do require D to receive a single challenge message ch from C to D. The fourth protocol, TFA-PC-Alt, is a variant of protocol TFA-PC, which reduces the bandwidth requirements of TFA-PC by replacing public key encryption with key encapsulation with some special properties which happen to be satisfied by hashed ElGamal in the random oracle model. Using reasonable cryptographic parameters, protocols TFA-SC, TFA-PC, and TFA-PC-Alt, require the C-to-D message ch to take respectively 80, 344, and 196 bits.

All protocols achieve optimal authentication-attack resistance, except that the security of the TFA-T protocol degrades to password-only authentication if the adversary can shift the internal clock of Alice's personal device D, e.g. by a virus, or because of an occasional access to D, or because of a human-engineering attack which modifies causes Alice to erroneously shift the clock on D. Moreover, protocols TFA-T and TFA-SC achieve optimal password-resistance only in its weaker form, i.e. without adversarial device access, while the public-key based protocols TFA-PC and TFA-PC-Alt remove that weakness as well. In the summary, taking D-to-C channel capacity as a constant, the security of the TFA protocols we present increases with the growing demands on the C-to-D communication channel.

Time-Based TFA Protocol. The TFA-T protocol (Init, Auth) assumes a Collision-Resistant Hash (CRH) function H (modeled as a random oracle) and a Pseudorandom Function (PRF) F. Given parameters (τ, t), H maps onto 2τ-bit strings, and for $|k| = \tau$, F_k maps domain $\{0,1\}^\tau$ onto range $\{0,1\}^t$. For $t \leq \tau$ PRF F could be implemented as the first t bits of an output of a block cipher. For $t > \tau$ one can implement F e.g. using a CBC-mode block cipher cascade. Since TFA-T is a time-based protocol, $\mathsf{Auth_S}$ and $\mathsf{Auth_D}$ take additional time input encoded as τ-bit string, denoted resp. T_S and T_D.

Time-Based TFA Scheme TFA-T

Init$(1^\tau, t, p)$: Pick $s \leftarrow \{0,1\}^t$ and $k \leftarrow \{0,1\}^\tau$, compute $h = H(p, s)$, and set $\mathsf{st_D} = (s, k)$ and $\mathsf{st_S} = (h, k)$.

Protocol Auth:
(assuming secure C-S channel s.t. S is authenticated to C)

1) Device D on input $\mathsf{st_D} = (s, k)$ and time T_D computes $r = F_k(T_D)$ and $z = s \oplus r$ and sends $\mathsf{resp} = z$ to C.
2) Client C on input p and D's message $\mathsf{resp} = z$, sends (p, z) on the secure channel to S.
3) Server S on input $\mathsf{st_S} = (h, k)$, time T_S, and C's message (p, z), computes $r = F_k(T_\mathsf{S})$, and accepts if and only if $h = H(p, z \oplus r)$.

Theorem 1: If F is a secure PRF and H is a Random Oracle then TFA-T is $(1/2^{t+d}, 1/2^d, 1/2^t)$-authentication-attack resistant and $(\bar{T}_H, 2^{t+d}, 2^d)$-password-recovery resistant without adversarial device access for parameters (t, d), where $\bar{T}_H$ is the time required to compute H on any input.

Proof: We first argue authentication-attack resistance. Note that by the security of PRF security of F without server or device leakage, which reveal k to Adv, probability $\mathsf{Succ_A}$ remains negligibly close (in τ) if we replace $r_T = F_k(T)$ values by random t-bit strings. Note also that by CRH security of H, probability $\mathsf{Succ_A}$ remains negligibly close if S stores (p, s) instead of $h = H(p, s)$ and the acceptance bit $b = 1$ on any $\mathsf{Auth_S}$ session is decided by checking if (p', z') received by $\mathsf{Auth_S}$ is equal to $(p, s \oplus r_T)$. Finally, if z_T is defined as $s \oplus r_T$ for any T then instead of picking random s and then random r_T's the game could equivalently just pick random z_T's. Thus the authentication game (without either server or device leakage query) can be rendered as follows: $\mathsf{Auth_D}$ at time T sends out z_T, and $\mathsf{Auth_S}$ accepts at time T if it receives (p, z_T) and rejects otherwise. Therefore Adv gains nothing from learning the server acceptance bit on honest client's sessions even if Adv hijacks the C-D channel, and if Adv cannot do with-device server sessions (and cannot violate the timing synchrony assumption), then seeing $z_{T'}$ values for any $T' < T$ carries no information about z_T. Therefore Adv has to guess p together with z_T, and hence the probability of $\mathsf{Succ_A}$ after this chain of modifications is at most $q_\mathsf{S}/2^{d+t}$, and therefore in the actual authentication game it is negligibly close to the same amount. If Adv makes the client leakage query then it can use the real password in its server sessions, but it succeeds again only if it predicts z_T for some unseen index T, hence $\mathsf{Succ_A}$ is at most $q_\mathsf{S}/2^t$. Finally if Adv makes the device leakage query then the authentication game reduces to just on-line password-guessing, where $\mathsf{Succ_A}$ is at most $q_\mathsf{S}/2^d$.

For password-recovery resistance (without adversarial device access) note that Adv is so heavily restricted in this authentication game that it reveals no additional information on top of $\mathsf{st_S} = (h, k)$ learned from the server leakage query. If Adv is prevented from device leakage, the PRF key k bears no relation to $h = H(p, s)$, and so the only way to find p is for Adv to query the random oracle H on (p, s) pairs in the $\mathsf{D} \times \{0,1\}^t$ search space, so $\Pr[\mathsf{Succ_P}] \leq \gamma$ after $\gamma \cdot 2^{d+t} \cdot \bar{T}_H$ search time. If Adv also learns s from a device leakage query, the search goes over just p's, so $\Pr[\mathsf{Succ_P}] \leq \gamma$ after $\gamma \cdot 2^d \cdot \bar{T}_H$ search time. ■

Note that the assumption that the adversary has no access to the device is essential for the $n_\mathsf{N} = 2^{d+t}$ time bound for password-recovery given $\mathsf{st_S} = (h, k)$. Indeed, from eavesdropping on a single C-D session where D runs $\mathsf{Auth_D(s, k)}$, Adv learns $z = s \oplus f_k(T)$ for a known T, and since Adv knows k from $\mathsf{st_S}$, it gets s, which reduces the search space to 2^d.

Symmetric-Key TFA Protocol. The time-based protocol has very low C-D communication requirements, but it requires clock synchronization between D and S, and it is vulnerable to attacks if an adversary can modify D's clock, e.g. by shifting it forward after having gained temporary access to D. TFA-SC eliminates this vulnerability, but just like TFA-T, it is password-recovery resistant only without adversarial device access. This last weakness is eliminated only by the public-key based protocols TFA-PC and TFA-PC-Alt.

The TFA-SC protocol is very similar to the TFA-T protocol, and it differs from it only in how the random value r is derived by D and S. As in TFA-T, it is computed via a PRF F whose key k is shared by D and S, but instead of applying F_k to the current time, here F_k is applied to a nonce x chosen by S. To enable adjusting the protocol to different constraints on the C-to-D bandwidth, we use an additional security parameter τ'. In the asymptotic security analysis we assume that τ' is a positive linear function of τ, but in practice τ' could be a little smaller, e.g. one could use an AES block cipher for F with $\tau = 128$, while setting τ' at 80. We discuss these exact security parameters below.

<ins>Symmetric-Key TFA Scheme TFA-SC</ins>

<ins>Init</ins>$(1^\tau, t, p)$: Pick $s \leftarrow \{0,1\}^t$ and $k \leftarrow \{0,1\}^\tau$, compute $h = H(p, s)$, and set $\mathsf{st_D} = (s, k)$ and $\mathsf{st_S} = (h, k)$.

Protocol <ins>Auth</ins>:
(assuming secure C-S channel s.t. S is authenticated to C)

1) Server S picks $x \leftarrow \{0,1\}^{\tau'}$ and sends x on the secure channel to C.
2) Client C passes x as its message ch to D.
3) Device D on input $\mathsf{st_D} = (s, k)$ and message $\mathsf{ch} = x$, computes $r = F_k(x)$ and sends $z = s \oplus r$ as message resp to C.
4) Client C on input p and D's message $\mathsf{resp} = z$, sends (p, z) on the secure channel to S.
5) Server S on input $\mathsf{st_S} = (h, k)$ and C's message (p, z), computes $r = F_k(x)$, and accepts if and only if $h = H(p, z \oplus r)$.

The asymptotic authentication-attack resistance and password-recovery resistance properties of TFA-SC are as for TFA-T, the only difference being that TFA-SC does not depend on D's internal clock. (On the other hand TFA-SC requires a device D of type II or higher because D receives message from C.)

Theorem 2: If F is a secure PRF and H is a Random Oracle then TFA-SC is $(1/2^{t+d}, 1/2^d, 1/2^t)$-authentication-attack resistant and $(\bar{T}_H, 2^{t+d}, 2^d)$-password-recovery resistant without adversarial device access for parameters (t, d), where $\bar{T}_H$ is the time required to compute H on any input.

Proof: The argument for authentication-attack resistance is similar as for TFA-T: Without server or device leakage, by security of PRF we can replace values r_x defined as $F_k(x)$ by random t-bit strings, and by CRH security of H we can have S accept (p', z') if it is equal to $(p, s \oplus r_x)$. Also, if $z_x = s \oplus r_x$ then instead of picking random s and random r_x's we can just pick random z_x's. Thus the authentication game (without either server or device leakage query) can be modified as follows: On device sessions, for any x chosen by Adv, D outputs z_x, and on either server or client sessions, $\mathsf{Auth_S}$ sends out a random τ'-bit x. On client sessions Adv learns z_x, on hijacked client sessions Adv can reply with any z'_x and learn whether $z'_x = z_x$, and on server sessions Adv succeeds if it sends $(p', z'_x) = (p, z_x)$. The only way Adv can win with probability larger than $q_\mathsf{S}/2^{d+t}$ is if there is a collision in x values, namely if on any server sessions S picks x on which D has been queries before, either on device or on client session. However, the probability of such collision is upper-bounded by $q_\mathsf{S}(q_\mathsf{D} + q_\mathsf{C}) \cdot 2^{-\tau'}$, which is negligible. An additional device leakage query reduces the game to on-line password-guessing, where $\mathsf{Succ_A}$ is at most $q_\mathsf{S}/2^d$, exactly as in the case of the TFA-T protocol.

As for password-recovery resistance (without adversarial device access), the situation is as in the case of the TFA-T protocol, i.e. Adv is so heavily restricted in this game that it reveals no additional information on top of $\mathsf{st_S} = (h, k)$ learned from the server leakage query. As in the case of TFA-T, without device leakage the PRF key k bears no relation to $h = H(p, s)$, and so Adv has to query the random oracle H on (p, s) pairs in the $\mathsf{D} \times \{0,1\}^t$ search space, hence $\Pr[\mathsf{Succ_P}] \leq \gamma$ after time $\gamma \cdot 2^{d+t} \cdot \bar{T}_H$. If Adv in addition learns s via the device leakage query, the search goes over just p's, so $\Pr[\mathsf{Succ_P}] \leq \gamma$ after time $\gamma \cdot 2^d \cdot \bar{T}_H$. ■

As in the case of TFA-T, the assumption that the adversary has no access to the device is essential for the $n_\mathsf{N} = 2^{d+t}$ time bound for password-recovery given $\mathsf{st_S} = (h, k)$, because given k one compute r from x and recover s.

Public-Key TFA Protocols. The TFA-PC and TFA-PC-Alt protocols modify protocol TFA-SC (and TFA-T) in only one respect, namely that the random challenge r is agreed-upon using a public-key encryption for which S holds the encryption key pk and D a decryption key sk. In this way the adversary who corrupts the server cannot derive past or future challenge values r from eavesdropping on the C-D channel, via an eavesdropped client session query. Note that an adversary who corrupts S and performs an active attack on D via a device session query can always encrypt some known value r and then derive s as $z \oplus r$ from D's response z. However, a service whose secret storage gets compromised can often detect that the compromise took place, and can respond by asking users to re-run the setup of an TFA protocol. Thus an adversary's ability to perform additional attacks against users, e.g. by active attacks on their devices, might be limited after the server's compromise. On the other hand, we do want to maintain security against an adversary who corrupts S and tries to recover users' passwords even if personal devices of some of these users were *previously* subjected to active attacks.

Using just public key encryption for transferring the challenge r from S to D does not suffice, because even if the public key pk is stored only at server S, an adversary might still be able to create a valid encryption of message r together with some information about r. In the first public-key based protocol, TFA-PC, we address this by simply adding a Message Authentication Code (MAC) on the ciphertext encrypting r, computed under a symmetric key shared by S and D. The advantage of this protocol is that it can use any public key encryption, but the challenge message ch which needs to be sent on the C-to-D channel becomes at least 344 bits long, if it is instantiated with ElGamal encryption on a 196-bit elliptic curve, a 128-bit MAC, and 20-bit challenge r. (The challenge value r can be shorter than in TFA-SC because of the presence of a MAC, as we explain below.) Further down, in protocol TFA-PC-Alt, we will show that under more specific assumptions we can significantly reduce this C-to-D bandwidth, e.g. from 344 to 196 bits.

The TFA-PC protocol assumes a CRH H as in TFA-SC above, a semantically-secure public key encryption $(\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$, and an unforgeable MAC $(\mathsf{Mkg}, \mathsf{Mac}, \mathsf{Ver})$.

Public-Key TFA Scheme TFA-PC

Init$(1^\tau, t, p)$: Pick $s \leftarrow \{0,1\}^t$, $(sk, pk) \leftarrow \mathsf{Kg}(1^\tau)$, $k \leftarrow \mathsf{Mkg}(1^\tau)$, compute $h = H(p, s)$, and set $\mathsf{st_D} = (s, sk, k)$ and $\mathsf{st_S} = (h, pk, k)$.

Protocol Auth:
(assuming secure C-S channel s.t. S is authenticated to C)

1) Server S on input $\mathsf{st_S} = (h, pk, k)$, picks $r \leftarrow \{0,1\}^t$, encrypts $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{r})$, computes a MAC $\sigma \leftarrow \mathsf{Mac}(\mathsf{k}, \mathsf{c})$, and sends (c, σ) on the secure channel to C.
2) Client C passes (c, σ) as its message ch to D.
3) Device D on input $\mathsf{st_D} = (s, sk, k)$ and message $\mathsf{ch} = (c, \sigma)$, stops if $\mathsf{Ver}(\mathsf{k}, \mathsf{c}, \sigma) \neq 1$, otherwise decrypts $r \leftarrow \mathsf{Dec}(\mathsf{k}, \mathsf{c})$, and sends $z = s \oplus r$ as message resp to C.
4) Client C on input p and D's message $\mathsf{resp} = z$, sends (p, z) on the secure channel to S.
5) Server S on C's message (p, z), accepts if and only if $h = H(p, z \oplus r)$.

The authentication-attack resistance property of TFA-PC is as in TFA-T and TFA-SC, but the password-recovery resistance is stronger since it holds whether or not the adversary eavesdrops on client-device sessions or has active access to the user's device prior to server corruption.

Theorem 3: If $(\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure PKE, $(\mathsf{Mkg}, \mathsf{Mac}, \mathsf{Ver})$ is an unforgeable MAC (in the sense of universal unforgeability under the chosen message attack), and if H is a Random Oracle, then TFA-PC is $(1/2^{t+d}, 1/2^d, 1/2^t)$-authentication-attack resistant and $(\bar{T}_H, 2^{t+d}, 2^d)$-password-recovery resistant for parameters (t, d), where $\bar{T}_H$ is the time required to compute H on any input.

Proof: We first argue authentication-attack resistance. To see the $\delta_{\mathsf{N}} = 1/2^{d+t}$ bound, first observe that by MAC unforgeability we can discount as occurring with negligible probability the event that Adv manages to get D to respond on any device session otherwise but by forwarding some (c, σ) pair received on a server session or eavesdropped on a client session. In either case Adv sees (c, σ, z) where $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{r})$, $\sigma = \mathsf{Mac}(\mathsf{k}, \mathsf{c})$, and $z = s \oplus r$. Whenever Adv sends some z' back to S on a hijacked client session, Adv learns if $z' = s \oplus r$ for some (c, σ) as above, but this gives no additional information because it holds iff z' is equal to z which Adv can see by merely eavesdropping on this client session. Note that σ's add no additional information on secrets (p, s) so we can ignore them. Event $\mathsf{Succ_A}$ holds if any (p', z') on a server session satisfies $p' = p$ and $z' = s \oplus r$. By semantic security of PKE, event $\mathsf{Succ_A}$ holds with at most negligibly different probability in a game where all c's are replaced by encryptions of independent random values, and therefore we can ignore the ciphertexts c, in which case Adv's view is reduced to values $z = s \oplus r$ for random strings r, which leaks no information about s. (Even repeats in values r are not a problem.) Therefore after this series of modifications $\Pr[\mathsf{Succ_A}]$ is upper-bounded by $q_{\mathsf{S}} \cdot 1/2^{d+t}$. Client leakage query changes the game only by giving p to Adv, in which case $\Pr[\mathsf{Succ_A}]$ is upper-bounded by $q_{\mathsf{S}} \cdot 1/2^t$ after the same modifications. Device leakage query reveals s but gives no information about p, hence in this case $\Pr[\mathsf{Succ_A}] \leq \mathsf{q_S} \cdot 1/2^{\mathsf{d}}$.

For password-recovery resistance, note that even $\mathsf{st_S} = (h, pk, k)$ does not significantly change the above arguments, as long as Adv is prevented from making device session queries after receiving $\mathsf{st_S}$ containing the MAC key k. The same series of modifications shows that Adv's view in device and client sessions before learning $\mathsf{st_S}$ and of eavesdropped client sessions afterwards, is indistinguishable when z's are replaced by random values, in which case Adv's search for (p, s) is reduced to querying the random oracle H on any γ fraction of the search space $D \times \{0,1\}^t$, hence lower-bounding Adv's time by $\gamma \cdot 2^{d+t} \cdot \bar{T}_H$. A device leakage query reveals (s, sk, k), making all network interactions predictable to Adv, but Adv's view still leaks nothing about p, thus Adv's time is still lower-bounded by $\gamma \cdot 2^d \cdot \bar{T}_H$. ■

Our final protocol, TFA-PC-Alt, is a variant of protocol TFA-PC. First, we observe that in protocol TFA-PC-Alt public key encryption (PKE) is used to encrypt a random value r, and therefore we can save bandwidth by replacing PKE with a key encapsulation mechanism (KEM). In the case of hashed ElGamal this saves $|r| \geq 20$ bits from the ciphertext. Moreover, we will assume a special property of KEM, satisfied by hashed ElGamal encryption in the random oracle model, that without knowledge of the public key an adversary has a negligible probability of creating a ciphertext with any sideline information on the encapsulated plaintext, thus shaving off also the 128 bits needed by a MAC.

A KEM is a triple $(\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ s.t. (1) a key generation algorithm $\mathsf{Kg}(1^\tau, \mathsf{t})$ outputs a public key pair (sk, pk), (2) an encapsulation algorithm $\mathsf{Enc}(\mathsf{pk})$ outputs a pair (c, r) where r is random t-bit string, and (3) a decapsulation algorithm $\mathsf{Dec}(\mathsf{sk}, \mathsf{c})$ outputs the same r value chosen by the encapsulation algorithm. A key encapsulation mechanism is semantically secure if for every polynomial time Adv, we have that the probability that $b' = b$ is a negligible function of ϵ where b', b are defined by the following game: The challenger generates $(pk, sk) \leftarrow \mathsf{Kg}(1^\tau, \mathsf{t})$, then it generates $(c, r) \leftarrow \mathsf{Enc}(\mathsf{pk})$, then it picks bit $b \leftarrow \{0,1\}$, defines r_b as r, picks $r_{1-b}\{0,1\}^t$, finally b' is computed by $\mathsf{Adv}(pk, r_0, r_1, c)$.

We call KEM $\Sigma = (\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ *outsider oblivious* if the key generation algorithm $\mathsf{Kg}(1^\tau, \mathsf{t})$ outputs public parameter π which defines a ciphertext space C_π together with the public key pair (pk, sk), and for every polynomial-time adversary Adv and every t polynomial in τ, function ϵ defined as $\epsilon(\tau) = |p^0_{\mathsf{Adv},\Sigma}(\tau) - p^1_{\mathsf{Adv},\Sigma}(\tau)|$ is negligible where $p^b_{\mathsf{Adv},\Sigma}$ for $b = 0, 1$ is defined as the probability that $\mathsf{Adv}(1^\tau)$

outputs 1 in the following game: The challenger generates $(sk, pk, \pi) \leftarrow \mathsf{Kg}(1^\tau, \mathsf{t})$, and whenever $\mathsf{Adv}(1^\tau, t, \pi)$ sends a decryption query $c \in C_\pi$, it receives $m = \mathsf{Dec}(\mathsf{sk}, \mathsf{c})$ if $b = 1$ and a random t-bit value r_c if $b = 0$ (for every c the corresponding r_c is chosen only once, so if $c' = c$ then $r_c = r_{c'}$). The game ends when Adv outputs a bit b' designating its judgment whether b is 0 or 1. Note that hashed Diffie-Hellman over a cyclic group is outsider oblivious in the random oracle model as long the group has super-polynomial number of elements: Let R be a hash function onto $\{0,1\}^t$, modeled as a random oracle. Let $\mathsf{Kg}(1^\tau, \mathsf{t})$ choose a cyclic group G of prime order q and generator g where the DH assumption holds with sec.par. τ, let x be a random element in Z_q, let $y = g^x$, and let $sk = (x, \pi)$, $pk = (y, \pi)$, and $\pi = (g, q)$. Let $\mathsf{Enc}(\mathsf{y}, \pi)$ pick $a \leftarrow \mathsf{Z}_q$ and output $c = g^a$ and $r = R(y^a)$, and let $\mathsf{Dec}((\mathsf{x}, \pi), \mathsf{c})$ check if c is an element of G (e.g. by checking if $c^q = 1$), and if so output $r = R(c^x)$. Note that Adv can distinguish $R(c^x)$'s from random values only by querying R on c^x for some decryption query c. Since x is random in Z_q and q is exponential in τ, this happens with negligible probability. It is also easy to see that this hashed Diffie-Hellman KEM is semantically secure in the random oracle model if the (computational) Diffie-Hellman assumption holds in group G. Apart of the outsider oblivious and semantically secure KEM, the TFA-PC-Alt protocol also assumes a CRH H as all the other TFA protocols shown above.

Bandwidth-Improved Public-Key TFA Scheme TFA-PC-Alt

Init$(1^\tau, t, p)$: Pick $s \leftarrow \{0,1\}^t$ and $(sk, pk, \pi) \leftarrow \mathsf{Kg}(1^\tau, \mathsf{t})$, compute $h = H(p, s)$, and set $\mathsf{st_D} = (s, sk)$ and $\mathsf{st_S} = (h, pk)$.

Protocol Auth:
(assuming secure C-S channel s.t. S is authenticated to C)

1) Server S on input $\mathsf{st_S} = (h, pk)$, generates $(c, r) \leftarrow \mathsf{Enc}(\mathsf{pk})$ and sends c on the secure channel to C.
2) Client C passes c as its message ch to D.
3) Device D on input $\mathsf{st_D} = (s, sk)$ and message $\mathsf{ch} = c$, checks if $c \in C$ and if so computes $r = \mathsf{Dec}(\mathsf{sk}, \mathsf{c})$ and sends $z = s \oplus r$ as message resp to C.
4) Client C on input p and D's message $\mathsf{resp} = z$, sends (p, z) on the secure channel to S.
5) Server S on C's message (p, z), accepts if and only if $h = H(p, z \oplus r)$.

Theorem 4: If $(\mathsf{Kg}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure and outsider oblivious KEM, and if H is a Random Oracle, then TFA-PC-Alt is $(1/2^{t+d}, 1/2^d, 1/2^t)$-authentication-attack resistant and $(\bar{T}_H, 2^{t+d}, 2^d)$-password-recovery resistant for parameters (t, d), where $\bar{T}_H$ is the time required to compute H on any input.

For lack of space, we defer this proof to the extended version of this paper, but the argument is very similar as in the case of the TFA-PC protocol. The key difference is that instead of using the MAC to disable Adv's active attacks on the device sessions, we rely on outsider obliviousness of KEM to argue that D's responses $z = s \oplus r$ for $r = \mathsf{Dec}(\mathsf{sk}, \mathsf{c})$ on ciphertexts c chosen by Adv are indistinguishable from random strings no matter how Adv chooses queries c to D.

Implementation/Extension Notes. (1) The only protocol where S needs UN to retrieve $\mathsf{st_S}$ at the beginning of the Auth protocol is TFA-PC. However, in TFA-T, TFA-SC, and TFA-PC-Alt instantiated with hashed Diffie Hellman KEM, if all users are initialized using the same security parameter τ (and τ' in the case of TFA-SC) then S needs $\mathsf{st_S}$ only in the last step, hence C can send UN together with (p, z). This is reflected in Section V, e.g. compare when UN is sent in Figures 2 and 3. **(2)** If a TFA protocol is implemented on device of type III or IV, where D's response resp to C can contain more information, and if the D-to-C channel is authenticated (e.g., as in our implementation involving bidirectional QR codes), or because D and C can establish authentication keys in the initialization process (as in our Bluetooth pairing based implementation), then D could include a hash of server's S public key in its message resp, and C could check, before sending its (p, z) (and UN) message to S, that the SSL session with S is established under the correct S's public key. This way we can extend user's security to the case when the adversary does not simultaneously corrupt the PKI *and* the device D and/or the authentication on the D-to-C channel.

V. System Design and Implementation

In this section, we present the design, implementation and performance evaluation of the different TFA mechanisms built on top of the TFA protocols presented in Section IV. As introduced in Section III, these mechanisms are categorized based on: (1) the underlying device type, namely, low-bandwidth device (LBD; type I and II), mid-bandwidth device (MBD; type III) and full-bandwidth device (FBD; type IV); (2) the underlying C-to-D and/or D-to-C channel, namely, based on PIN entry, QR codes, Bluetooth or WiFi; and (3) the underlying protocol, namely, TFA-T, TFA-SC and TFA-PC.[2] The mechanisms are: LBD-PIN which is based on the protocol TFA-T, and LBD-QR-PIN, MBD-QR-QR, FBD-BT-BT, FBD-WF-WF, FBD-QR-BT and FBD-QR-WF which can be based on either TFA-SC or TFA-PC. This results in a total of 13 TFA mechanisms that we have developed. Table II highlights the different features of these mechanisms.

In our implementation, S is a web server maintaining and accessing user accounts database to authenticate each user U labeled with a unique user name UN. In all our mechanisms, S communicates to C over a secure SSL channel. We used an Apache web server with MySQL database running our server-side PHP scripts. C is a terminal having a plain HTML browser in our LBD mechanisms, and having a browser extension (written in JavaScript) in all our FBD mechanisms, except of MBD-QR-QR which requires an HTML5 browser. C communicates to D over mix-bandwidth channels, formed using manual PIN entry, QR codes, Bluetooth or WiFi, and combinations thereof. Finally, D is an Android smartphone running the TFA authentication application (TFA-App) written in Java (SDK 10 or up) that supports all of our proposed mechanisms. The crypto operations in our implementation utilize the OpenSSL and PHP mcrypt libraries on S, and java.security class on D. The QR code encoding and decoding, when needed, uses

[2]Our current implementation excludes the TFA-PC-Alt protocol as it requires certain crypto primitives not built into off-the-shelf crypto libraries. However, since it requires shorter bandwidth over D-C channel than the TFA-T, TFA-SC and TFA-PC protocols as well as crypto operations of similar complexities, by implementing and testing these protocols and underlying channels, we are implicitly demonstrating the feasibility of TFA-PC-Alt protocol also.

the ZXing library [15]. We next elaborate on the design and implementation of our LBD TFA mechanisms, our MBD TFA mechanism and our FBD TFA mechanisms We describe their initialization phase followed by their authentication phases.

A. Initialization Phase

Before using the TFA mechanism, U needs to register with the service deploying that TFA mechanism. This is done using the Init procedure of our protocols, during which protocol parameters and keys are agreed upon between S to D. This information varies across our protocols but it follows a generic URI syntax (as per RFC3986[3]). Regardless of the type of underlying protocol (TFA-T, TFA-SC or TFA-PC), S transfers set-up information to D via C using QR codes, following the approach adopted by Google Authenticator [6]. Generally the URI includes protocol type, service domain name, encryption keys, and secret value, and PRF key. S embeds this information into a QR Code and delivers to C, which is captured by D and interpreted by TFA-App on D. Subsequently, one TFA authentication phase (see next subsection) round is completed to accomplish enrollment.

Only a one-time initialization phase is required for each user account after which user account information including protocol type, username UN, random salt value $salt$ (128 bits), salted hash of password h $(= H(p, s, salt))$, and key k (128 bits), and additionally D's public key pk (for TFA-PC protocol), is stored in server database, and domain name DN, key k and secret s (19 bits in LBD mechanisms and 128 bits in FBD mechanisms), and additionally D's private key sk (for TFA-PC protocol), is stored on device database. To provide better security, unlike Google Authenticator, we do not store UN on the device unless a user has more than one account with a service. This prevents an attacker, who compromises the user's device, from determining which user account corresponds to the key k stored on the device. The service's DN is stored on the device to identify different TFA services user has registered with. Similar to known TFA mechanisms (such as Google Authenticator), we assume that the initialization phase is not compromised by an attacker.

B. LBD Authentication Phase

1) LBD-PIN: LBD-PIN is essentially an improvement to Google's TFA system. Here, U first launches the app on D and manually identifies the service she wants to authenticate to. D then creates the PIN z (19 bits encoded into 6 digits) derived from s and a PRF computation of current timestamp T_d (Unix time in compliance with RFC 6238) using k. U then copies z onto C and inputs the username UN and password p, and submits (UN, p, z) to S. Finally, S evaluates the response by computing the PRF using k on its own current timestamp, and authenticates the user based on the received information. In our implementation, we instantiated the PRF using HMAC-SHA256.[4] Figure 1 depicts the mechanism as implemented.

We re-purposed Google Authenticator's open source code [6] to fit our LBD-PIN mechanism. Fresh PIN is generated every 30s and U is given up to 1min to copy the PIN to C.

[3]https://tools.ietf.org/html/rfc3986

[4]HMAC is proven to be a PRF under the assumption that the underlying hash compression function is a PRF [17].

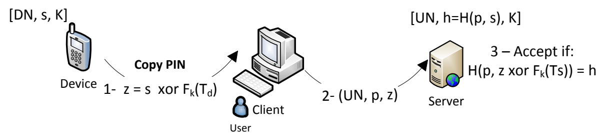


Fig. 1. LBD-PIN

2) LBD-QR-PIN: In the LBD-QR-PIN mechanism, S generates a random (128-bit) challenge. When using TFA-SC, no computation is further performed on the challenge. When using TFA-PC, S encrypts the challenge with D's public key pk using RSA-OAEP-3072[5] and authenticates the resulting ciphertext using k with HMAC-SHA256. S encodes the response and DN into a QR code and sends the QR code image to C. U takes the snapshot of the QR code displayed on C's screen using camera on D, which then reads the contents of the code, processes/verifies it and produces a response PIN z (19 bits encoded into 6 digits) as per the protocol used (TFA-SC or TFA-PC). U then copies z onto C and inputs p, and submits (p, z) to S. Finally, S evaluates the response and authenticates the user based on the received information. Note that the LBD-QR-PIN version based on TFA-SC allows the user to input (UN, p, z) in one single page, whereas the version based on TFA-PC requires sending UN before providing (p, z). The TFA-SC and TFA-PC versions of the mechanisms are depicted in Figures 2 and 3. Figures 4 depicts the implementation snapshots of the client terminal and device during an authentication session.

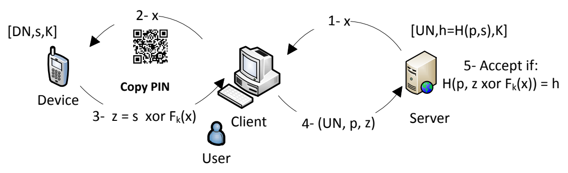


Fig. 2. LBD-QR-PIN (SC)

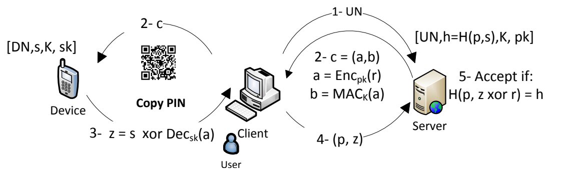
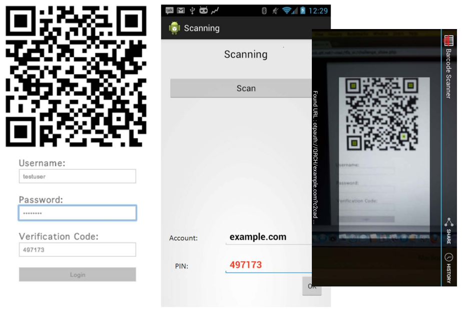


Fig. 3. LBD-QR-PIN (PC)

C. MBD Authentication Phase (MBD-QR-QR)

HTML5 introduced video, audio and canvas, which make handling multimedia and graphical contents easy in a plain browser without extensions and plugins. Some JavaScript APIs use this functionality to access webcam. getUserMedia is an example of such an API. It is supported on most of the browsers, some of which request user consent before opening webcam. An application of such APIs is a webcam QR Code

[5]3072-bit RSA provides security equivalent to 128 bits in a symmetric key system [12].


Username: testuser

Password: ••••••••

Verification Code: 497173

Login

Scanning

Scan

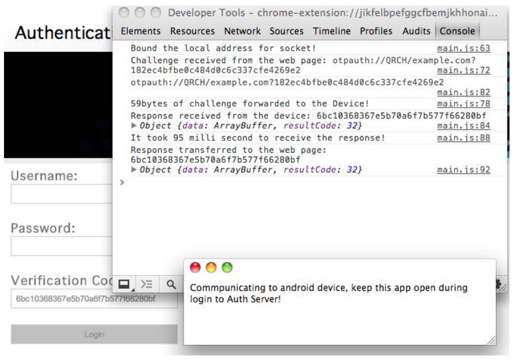
Account: example.com

PIN: 497173

OK

Found URL : otpauth://QRCH/example.com?c2cad ...

Fig. 4. LBD-QR-PIN Server Challenge; Device Scanning and Response


Developer Tools – chrome-extension://jikfelbpefggcfbemjkhhonai...

```
Bound the local address for socket!                                      main.js:63
Challenge received from the web page: otpauth://QRCH/example.com?
182ec4bfbe0c484d0c6c337cfe4269e2                                         main.js:72
otpauth://QRCH/example.com?182ec4bfbe0c484d0c6c337cfe4269e2
                                                                         main.js:82
59bytes of challenge forwarded to the Device!                            main.js:78
Response received from the device: 6bc10368367e5b70a6f7b577f66280bf
▶ Object {data: ArrayBuffer, resultCode: 32}                             main.js:84
It took 95 milli second to receive the response!                         main.js:88
Response transferred to the web page:
6bc10368367e5b70a6f7b577f66280bf
▶ Object {data: ArrayBuffer, resultCode: 32}                             main.js:92
```

Commpunicating to android device, keep this app open during login to Auth Server!

Username:

Password:

Verification Coc 6bc10368367e5b70a6f7b577f66280bf

Login

Authenticat
Elements Resources Network Sources Timeline Profiles Audits Console

Fig. 5. FBD-WF-WF in action

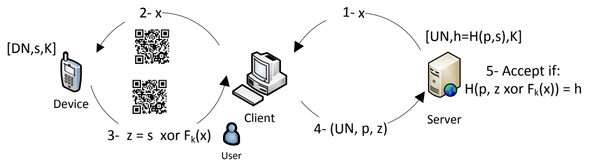


Fig. 6. MBD-QR-QR (SC)

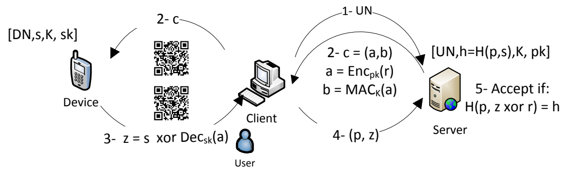


Fig. 7. MBD-QR-QR (PC)

reader operating in the browser, which we used to form a D to C QR visual channel central to our MBD-QR-QR mechanism.

In this mechanism, same as LBD-QR-PIN, the challenge is encoded in a QR Code and is shown on the web page, which D captures and interprets. In contrast to LBD-QR-PIN, the device-generated response is also encoded in a QR code by the TFA-App using ZXing encode class. To receive the response, LazarSoft JavaScript QR Code reader [7] is integrated with our server-side PHP scripts. LazarSoft uses getUserMedia to capture the QR Code and ZXing to interpret the QR code. Once decoded, response is transferred automatically to the webpage to be submitted to S. U should assist by showing the generated response to the terminal webcam and submitting UN and/or (p, z). Figures 6 and 7 depict the MBD-QR-QR mechanisms based on TFA-SC and TFA-PC, respectively. Our tests show that this channel is robust for sending between 20-128 bits from D to C (the details are provided in Section V-E).

D. FBD Authentication Phase

The authentication phase of each FBD mechanism is described next. These mechanisms follow the TFA-SC and TFA-PC protocols in the same way as the LBD-QR-PIN or MBD mechanisms, with an exception that the length of the response PIN can be long (e.g., 128 bits).

1) FBD-BT-BT: In the FBD-BT-BT authentication scenarios, D and C establish a bidirectional Bluetooth channel. On D, this channel is implemented as an Android application operating in server mode by listening on a RFCOMM socket, which is addressed using a UUID (universally unique identifier) in accordance with the Serial Port Profile (SPP).[6] C runs a Google Chrome browser extension which employs the use of the Bluetooth API. D must run the application in the background to establish connectivity.

The Bluetooth API enables Google Chrome to operate in either client or server mode. In our case, we have developed a packaged extension, called "TFA-BT-App". When TFA-BT-App is run, the terminal browser loads the authentication page on S. When U initiates the log in process, the web site JavaScript hosted on S calls the browser extension using a well-known ID[7] with the challenge sent by S as input. The extension (1) establishes Bluetooth connectivity using the BT adapter address of D (which is provided to C during the initialization phase) rather than going through the slow process of BT device discovery during the authentication session, (2) sends the challenge to D, (3) receives the response PIN (128 bits or more), and finally (4) returns the response to the terminal browser. The browser then sends the response including the password to S which then authenticates U. Figures 8 and 9 illustrate the FBD-BT-BT mechanisms.

One limitation of our implementation is that it requires a paired Bluetooth connection (although our protocols do not require the C-D channel to be secure). In our implementation, this one-time pairing is established during the initialization phase after which C and D can take part in the authentication process without involvement from U (except of launching the app). The pairing process will need to be repeated whenever U roams over to another client terminal. It is also possible to establish unpaired (insecure) BT connection as shown in PhoneAuth [19], but it requires developing a new NPAPI plugin embedded with the browser extension. While NPAPI provides more flexibility in the design space for browser plugins, it allows access to the host system libraries subverting the sandbox security of Google Chrome. Due to the security consequences of NPAPI this traditionally makes wide-scale deployment more difficult to achieve. We aim to comply with browser security features and to remain forward-compatible,

[6]By using an insecure RFCOMM socket, C can establish a connection to D without pairing, but this requires support from the host platform Bluetooth stack.

[7]The ID is essentially a signature to uniquely identify the extension.

Mechanism	**LBD-PIN**	**LBD-QR-PIN**		**MBD and FBD mechanisms**	
Protocol	*TFA-T*	*TFA-SC* [1]	*TFA-PC* [2]	*TFA-SC*	*TFA-PC*
|z| = |s|	19 bits	19 bits	19 bits	128 bits	128 bits
Creating Challenge at "S"	N/A	1.3×10^{-5} µs	8.7×10^{-4} µs	1.3×10^{-5} µs	8.7×10^{-4} µs
Challenge "c" is:		c = x; 128-bit	c = (a = Enc(r), b = MAC(a)); 3328-bit	Same as LBD-QR-PIN	Same as LBD-QR-PIN
Verification at "S"	1.3×10^{-4} µs	3.7×10^{-5} µs	1.2×10^{-5} µs	4.9×10^{-5} µs	1.3×10^{-5} µs
Accept if: H(p, z xor r) = h r is:	PRF(T_s)	PRF(x)		PRF(x)	
Verify message integrity at "D"	N/A	N/A	0.48 ms	N/A	0.48 ms
Accept if MAC(a) = b			a = Enc(r), b = MAC(a)		a = Enc(r), b = MAC(a)
Creating response at "D"	3.2 ms	0.54 ms	114 ms	0.59 ms	114 ms
Response "z" is:	s xor PRF(T_d)	s xor PRF(x)	s xor Dec(a)	s xor PRF(x)	s xor Dec(a)
Reading of QR Code by "D"[3]	N/A	≅ 5 s	≅ 8 s	N/A	N/A
Reading of QR Code by "C"[4]	N/A	N/A	N/A	≅ 870ms	≅ 870ms
WF Auto response[5]	N/A	N/A	N/A	≅ 239 ms	≅ 553 ms
BT Auto response	N/A	N/A	N/A	≅ 2s	≅ 2s

Execution times were evaluated on a *dual-core 1GHz processor smart phone* with *8.0 MP Camera* and an *Intel Core Due 2.26 server and client*. The values presented represent the average across 10,000 trials for each computation. For QR, 20 manual scans were repeated. To time BT/WF auto-response, 100 trials were performed.

[1] PRF is instantiated using an HMAC function. It starts by inputting a random generated 128-bit session "r" and a 128-bit "key" to an *HMAC-SHA256* function. Output is a truncation of the HMAC starting at an offset defined by the last nibble of the HMAC. HMAC output is presented in Hex-String.

[2] In this model, a *3072-bit RSA with OAEP* padding is implemented, message is authenticated using HMAC-SHA256. The output is presented in Base64 encoding to maximize efficiency when embedding the challenge in a QR Code.

[3] QR size: 350x350, Error correction: M (15% of code words get restored)

[4] The time shows an average cost of scanning and decoding a well posed QR code, user time to align the device is ignored but it is estimated to be up to 5 seconds.

[5] Auto-Response denotes the delay between the time browser extension receives a challenge from S and the time it responds back to "S". It includes the entire computation performed on D and the 2-way C-D communication (for both WF and BT).

TABLE I. EXECUTION TIME FOR THE DIFFERENT TFA MECHANISMS

trading off the usability aspect of allowing unpaired Bluetooth communication.

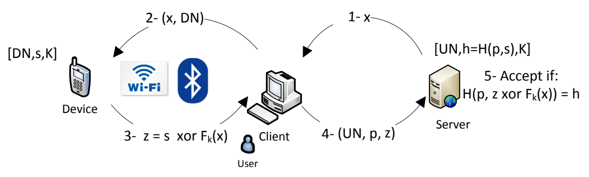


Fig. 8. FBD-BT-BT/FBD-WF-WF (SC)

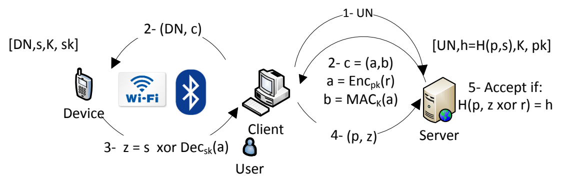


Fig. 9. FBD-BT-BT/FBD-WF-WF (PC)

2) FBD-WF-WF: The FBD-WF-WF mechanism utilize WiFi as a bidirectional D-C communication medium. While Bluetooth has been explored in user authentication domain in prior work (e.g., [19], [24], [20]), WiFi has received little attention so far. Ad hoc mode WiFi is a common point-to-point infrastructureless wireless channel that can be formed between a device and client terminal equipped with WiFi adaptors. However, laptops terminals may dedicate wireless adaptor to the Internet connection, which may render ad hoc WiFi unavailable for D-to-C communication in our TFA application. We show that this constraint can be addressed by utilizing a tool called Virtual WiFi (offered by Microsoft Research), also known as Wireless Hosted Network [11].

Virtual WiFi features two coexisting functionalities: "virtualizing a physical wireless adapter", and "running a software-based wireless access point (SoftAP)," which is appropriate for use in our TFA application. We used Virtual WiFi to establish a direct connection between C and D upon a request from S in order to transfer a challenge to D through C and receive a response back whenever wireless adaptor is serving other networks. Therefore, by virtualizing one physical wireless adapter, we can connect D to C whilst user is surfing web on C. Virtual WiFi is built-into some Microsoft platforms (e.g., Windows 7 and 8) and installable on some others, and is easy to configure and fairly stable. Still in our implementation, we had to adapt it to address the constraint that all devices connected to SoftAP should use the WPA2-PSK/AES cipher suite. In other words, an insecure connection is not an option with Virtual WiFi, and therefore we specified a static key and hard-coded it into the application[8]. After the wireless connection is established between C and D, applications on the two sides can communicate back and forth.

We have developed a chrome packaged app, titled "TFA-WiFi-App", that is launched on the client browser, and extended our TFA-App on the device to interact with the client. TFA-WiFi-App stands between the server and the client to relay challenges received from the web page to the device, receive device's responses and forward it to the web-page. Chrome provides the chrome.socket API to the packaged apps to send and receive data over the network using TCP and UDP connections; we used this API to create a UDP channel between the client and the device. Furthermore, chrome.runtime API allows the chrome extensions and apps to listen for and respond to events; we used this API for message passing between the web-page and WiFi-App. Every time the user opens a login web-page the server sends a challenge, which fires a function in our TFA-WiFi-App to "multicast" the challenge on the created UDP socket to the device. The device receives the datagram packet, processes it, creates a response and sends it back on the UDP socket to the TFA-WiFi-App to be forwarded to the server.

The authentication process implemented by TFA-WiFi-App is shown in Figures 8 and 9. The underlying protocol flow is exactly the one used in the FBD-BT-BT mechanism. In the resulting FBD-WF-WF mechanism, similar to FBD-BT-BT, the user is not involved in transferring the challenge to the device (in contrast to LBD-PIN, LBD-QR-PIN or MBD-QR-QR). Moreover, device response PIN is automatically passed to the web page through the applications on the browser and the device, and it can be easily long (at least 128 bits), as in FBD-BT-BT. Hence, besides entering username and password, the user's role is minimized to simply launching the application on the terminal browser and the device. This would provide increased usability and security at the cost of the need for additional software on the client terminal browser (and the Virtual WiFi application in case the terminal dedicates WiFi to an internet connection). Figure 5 shows a snapshot of our implementation in action.

3) FBD-QR-BT and FBD-QR-WF: Our implementation of the FBD-QR-BT and FBD-QR-WF mechanisms simply uses the QR codes for C-to-D communication and BT/WiFi for D-to-C communication. Since BT/WiFi is only used for receiving data (not for sending), this might provide better security against potentially malicious extensions which could leak sensitive information from the client terminal over BT/WF without user's knowledge. Figures 10 and 11 show these mix-capability mechanisms.

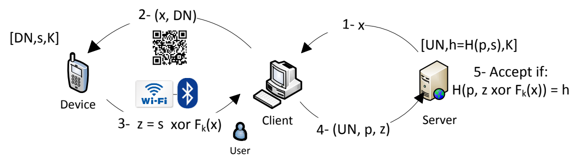


Fig. 10. FBD-QR-BT/FBD-QR-WF (SC)

[8]This does not affect the security of our protocols because we do not require the C-D channel to be secure.

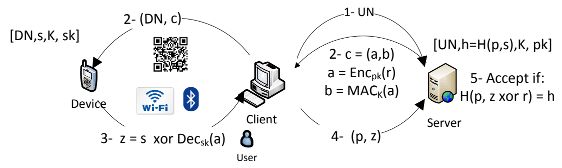


Fig. 11. FBD-QR-BT/FBD-QR-WF (PC)

E. Performance Measurements

To estimate the performance of our TFA prototypes, we measured the execution times corresponding to each operation (except of manual PIN transfer and password entry) as part of the prototypes. These average timing measurements are presented in Table I, along with measurement details, platform and devices used, and other relevant information. It is clear from the table that all server side operations in our implementation are very fast (although we tested these on a personal laptop, not a real server). As one would expect, the device computation is more time-consuming compared to server computation but still not exceeding 114 ms on an average.

Scanning the (C-to-D) QR code takes about 5-8 s depending on whether a public-key challenge (3328 bits) or a symmetric-key challenge (128 bits) is used. The more the information embedded within the QR code, the longer it takes to scan the code. This can be reduced to 3-5 s by using a lower error correction code at the cost of potential for increased errors in code scanning. We have also tested the D-to-C QR channel by trying to interpret 20 to 128-bit long PIN numbers. Evaluation shows that once the user properly aligns the device, QR code can be decoded between 820-950 milliseconds. For a trained user, it takes up to 5 seconds to show the QR code correctly to the webcam. Good webcam quality, device brightness and resolution, and correctly aligning device helps Client ZXing JavaScript to interpret the QR code faster. WiFi 2-way average response time (inclusive of device computation) is not exceeding 550 ms or so. Bluetooth response time, on the other hand, is much longer, about 2 s. This seems to imply that WiFi, when available, is a better communication medium for our application.

In summary, these measurements show that FBD-WF-WF mechanism yields the shortest execution times, which would likely not cause a perceivable delay to the user when authenticating to a service, except of the time taken to input the password and launch the app on the device. The LBD-QR-PIN mechanism may take more than 10 s overall, because the user has to type in the password and launch the application besides scanning the QR code. LBD-PIN takes the same amount of time as the traditional TFA mechanism with bulk of the time spent in copying the PIN and typing the password.

VI. Discussion

In this section, we summarize, and provide a comparison of, our different TFA mechanisms in terms of security, usability and deployability, and contrast them with the traditional TFA mechanism (called Traditional). Table II depicts this summary.

The primary advantage offered by our new TFA mechanisms over Traditional is improved resilience to offline dic-

	Traditional	**LBD-PIN**	**LBD-QR-PIN**	**MBD-QR-QR**	**FBD-BT-BT / FBD-WF-WF**	**FBD-QR-BT / FBD-QR-WF**
Time or Challenge Response (CR)?	Time	Time	CR	CR	CR	CR
Protocol	Traditional	TFA-T	TFA-SC or TFA-PC	TFA-SC or TFA-PC	TFA-SC or TFA-PC	TFA-SC or TFA-PC
Device Type	type I	type I	type II	type III	type IV	type IV
$\|z\|$	19 bits	19 bits	19 bits	t = 20 - 128 bits	128 bits (or more)	128 bits (or more)
Online Attack Success Probability	$1/(\|D\|*2^{19})$	$1/(\|D\|*2^{19})$	$1/(\|D\|*2^{19})$	$1/(\|D\|*2^{t})$	$1/(\|D\|*2^{128})$	$1/(\|D\|*2^{128})$
Offline Attack Overhead	$\leq \|D\|$ hashes	$\leq 2^{19}*\|D\|$ hashes	$\leq 2^{19}*\|D\|$ hashes	$\leq 2^{t}*\|D\|$ hashes	$\leq 2^{128}*\|D\|$ hashes	$\leq 2^{128}*\|D\|$ hashes
Lunch-Time Security?	Yes (but can manipulate clock)	Yes (but can manipulate clock)	Yes	Yes	Yes	Yes
Secure against Lunch-Time/Eavesdrop + Server Compromise?	No	No	Yes (with TFA-PC)	Yes (with TFA-PC)	Yes (with TFA-PC)	Yes (with TFA-PC)
Time-Sync Necessary?	Yes	Yes	No	No	No	No
Client Software/ Special Hardware	HTML Browser/ None	HTML Browser/ None	HTML Browser/ None	HTML5 Browser/ Webcam	Browser Extension/ Bluetooth or WiFi	Browser Extension/ Bluetooth or WiFi
User Effort	Input user name on C	Input user name on C	Input user name on C	Input user name on C	Input user name on C	Input user name on C
			Enter to send user name (with TFA-PC)	Enter to send user name (with TFA-PC)	Enter to send user name (with TFA-PC)	Enter to send user name (with TFA-PC)
	Launch app on D	Launch app on D	Launch app on D	Launch app on D	Launch app on D	Launch app on D
	Select DN	Select DN				
			Take snapshot of QR shown on C	Take snapshot of QR shown on C		Take snapshot of QR shown on C
	Copy z onto C	Copy z onto C	Copy z onto C			
				Take snapshot of QR shown on D		
	Input p on C, and submit (UN, z, p)	Input p on C, and submit (UN, z, p)	Input p on C, and submit UN and/or (z, p)	Input p on C, and submit UN and/or (z, p)	Input p on C, and submit UN and/or (z, p)	Input p on C, and submit UN and/or (z, p)

TABLE II. COMPARISON OF DIFFERENT TFA MECHANISMIS (*cells with lighter shades represent positive features*)

tionary attacks. Specifically, our LBD mechanisms provide a 2^{19} factor improvement ($|z| = 19$ bits) and MBD/FBD mechanisms provide a 2^{128} factor or more improvement ($|z| \geq 128$ bits). This is a significant improvement since offline dictionary attacks allows an attacker to compromise (and undermine the security of) the passwords of multiple user accounts (these passwords might be re-used on another service). Clearly, FBD and MBD mechanisms provide a significantly stronger (computationally sufficient) protection compared to LBD mechanisms. In terms of security against online attacks also, FBD mechanisms are significantly stronger, whereas LBD mechanism and traditional scheme have the common fundamental limitation – the one-time PINs can not be longer than 19 or so bits due the requirement of manual PIN transfer.

Clearly, LBD-PIN and Traditional are time-based schemes whereas others are challenge-response. As such, these two schemes require time synchronization between D and S. Establishing and maintaining such a synchronization can be challenging in practice, and any loss of synchronization will result in the user not being able to authenticate to the service. Google Authenticator has a time-synch feature which can be used to re-synchronize the device with the server but requires network connectivity at the time of re-synchronization. Another limitation of the time-based mechanisms is that they are not secure against a lunch-time attacker. This attacker can manipulate the timestamp on D to future values, record the PINs corresponding to those values, reset the timestamp, and then use the recorded PIN values in the future to authenticate on behalf of the user. This suggests that challenge-response (CR) mechanisms might be more appealing in practice. Such mechanisms also have an usability advantage in that the domain name (DN) of the site can be embedded within the challenge and sent to D, which then automatically locates the account information (e.g., (K, s)). The time-based mechanisms, in contrast, requires the user to manually select the DN/account on D.

In terms of usability also, FBD mechanisms have an edge over other mechanisms because the user does not need to manually transfer the PIN from D to C. Between mechanisms that use WF or BT, the former seems preferable due to short response time (BT response time was much higher as shown by our performance measurements in Section V-E). MBD-QR-QR mechanism does not involve a manual PIN transfer, but still needs the user to take a snapshot of QR displayed on the phone using webcam, which may have usability implications compared to the FBD mechanisms, although mobile device QR "reading" is also becoming popular (e.g., in mobile payment systems [2], [8]).

Where the FBD mechanisms may have a limitation compared to all other mechanisms is in their requirement of extra hardware (Bluetooth or WiFi, or webcam) and extra software (browser extension) on C. Traditional and LBD schemes, in contrast, all just work with a plain browser and no special hardware interfaces, which can be a prominent advantage in practice due to deployability reasons.

Comparing the LBD schemes, we can claim that LBD-PIN could immediately replace Traditional because the former offers all the same properties but significantly better resilience to offline dictionary attacks. Overall, for other schemes, our analyses suggest that each scheme has its own advantages which would make it attractive to be deployed per the desired requirements of the application and usage scenario at hand.

VII. Conclusion

We provided a formalization of two-factor authentication, and designed novel TFA mechanisms built on top of four TFA protocols resilient to server compromise. These mechanisms leveraged a wide range of capabilities of devices and client terminals, ranging from a plain display to camera and wireless interfaces, which enable mix-bandwidth unidirectional or bidirectional device-client communication. All these mechanisms offer different level of security and usability advantages.

As per our overall analysis of these mechanisms and protocols, we provide the following recommendations. If a compatible browser extension and radio interface are available on the client, the FBD-WF-WF mechanism would offer the highest level of security and usability, followed by FBD-BT-BT. The versions of these mechanisms based on TFA-PC provides the strongest security guarantees. The TFA-SC versions may also be preferable due to usability reasons (in particular, the advantage of submitting (UN, p, z) in a single login page, which TFA-PC-Alt also provides). FBD-QR-WF or FBD-QR-BT could also be used if malicious browser extensions, that may exploit outgoing radio communications, is a concern. If a browser extension is not supported and time-synchronization between the device and server is difficult, LBD-QR-PIN would be a good choice, followed by MBD-QR-QR. If browser extensions are not supported and time-synchronization (and re-synchronization) is feasible, as assumed in the currently deployed TFA systems, LBD-PIN should be used instead. LBD-PIN offers all the same advantages of traditional systems plus improved resilience to offline dictionary attacks and can immediately replace the traditional deployment.

Acknowledgments

We thank NDSS'14 anonymous referees for their feedback. This work was supported in part by an NSF grant CNS-1209280.

References

[1] "Anonymous hackers claim to leak 28,000 paypal passwords on global protest day," Available at: http://thenextweb.com/insider/2012/11/05/anonymous-leaks-sensitive-data-from-alleged-paypal-hack-on-global-protest-day/.

[2] "Bitcoin," Available at: http://bitcoin.org/en/.

[3] "Blizzard servers hacked; emails, hashed passwords stolen," Available at: http://www.electronista.com/articles/12/08/09/lost.data.reportedly.insufficient.to.allow.illicit.battlenet.access/.

[4] "Celestix HotPin," http://www.celestixworks.com/HOTPin.asp.

[5] "Duo Security Two-Factor Authentication," https://www.duosecurity.com/.

[6] "Google Authenticator for Two-Step Verification," http://code.google.com/p/google-authenticator/.

[7] "Lazarsoft javascript qr code open source implementation," Available at: https://github.com/LazarSoft/jsqrcode.

[8] "Level up: Pay with your phone," Available at: https://www.thelevelup.com/.

[9] "Linkedin confirms account passwords hacked," Available at: http://www.pcworld.com/article/257045/6_5m_linkedin_passwords_posted_online_after_apparent_hack.html.

[10] "Microsoft PhoneFactor," https://www.phonefactor.com/.

[11] "Microsoft research – virtual wifi: Connecting to multiple ieee 802.11 networks with one wifi card," Available at: http://research.microsoft.com/en-us/um/redmond/projects/virtualwifi/.

[12] "NIST – cryptographic algorithms and key sizes for personal identity verificationdecember," Available at: http://csrc.nist.gov/publications/nistpubs/800-78-3/sp800-78-3.pdf.

[13] "RSA SecureID – World's Leading Two-Factor Authentication," http://www.emc.com/security/rsa-securid.htm.

[14] "Unimate Authentication Token with Connectivity to Smartphones using Audio Jack," http://www.esecutech.com/en/products/unimate/unimate-std.html.

[15] "ZXing: Multi-format 1D/2D barcode image processing library with clients for Android, Java," http://code.google.com/p/zxing/.

[16] A. Adams and M. A. Sasse, "Users are not the enemy," *Commun. ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[17] M. Bellare, "New proofs for nmac and hmac: security without collision-resistance," in *Proceedings of the 26th annual international conference on Advances in Cryptology*, ser. CRYPTO'06, 2006.

[18] D. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. van Dijk, Srinivas, and R. Rivest, "The untrusted computer problem and camera-based authentication," in *Pervasive Computing*. Springer-Verlag, 2002, pp. 114–124.

[19] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz, "Strengthening user authentication through opportunistic cryptographic identity assertions," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[20] M. Mannan and P. C. van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *Financial Cryptography*, 2007.

[21] D. McCarney, D. Barrera, J. Clark, S. Chiasson, and P. C. van Oorschot, "Tapas: design, implementation, and usability evaluation of a password manager," in *Annual Computer Security Applications Conference (ACSAC)*, 2012.

[22] R. Morris and K. Thompson, "Password security: a case history," *Commun. ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[23] B. Parno, B. Parno, C. Kuo, C. Kuo, A. Perrig, and A. Perrig, "Phoolproof phishing prevention," in *In Proceedings of Financial Cryptography (FC06*, 2006.

[24] N. Saxena and J. Voris, "Exploring mobile proxies for better password authentication," in *International Conference on Information and Communications Security (ICICS)*, 2012.

[25] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *IEEE Security and Privacy*, vol. 2, no. 5, pp. 25–31, 2004.