Device-Enhanced Password Protocols with Optimal Online-Offline Protection

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

Hugo Krawczyk
IBM Research
hugo@ee.technion.ac.il

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

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

ABSTRACT

We introduce a setting that we call *Device-Enhanced PAKE (DE-PAKE)*, where PAKE (password-authenticated key exchange) protocols are strengthened against online and offline attacks through the use of an auxiliary device that aids the user in the authentication process. We build such schemes and show that their security, properly formalized, achieves *maximal-attainable* resistance to online and offline attacks in both PKI and PKI-free settings. In particular, an online attacker must guess the user's password *and* also corrupt the user's auxiliary device to authenticate, while an attacker who corrupts the server *cannot* learn the users' passwords via an offline dictionary attack. Notably, our solutions do not require secure channels, and *nothing* (in an information-theoretic sense) is learned about the password by the device (or a malicious software running on the device) or over the device-client channel, even without any external protection of this channel. An attacker taking over the device still requires a full *online* attack to impersonate the user. Importantly, our DE-PAKE scheme can be deployed at the user end without the need to modify the server and without the server having to be aware that the user is using a DE-PAKE scheme. In particular, the schemes can work with standard servers running the usual password-over-TLS authentication.

We use these protocols to implement a practical DE-PAKE system and we evaluate its performance. To improve usability the implemented system utilizes automated and *user-transparent data channel* between the mobile device and the client, falling back to localized communication if the device looses primary connectivity.

1. INTRODUCTION

Today, passwords constitute the prevalent authentication mechanism for bootstrapping security in most online applications (and many offline systems). A plethora of sensitive information stored in many different contexts therefore depends on the security of password-based authentication. However, passwords are vulnerable to both online and offline dictionary attacks that build on password dictionaries from which a significant portion of passwords are chosen. Candidate passwords for authenticating a user to a server can be tested by an attacker through online interactions with the server. Furthermore, an attacker breaking into a server can mount an offline attack that uses information stored on the server (typically, a salted one-way mapping of the password) to test the different passwords in the dictionary. Such offline dictionary attacks are a serious concern, especially in light of frequent attacks against major commercial vendors recently, such as PayPal [1], LinkedIn [4], Blizzard [2] and Gmail [3]. The offline attacks are particularly devastating because a single server break-in may lead to compromising a huge number of user accounts [7]. Furthermore, since many users re-use their passwords across multiple services, compromising one service may compromise user accounts at other services.

In this paper we present solutions which enhance password protocols against *both online and offline attacks* by an active man-in-the-middle attacker acting on user-server and user-device links, and capable of compromising devices and servers by learning their full internal state (e.g., server's password file and device's secrets).

1.1 Our Contributions

We introduce a setting of *Device-Enhanced PAKE (DE-PAKE)*, where PAKE (password-authenticated key exchange) protocol is strengthened against online and offline attacks through the use of an auxiliary device which aids the user in the authentication process. We build such scheme and show that its security, properly formalized, achieves *maximal-attainable* resistance to online and offline attacks in both PKI and PKI-free[1] settings. Moreover, our DE-PAKE scheme can be deployed at the user end without the need to modify the server and without the server having to be aware that the user is using a DE-PAKE scheme. In particular, in the PKI setting the scheme can work with unmodified servers running the usual password-over-TLS authentication. Finally, while we focus the presentation on a setting where the auxiliary device is a physical personal device, e.g. a phone, our protocols apply also to the setting where the device entity is implemented by an auxiliary on-line service. Indeed, since our solution is secure against a fully capa-

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

DOI: http://dx.doi.org/10.1145/2897845.2897880

[1]The PAKE notion was originally used for the peer-to-peer password setting (where the two peers share a password) but was later extended to the client-server setting too. In the latter case one differentiates between the PKI case, where the client possesses a server's certificate in addition to the password, and the PKI-free case where no such certificate, or other form of secure channels, is assumed for user login. Password registration, being much less frequent than regular login, can use special safeguards, such PKI-based interactions or out-of-band channels.

ble man in the middle and provides strong protection even in case of device corruption, an implementation that uses an auxiliary web service in the role of the device is protected against both network attacks and against the compromise of this service.

Secure and Efficient DE-PAKE Protocols: We introduce efficient DE-PAKE protocols with the following properties:

• *Resistance to online and offline attacks:* Our DE-PAKE schemes provide *maximal-attainable* security in terms of resistance to both online and offline attacks. That is, the only attack allowed by the scheme is the *unavoidable* online guessing attack where the attacker tests if a given value p is the user U's password by interacting with *both* device and server in the role of U with password p and observing whether S accepts. In other words, each guess attempt requires the attacker to interact on both U-D and U-S links. No amount of attacking on one link helps without a corresponding attack on the other. Moreover, fully compromising D still requires a full online guessing attack on the U-S link and fully compromising the server requires a full online guessing attack on the U-D link. And even if both S and D are compromised, a full offline dictionary attack is required. More formally, to have an impersonation probability of $q/|\mathsf{Dict}|$, where Dict is the passwords dictionary, the attacker needs to either run q online interactions with D *and* q online interactions with S, or q online interactions with U impersonating both D and S. Moreover, even when compromising D and finding all its secrets, the attacker still needs to run q online interactions with either S or U, and if the server is compromised, q online interactions with D or U. Finally, if both D and S are compromised the adversary must stage a full offline dictionary attack to learn any passwords. We formalize these properties through a security model that extends the traditional PAKE security setting, and then we use this model to prove the security of our schemes.

• *PKI-agnostic:* The above security is achieved even when the user-server channel is not protected by a server public key. On the other hand, when such protection is available one obtains the additional benefit that impersonating the server to the user is infeasible even if the user's password is disclosed. Luckily, our schemes can work, without modification and without having to be aware of it, with PKI-based and PKI-free authentication protocols, providing in each case the best possible offline-online protection.

• *Modularity and server-transparency:* Our design is modular allowing for the use of independent device and server components, in particular enabling the use of our scheme with existing password protocols and *without the need to modify the server side.*

DE-PAKE Systems Design and Evaluation: To demonstrate the flexibility provided by our modular approach, we design, develop and test a concrete instantiation of DE-PAKE scheme in the PKI-free model. We develop server-side (PHP/JavaScript), client-side (Chrome browser extensions) and device-side (Android app) implementations, and measure their computational performance and communication delays. A key component of DE-PAKE system is the D-U communication channel. To support this, we propose a "hybrid model" whereby the primary channel is based on fast and *user-transparent* data channel, and the secondary channel involves localized WiFi communications. Since the secondary channel can be potentially difficult for the user to establish, we use it only as a fall-back when the primary channel is not available (e.g., when the user's phone is out of network coverage). This approach will significantly improve the user experience underlying DE-PAKE logins. We design, implement and evaluate the performance of our DE-PAKE construction (Section 6).

1.2 Related Work

Prior Work on Device-Enhanced Authentication. Our approach is closely related to the work of Acar, Belenkiy, and Kupcu [8], and Boyen [13]. A close variant of the key ingredient of our solution, a "Password-to-Random" (PTR) protocol, was proposed as a "Hidden Credential Retrieval" (HCR) scheme by [13]. Our DE-PAKE protocol has similar goals as the "Single Password Authentication" (SPA) scheme of [8], namely, strengthening password authentication by making offline dictionary attack against the server infeasible. Moreover, one of their schemes does so in roughly the same way as we do, following an approach first suggested by Ford-Kaliski [17], i.e. having the user "strengthen" her password into a strong secret via the HCR/PTR protocol with an auxiliary device (or service), and then use this strong secret to authenticate to the primary server. Despite these similarities, our work improves on [8, 13] in the following aspects: (1) Whereas [13] shows that HCR can be realized almost identically to our PTR, i.e. using the same variant of the Ford-Kaliski protocol [17], the SPA scheme of [8] did not show a general compiler from HCR, but assumed a stronger tool of unique blind signature which is costlier to realize (moreover, in their scheme the user-device blind signature protocol goes over TLS, while in ours the PTR protocol goes over insecure links); (2) The SPA scheme of [8] is weaker than our DE-PAKE, in that it achieves only entity authentication rather than authenticated key exchange as in our case and it assumes PKI while our DE-PAKE works in both PKI-free and PKI settings;[2] (3) Very importantly, our modular approach that treats PTR and PAKE as independent components and has no involvement of the server in the PTR execution, allows us to obtain DE-PAKE security without any modification to existing servers (e.g., those running password-over-TLS authentication). In contrast, the SPA scheme of [8] requires the server to perform additional public key operations.

Relation to Work on Two-Factor Authentication. One important and increasingly common line of defense against password attacks is the use of *two-factor password authentication (TFA) schemes.* TFA mechanisms are used for authenticating a user U to a server S, and establishing a session key between the two, where the user has a password and a personal device D (e.g., a smartphone) that contains some secret auxiliary information. This secret information is used to increase the security of password authentication by preventing online attacks for an adversary that does not have access to D. Typically, D displays a short one-time PIN (OTP), received directly from the server or computed by D based on a key shared with the server, that the user manually copies over to the authentication terminal in addition to providing her password.

Traditionally, these TFA schemes have been used for increasing resistance to online dictionary attacks. However, as argued by Shirvanian et al. [29], with the increasing vulnerabilities of servers to compromise [5, 6], TFA schemes should also be enhanced to strengthen security against offline attacks. The work of [29] presented several schemes for achieving this goal. There are, however, two important aspects in which the schemes of [29] can be improved, and we do so in this paper. First, all their schemes assume a PKI setting, namely, the user (through a client application) must be in possession of an *authentic* public key of the server which is used to establish a secure channel, e.g., via TLS. If such public key is not available or is compromised, the security of the scheme completely breaks down. Given the vulnerabilities of PKI to certification failures and man-in-the-middle (MitM) attacks (either due to

[2]Our DE-PAKE security model is also more precise, e.g. letting adversary observe whether user authentication succeeds or not.

programmatic errors or human mistakes), e.g., [15,20,30], reducing the dependency of authentication security on public keys is an increasingly important goal. Second, the schemes of [29] require the authentication server to run a different protocol than currently standard PKI-setting TFA schemes which can inhibit the deployment of such schemes. For example, an individual user (or an application) cannot adopt the schemes from [29] to protect her password stored by a web service without that service being modified to support the new TFA method.

The notion of DE-PAKE scheme we propose (and its efficient realization we construct) addresses many of the same concerns as TFA schemes, e.g. increased security against *on-line* attacks viz-a-viz solely-password based authentication, and it surpasses security offered by existing TFA schemes by offering maximal resistance to *off-line* attacks, i.e. making offline dictionary search impossible in case of server compromise. However, traditional TFA schemes offer also increased security against compromise of the client machine (compared to solely-password authentication). A generic DE-PAKE scheme does not provide this, although the specific DE-PAKE scheme we propose can be extended to offer additional defenses against attacks on the client (see Section 5). Moreover, it is possible to combine a DE-PAKE scheme with a traditional MFA mechanism (e.g. a one-time PIN delivered to or generated by the hand-held device) to extend the security properties of DE-PAKE to include the same level of resistance against client compromise as offered by traditional TFA schemes.

1.3 Organization and Glossary

We overview our design and proof methodology in Section 2, with full details given in Section 4. Section 3 presents our formal DE-PAKE security model on which we base our analysis. Section 5 presents protocol instantiations and extensions: A fully specified DE-PAKE scheme secure in the PKI-free (i.e. CRS) model; a description of our approach for armoring existing servers against on-line and offline attacks while keeping the servers unmodified; and extensions which address issues related to client security (which are not covered in our DE-PAKE model). In Section 6 we present our system and implementation work and report on performance measurements. For easy reference, in Table 1 we provide a summary of the main terms and acronyms used throughout the paper.

2. OVERVIEW: DESIGN AND ANALYSIS

Our design follows the "password hardening" approach of Ford and Kaliski [17], but dispenses of authenticated channels (other than during a registration phase), multiple servers and/or other safeguards that were required for the secure use of these techniques in prior work [17, 21].

The idea is simple: the user memorizes a regular password pwd but uses as her password with server S a value $\mathsf{rwd} = F_k(\mathsf{pwd})$ where F is a pseudorandom function and k is a key held by device (or an auxiliary on-line service) D (rwd is a mnemonics for "randomized password"). Before authenticating to S, U contacts D (through a client application) and obtains rwd via a special protocol with D (in which D learns *nothing* about pwd or rwd). U then authenticates to S via a (standard) PAKE protocol using rwd as a password. Note that without knowledge of k, the value rwd has full entropy (in the range set of function F) hence dictionary attacks do not apply against rwd (neither online or offline attacks, not even if the server is compromised). Moreover, we will ensure that even if D (or S) is compromised, offline attacks against pwd are infeasible. Thus, the challenge is in implementing the protocol between U and D, to which we refer as *PTR (for Password-to-Random)*, so that U can compute $F_k(\mathsf{pwd})$ but the protocol leaks no other information about protocol inputs, i.e. pwd and k. Note that we cannot assume an authenticated or secret channel between the client and D since this would either require knowing a device public key or storing pwd-related information at D (the latter would open pwd to an offline dictionary attack upon compromising D). We show that in spite of the client-device link being unauthenticated, hence controlled by the attacker, the "blinded DH" approach of Ford-Kaliski, (see Section 4.1), can be used to implement the PTR protocol.

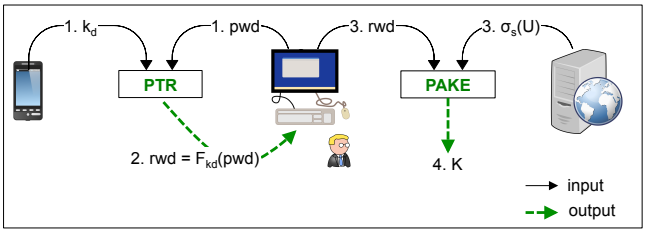

Figure 1: PTR-PAKE Authentication Phase

Hence, we obtain a DE-PAKE scheme, which we call *PTR-PAKE*, as the composition of a PTR protocol and a secure PAKE. We depict this generic construction of a DE-PAKE protocol in Figure 1. The output K in step 4 denotes the session key established between the user and the server, i.e. the output of the DE-PAKE protocol, which is equal to the output of the PAKE subprotocol. The server's input $\sigma_{\mathsf{S}}(\mathsf{U})$ to the PAKE subprotocol denotes the user-specific information stored at S created in the PAKE initialization using rwd in the role of the password (e.g. in a common PKI-based PAKE $\sigma_{\mathsf{S}}(\mathsf{U})$ would be a salted hash of rwd).

In order to prove the security of such scheme, we first extend the established security models for the PAKE functionality to the DE-PAKE setting (Section 3). For the DE-PAKE modeling we consider a fully capable man-in-the-middle attacker active on all the links between all parties and one who is allowed to compromise servers and devices at will. No external source of authentication is assumed other than the user's password (except for secure registration of a user with the server and device). Second, we define the security requirements from a PTR protocol and show a PTR instantiation, FK-PTR, that satisfies this definition in the random oracle model. Finally, we prove a *generic security composition theorem* showing that the composition of a secure PTR scheme (run between U and D on the basis of the user's password pwd) with a secure PAKE protocol (run between U and S on the basis of the hardened password $\mathsf{rwd} = F_k(\mathsf{pwd})$) results in a secure DE-PAKE scheme, provided that the PAKE protocol satisfies "security against server compromise" or the more precise notion of KCI resistance introduced and discussed in Section 3.1 (we note that typical protocols that store a salted version of the password satisfy this property).

Since our FK-PTR scheme does not require PKI, using a PKI-free PAKE in the above composition results in a DE-PAKE protocol that does not rely on public keys or other secure channels.

In order to demonstrate full standalone solutions, in Section 5 we describe two instantiations of our PTR-PAKE construction of a DE-PAKE scheme. First, we compose our FK-PTR scheme with a specific PAKE protocol - a KCI-resistant version of the single-server variant of threshold PAKE from [22, 23]. Since this PAKE protocol does not require PKI, neither does the resulting DE-PAKE protocol. Secondly, we expand on the fact that since our PTR-PAKE construction can be used with *any* existing password protocol with resistance to KCI attacks, one obtains DE-PAKE schemes

Table 1: Glossary

Acronym	Description
PAKE	Password-authenticated key exchange. We refer to password protocols by PAKE as well as to their security model (recalled in Section 3.1).
PKI-Free PAKE	A PAKE protocol that does not assume any secret or authenticated key carried by the user other than its own password. In particular, no PKI-based server-authentication is assumed. This is also known as the CRS or password-only model.
DE-PAKE	A new notion of *Device-Enhanced PAKE* protocols that guarantees optimal resilience to offline and online attacks upon compromise of device and/or server. We use DE-PAKE to refer to the security model (Section 3.2) as well as to the constructions satisfying this model (Sections 4 and 5) .
PTR	A security notion and model for password hardening protocols. We use PTR (password-to-random) to refer to the security model (Section 4.2) as well as to the constructions satisfying this model (Section 4.1).
FK-PTR	Specific PTR construction using the Ford-Kaliski password hardening technique (Section 4.1).
OPRF	Oblivious PRF (see Section 4.1) is the basis for the FK-PTR protocol. When implemented via the function $F_k(x) = H(x, (H'(x))^k)$ we obtain FK-PTR.
PTR-PAKE	A general name for DE-PAKE protocols built by composing a PTR and a PAKE protocols (their generic security is based on Theorem 3)
FK-PTR-PAKE	A PTR-PAKE scheme where the PTR part is implemented with the FK-PTR construction. It is also the name of the protocol in Fig 3 that combines FK-PTR with the PKI-free PAKE protocol of [22, 23].

without changing the server that implements the PAKE protocol. In particular, this allows us to use without any change a server that implements the standard PKI-based password-over-TLS protocol.

A PTR scheme is a close variant of OPRF, and the potential of OPRF to strengthen password authentication was recently used in the Pythia system [16], but their proposal differs from ours in at least three ways: (1) Their OPRF scheme is significantly more costly than our PTR: it requires 6 exponentiations and a bilinear map compared to 3 exponentiations in our PTR scheme (and we can use elliptic curves without bilinear maps where exponentiations are cheaper); (2) Their solution relies on PKI, while ours does not; (3) Their solution does not offer a server-transparent instantiation.

3. SECURITY MODEL

We introduce the *Device-Enhanced PAKE (DE-PAKE)* security model under which we prove the security of our schemes. The model extends the standard PAKE (Password Authenticated Key Exchange) formalisms to include user-specific devices and formulates a security definition that guarantees maximal online and offline security of password protocols. We start by recalling the PAKE security model (adapted to the client-server setting) and then we present the extension of the PAKE model to the DE-PAKE setting.

3.1 PAKE Security Model

We recall the security model for PAKE (Password-based Authenticated Key Exchange) protocols, based on the model of Bellare, Pointcheval and Rogaway [10] that extends authenticated key exchange models to account for the inherent vulnerability of password protocols to online guessing attacks. We adapt the PAKE model to the client-server setting borrowing some of the formalism from [22], and we recall an extension of this standard PAKE model to security against server compromise.

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

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

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

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

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

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

compromise(S, U): Outputs state $\sigma_S(U)$ at S. S is U-*compromised.*

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

The following notion taken from [22] is used in the security definition below to ensure that legitimate messages exchanged between honest parties do not help the attacker in online password guessing attempts (only adversarially-generated messages count towards such online attacks). It has similar motivation as the execute query in [10], but the latter fails to capture the ability of the attacker to delay and interleave messages from different sessions.

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

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

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

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

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

Definition 1. *A PAKE protocol* PAKE *is* (q_S, q_U, T, ϵ)-secure *if it is correct and for any password dictionary* Dict *and any attacker* A *that runs in time* T*, it holds that* $\mathsf{Adv}_{\mathsf{A}}^{\text{PAKE}} \leq \frac{q_U + q_S}{|\text{Dict}|} + \epsilon$ *where* q_U *is the number of rogue* send *queries having the target user* U *as recipient and* q_S *is the number of rogue* send *queries having the target* S *as recipient.*

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

Security against server compromise and KCI-resistance. In the asymmetric client-server setting of password authentication that concerns us in this paper, plain passwords should not be stored at the server, so as to prevent the leakage of the password in case of server compromise. Instead, the server should store some other verification information corresponding to this password, such as the salted password hash. The security requirement in this case, often referred to as *security against server compromise* [19], is that access to the server's state for a particular user (i.e, U-compromise in our terminology) does not allow the attacker to authenticate that user to the server except after running an offline dictionary attack that recovers the password given the server's state.[3] In the key-exchange literature an attack in which the compromise of a party P allows the attacker to falsely authenticate another party P' to P is called a *Key-Compromise Impersonation (KCI) attack* [12]. Therefore, the above notion of security against server compromise can be seen as a *weak form of KCI resistance,* where impersonation of U to S is possible but only after running an offline dictionary attack.

We extend the above PAKE formalism to capture resistance to weak KCI attacks (*wKCI-resistance*) through the following game (which is well-suited to ROM-based implementations that hash the password). The security experiment is as before except for the following changes. User U (associated with server S) chooses its password at random from a dictionary Dict, where Dict is a random subset of $\{0,1\}^\tau$ of size 2^d (for integers $d<\tau$). The attacker A is given a random subset of Dict of size q as well as the server's state $\sigma_S(U)$, and it must choose the test session at an instance of S with peer U (in the regular case this is not allowed since S is U-compromised). We call a PAKE scheme ϵ-*wKCI-resistant* if for any $q \leq 2^d$, the attacker's advantage in this game is at most $q/2^d + \epsilon$.

A strong notion of KCI resistance is achieved in the DE-PAKE model as we will see next.

3.2 DE-PAKE Security Model

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

The attacker's goal is the same as before, i.e. to win the test experiment at a user or server instance, as in the PAKE setting. Also the correctness property is unchanged. However, to the attacker resources we add the number of *rogue* send queries (see Section 3.1) where the target user is the recipient and the device the sender (denoted q'_U) and the number of *rogue* send queries where the target user is the sender and the device the recipient (denoted q_D). We refer to this more powerful adversary as a DE-PAKE attacker.

Strong KCI resistance. The DE-PAKE model is intended to provide a much stronger notion of security in case of server compromise than achievable in the PAKE case. While in the latter, impersonating U to S in case of U-compromise is possible (and unavoidable) through an offline dictionary attack, in DE-PAKE protocols

[3]Recovering the password via an offline dictionary attack is unavoidable in the PAKE model. Also unavoidable is impersonating S to U when S is U-compromised (except if one assumes, as in the PKI model, an independent authenticated channel from S to U).

this is prohibited. In order to formalize this requirement we follow the treatment of KCI resistance from [27] and we strengthen the capabilities of a DE-PAKE attacker through a more liberal notion of fresh sessions at a server S. All sessions considered *fresh* in the PAKE model are also considered fresh in the DE-PAKE model; in addition, in the DE-PAKE model, a session Π_i^{S} at server S with peer U is considered fresh *even if* $\mathsf{corrupt}(\mathsf{S})$ *or* $\mathsf{compromise}(\mathsf{S},\mathsf{U})$ *were issued* as long as all other requirements for freshness are satisfied and *the attacker* A *does not have access to the temporary state information created by session* Π_i^{S}. This relaxation of the notion of freshness captures the case where the attacker A might have corrupted S and gained access to S's secrets (including long-term ones), yet A is not actively controlling S during the generation of session Π_i^{S}. In this case we would still want to prevent A from authenticating as U to S on that session. Definition 2 (item 2) below ensures that this is the case for DE-PAKE secure protocols even when *unbounded* offline attacks against S are allowed.

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

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

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

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

2. *If* D *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathrm{DPK}} \leq (q_U + q_S)/2^d + \epsilon$.
3. *If* S *is corrupted then* $\mathsf{Adv}_{\mathsf{A}}^{\mathrm{DPK}} \leq (q'_U + q_D)/2^d + \epsilon$.
4. *When both* D *and* S *are corrupted, expression (1) holds but* q_D *and* q_S *are replaced by the number of offline operations performed based on* D*'s and* S*'s state, respectively.*

Note that the bounds in items 3 and 4 hold also when S is U-compromised (since being corrupted implies U-compromise for all users U associated with S).

Note on modeling DE-PAKE via a (2,2)-TPAKE. In a $(t+1, n)$ threshold-PAKE (TPAKE) (cf. [22]), a user holding a single password can securely establish authenticated keys with a subset of n servers as long as no more than t of them are corrupted (and the user interacts with at least $t+1$ well-behaving servers). One can implement DE-PAKE on the basis of (2,2)-TPAKE by letting D and S act as the two servers in a (2,2)-TPAKE scheme (this would imply the first three conditions of Definition 2 but the last one should be added as an additional requirement). Moreover, one can use (2,2)-TPAKE as the basis for the definition of DE-PAKE where the user only authenticates to one of the parties. However, the dedicated DE-PAKE definition we present, and its instantiations, provide several advantages: (1) It makes the security goals for the DE-PAKE notion clearer; (2) It allows for a more precise specification of the (strict) upper bounds on attacker's advantage depending on the attack setting; and (3) It allows for more efficient implementations, in particular enabling a server-transparent DE-PAKE implementation, which cannot be done using TPAKE. (A TPAKE cannot be server-transparent because if S runs the code as in PAKE then S's presence cannot help U to authenticate to D.)

Note on client security. The DE-PAKE model is designed to capture (maximal) security against online and offline attacks where the attacker fully controls all communication channels and can compromise servers and devices. However, as it is customary in the PAKE setting, the model does not consider the security of the machine (the "client") into which the user enters the password. Yet, our solutions, while vulnerable to some forms of attack by an attacker controlling the client machine, also provide defenses to common attacks such as keyloggers or phishing attacks (see Section 5).

4. A MODULAR DE-PAKE SCHEME

In this section we present and analyze our generic DE-PAKE scheme, i.e. the PTR-PAKE shown in Figure 1, which results from the composition of two independent cryptographic primitives, a PTR protocol and a PAKE protocol with resistance to wKCI attacks (see section 3.1). For a high-level description of the functionality of a PTR (password-to-random) scheme and its use for obtaining a DE-PAKE scheme see Section 2. We start by describing a specific efficient PTR implementation we call FK-PTR, with is based on the "password hardening" protocol of Ford-Kaliski [17] (Section 4.1). We then use this protocol example to formalize the PTR notion and its security requirements (Section 4.2), and we prove that the FK-PTR protocol satisfies the PTR security notion (Section 4.3). Finally, we prove that the *generic* composition of any secure PTR scheme and any PAKE scheme with resistance to wKCI attacks results in a secure DE-PAKE scheme (Section 4.4). Thus, our scheme can be instantiated with the FK-PTR scheme as the PTR part and any secure wKCI-resistant PAKE protocol (e.g., [19, 22]). Moreover, if the PAKE scheme is in the password-only model[4] then the DE-PAKE scheme is also secure in this model.

4.1 The FK-PTR Scheme

The instantiation of a PTR scheme we call FK-PTR is based on Ford-Kaliski's "password hardening" [17] or its more general interpretation as an Oblivious PRF (OPRF) [18, 25, 26]. Roughly, an OPRF is a pseudorandom function that is computed by two parties, one that holds the key to the function and learns nothing from the computation, and one that holds an input and learns the output of the function on that input and nothing else.[5] In Figure 2 we present a particular instantiation of the PTR-PAKE protocol, which we call FK-PTR-PAKE, that results from a composition of FK-PTR, which is a specific instantiation of a PTR scheme, with a PAKE scheme. Figure 2 fully specifies the FK-PTR protocol, which is an interaction between U and D by which U retrieves a random value rwd with the help of its password pwd. At initialization, U chooses and remembers password pwd while D chooses and stores $k \leftarrow Z_q$. To retrieve rwd, U first blinds pwd by raising the hashed value $H'(\mathsf{pwd})$ to a random exponent ρ, and send it to D. This perfectly hides pwd from D and from any eavesdropper on the $\mathsf{U} - \mathsf{D}$ link. D checks that the received value is in the group G and if so it raises it to the secret exponent k. Now, U can de-blind this value by rais-

[4]This model assumes that user/password registration is implemented over secure channels but user authentication after registration does not assume public keys or secure channels for any party in the system - only the existence of public parameters, e.g., for defining an elliptic curve, is assumed. These parameters are common to all users and are part of the client program run by a user; they require the same integrity guarantees as the program itself.

[5]We use the PTR notion instead of existing OPRF definitions because game-based OPRF notions, e.g. [18, 25], do not seem to enable our PTR-to-DEPAKE compiler, while the UC OPRF of [22] includes verifiability, which is costlier to achieve than PTR. However, it is possible that the recent UC notion of *non-verifiable* OPRF [23] would suffice in our compiler. (Note that the FK-PTR scheme is identical to the OPRF construction in [23].)

Setup

- *Group G.* The scheme works over a cyclic group G of prime order q, $|q| = \ell$, with generator g.
- *Hash functions H, H'* map arbitrary-length strings into elements of $\{0,1\}^\tau$ and G, respectively, where τ is a security parameter.
- *OPRF.* For a key $k \leftarrow Z_q$, we define function F_k as $F_k(x) = H(x, (H'(x))^k)$.
- *Parties.* User U, Device D, Server S.
- *Dictionary* Dict of size 2^d (a power of 2 is used for notational convenience only).
- Any PAKE protocol Π.

Initialization Phase *(assumed to be executed over secure links)*

- FK-PTR Initialization: U chooses password $\mathsf{pwd} \leftarrow \mathsf{Dict}$; D chooses and stores OPRF key $k \leftarrow Z_q$; U interacts with D to compute $\mathsf{rwd} = F_k(\mathsf{pwd})$.
- PAKE Initialization: User U and server S are initialized with value rwd used as a password according to the specification of PAKE protocol Π.

Login Phase

- **User-Device Interaction (FK-PTR)**
  1. U chooses $\rho \leftarrow Z_q$; sends $\alpha = (H'(\mathsf{pwd}))^\rho$ to D.
  2. D checks that the received $\alpha \in G$ and if so it responds with $\beta = \alpha^k$.
  3. U sets $\mathsf{rwd} = H(\mathsf{pwd}, \beta^{1/\rho})$.
- **User-Server Interaction (PAKE)**

  Follows the specification of the PAKE protocol Π where U uses rwd as its password.

Figure 2: The FK-PTR-PAKE Scheme

ing it to the power $1/\rho$ to obtain $H'(\mathsf{pwd})^k$. Finally, U hashes this value with pwd to obtain the randomized password rwd.

Note that D contains no information related to pwd hence an attacker interacting with D or even breaking into it learns nothing about pwd. Also, U does not run any test on the value reconstructed in the FK-PTR protocol. Hence, an attacker that interacts with U in the role of D does not learn anything about pwd from watching the behavior of U. These "obliviousness" and minimality properties of FK-PTR are essential to achieve PTR security and make the security analysis challenging. We will use this scheme to motivate the security requirements from a PTR scheme as needed for composing it with a PAKE protocol and obtain a secure DE-PAKE protocol. We establish these requirements in the next subsection and then prove the security of FK-PTR.

4.2 PTR Security Model

Here we present the security model for (generic) PTR schemes. We first define the adversarial game underlying this model and then use the FK-PTR scheme and explicit potential attacks against it to motivate the security definition.

PTR adversarial game. The game is parameterized by a function family F and a password dictionary Dict of size 2^d for some d (the power of two is chosen for notational convenience only). User U is initialized with password $\mathsf{pwd} \leftarrow \mathsf{Dict}$ and device D with a key k defining function F_k. Later, the parties interact so that in an undisturbed interaction between U and D, where U runs with input pwd, U outputs the secret $\mathsf{rwd} = F_k(\mathsf{pwd})$. Attacker A has oracle access to U and D, calling these parties with any message of its choice and receiving the corresponding response as defined by the scheme depending on the internal secrets and state of the responding party. The security requirements are defined below in Definition 3 but we first motivate them as follows.

Attack avenues and PTR security requirements. We define security of a PTR scheme in a way that guarantees that the generic composition of PTR and PAKE protocols results in a secure DE-PAKE scheme. The definition consists of several requirements that we motivate next via concrete attacks showing these requirements to be *necessary* (and by virtue of Thm. 3 also sufficient). Reducing the PTR requirements to the minimum necessary is pivotal for obtaining our *very* efficient FK-PTR implementation that would not be possible otherwise.

Attack avenue 1: Leakage on $\mathsf{rwd} = F_k(\mathsf{pwd})$. Given that A can obtain values in RDict by interacting with D on input any password in Dict we need to assure that nothing in the scheme leaks information on the specific value of $\mathsf{rwd} = F_k(\mathsf{pwd})$ or otherwise the attacker can use this information to gain advantage on guessing which of the RDict values is more plausible to be the correct rwd (e.g., it shouldn't be possible for A to test a possible value p as a candidate for pwd or to test a value r as a candidate for rwd). More generally, to apply PAKE we need to ensure that the view of the attacker at the end of the PTR run is independent, computationally or statistically, from rwd.

To capture this property we define the following experiment referred to as the *distinguishing test*. Define RDict as $\{F_k(p) : p \in \mathsf{Dict}\}$ where k is D's secret key. Let $\mathsf{rwd} = F_k(\mathsf{pwd})$ and choose $r \leftarrow \mathsf{RDict} \setminus \{\mathsf{rwd}\}$. A is given both rwd, r (in random order) and it needs to guess which one equals $F_k(\mathsf{pwd})$.

Attack avenue 2: Learning values in RDict. Since A can learn values in RDict by interacting with D, A can later interact with S in the PAKE protocol using these values. Thus, the best we can do is to require the PTR protocol not to leak to A more than one value in RDict for each interaction with D. We formalize this by defining a game where the attacker, at the end of its run, outputs a set of candidate values in RDict, and requiring that this set does not contain more than q_D correct values where q_D is the number of rogue activations of D by A.

Attack avenue 3: Using U *to test passwords.* Since the attacker can influence the values output by U in the PTR protocol, the possibility exists, at least in principle, that A makes U output a value $F_k(\mathsf{pwd}')$ where $\mathsf{pwd}' \in \mathsf{Dict}$ is known to A. In this case, A can observe the PAKE run of U with S and see if pwd' is the correct password. This allows A to test passwords in Dict without having to act as an active MitM in the PAKE protocol between U and S. While this attack is not possible against FK-PTR (as we will prove later), one can show PTR schemes where this attack is feasible. There are

two ways of dealing with this issue. We either show that any such "dictated password" requires a specific rogue activation of D (as in Attack 2 above) hence treating it as any other password in RDict that A may learn by interacting with D or we require that a secure PTR scheme does not allow for such attack. The latter is better as it prevents A from testing passwords without a rogue activation of U but the former can be acceptable in a protocol that allows the attack. Given that our FK-PTR protocol does not allow the attacker to use U as an oracle for testing passwords in RDict $\setminus$ {rwd}, we choose the stronger notion by adding an explicit requirement against such possibility.

To prevent this we require that U's PTR output equals $F_k(\mathsf{pwd}')$ for $\mathsf{pwd}' \in \mathsf{Dict} \setminus \{\mathsf{pwd}\}$ with at most negligible probability.

Attack avenue 4: Running U *on passwords outside* RDict. The PTR-PAKE composition presents an attack avenue not present in regular PAKE protocols: A can make U run the PAKE protocol on a password from a dictionary RDict^* different than RDict (note that this is different from attack scenario 3). To see this, consider an attack in which A impersonates D to U running the protocol with a key k' chosen by A. As a consequence, U will run the PAKE protocol with the value $F_{k'}(\mathsf{pwd})$, i.e., with a value uniformly distributed over $\mathsf{RDict}^* = \{F_{k'}(p) : p \in \mathsf{Dict}\}$ where RDict^* is known to A. This allows A to attack the PAKE protocol as follows. It impersonates S to U as if the server's state was initialized with password $F_{k'}(p)$ for $p \in \mathsf{Dict}$. If $p = \mathsf{pwd}$, A succeeds in the impersonation and learns pwd.[6] This attack is not contemplated in standard PAKE models where the user is assumed to run with a password from the specified dictionary and without adversarial choice of the password. To illustrate the dangers of such attack, imagine that the family F has a key k^* such that $F_{k^*}(\cdot)$ is a constant function (with an output known to A). This is a real possibility against FK-PTR if we define $F_k(p)$ to be $H((H'(p)^k)$ in which case $k^* = 0$ has exactly this effect. Similarly, if there is a key k^* for which F_{k^*} is a t-to-1 function, A could discard t passwords with each S-impersonation attempt against U. Again, this is possible against FK-PTR with the modified F_k where A can choose β', the response returned to U, to be in a group of small-order. (Such an implementation of FK-PTR would require to test $\beta' \in G \setminus \{1\}$.)

To prevent this attack avenue we require that *any* attack strategy by A for generating a dictionary RDict^* induces a 1-1 function. We formalize this as follows. Let c denote a set of coins for parties U, D, A in a PTR run. For any such c define $f_c(p)$ as the output from U if its password was p. We require that except for negligible probability over the choice of c, f_c is 1-1. (Note that each such c defines a dictionary $\mathsf{RDict}^* = \{f_c(p) : p \in \mathsf{Dict}\}$ of size $|\mathsf{Dict}|$.)

We are now ready to define PTR security.

Definition 3. *We say that a PTR scheme is* (q_D, q_U, T, ϵ)-secure *if for any PTR attacker* A *that runs time* T *and performs* q_D *and* q_U *rogue activations of* D *and* U*, respectively,* ϵ *is an upper bound on the values* $\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4$ *defined as follows (these* ϵ_i *are functions of* q_D, q_U, T *and they correspond to the above attack avenues):*

1. *the probability that* A *passes the distinguishing test of attack avenue 1 is at most* $1/2 + \epsilon_1$*;*
2. *the probability that* A *outputs more than* q_D *values in* RDict *following attack avenue 2 is at most* ϵ_2*;*
3. *the probability that* U *outputs* $F_k(\mathsf{pwd}')$ *for* $\mathsf{pwd}' \in \mathsf{Dict} \setminus \{\mathsf{pwd}\}$ *is at most* ϵ_3*;*
4. *the probability that* f_c*, as defined in attack avenue 4, is not 1-1 is less than* ϵ_4.

where in all four cases the probability goes over random PTR key k *and random* pwd *in* Dict.

4.3 Security of the FK-PTR Scheme

Theorem 1 below summarizes the security of the FK-PTR scheme in terms of Definition 3. It uses the One-More Gap Diffie-Hellman assumption defined next.

Note on proofs: Proofs for all statements below are in [24].

The One-More Gap DH (OMG-DH) Assumption [9, 26]: Let G be a group of prime order q and k a random value in Z_q. Let DH_k be an oracle[7] that on input $g \in G$ outputs g^k, and let DDH_k be an oracle that on input a pair (a, b) answers whether $b = a^k$. We say that G satisfies the ϵ_{omg}-OMG-DH assumption for function ϵ_{omg} if any attacker A that runs in time T has probability at most $\epsilon_{omg}(T, q_{dh}, q_{ddh})$ to win the following game: A is given access to the DH_k and DDH_k oracles, which it queries q_{dh} and q_{ddh} times, resp., and is given a set R of random elements in G. It wins if it outputs $q_{dh} + 1$ different pairs $(g, g^k), g \in R$.

Theorem 1. *Let* G *be a group where the* $\epsilon_{omg}(\cdot)$*-One-More Gap DH holds. Let the hash functions* H, H' *be modeled as random oracles and* q_H *be the number of invocations to* H*. Then, the FK-PTR scheme run over group* G *with a dictionary* $\mathsf{Dict} \subset \{0,1\}^\tau$ *is* (q_D, q_U, T, ϵ)-secure *where* $\epsilon = \max\{\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4\}$ *with* $\epsilon_1 = 0$*,* $\epsilon_2 \leq T/2^\tau + \epsilon_{omg}(T, q_D, q_H)$*,* $\epsilon_3 \leq 1/2^\tau$*,* $\epsilon_4 \leq |\mathsf{Dict}|^2/2^\tau$.

Note: The bound $|\mathsf{Dict}|^2/2^\tau$ on ϵ_4 can be reduced significantly if one relaxes requirement 4 of PTR security to allow for some deviation from injectiveness, e.g., allowing RDict to be of size $\alpha \cdot |\mathsf{Dict}|$ for some α, say $\alpha = 1/2$.

Lemma 2. *Let* G *be a group where the* ϵ_{omg}*-One-More Gap DH assumption holds and model hash functions* H, H' *as random oracles. Let* A *be a PTR-attacker against the FK-PTR scheme that runs time* T *and activates* D q_D *times with values chosen by* A *(i.e., rogue activations). Then, the probability that* A *outputs more than* q_D *values in* RDict *(as in attack avenue 2) is at most* $\epsilon_2 = T/2^\tau + \epsilon_{omg}(T', q_D, q_H)$ *where* q_H *is the number of invocations of* H *by* A *and* $T' \approx T$.

4.4 PTR-PAKE Composition Theorem

We are now ready to prove the composition theorem showing that composing a secure PTR with a PAKE that offers the wKCI-resistance property, results in a secure DE-PAKE scheme (with security definitions as presented in Section 3). As noted earlier, if the PTR and PAKE schemes dispense of PKI so does our DE-PAKE protocol: An example of such composed scheme free of PKI (except for initialization) is presented in Section 5.

Theorem 3. *Let* P *be a* $(q_D, q'_U, T_P, \epsilon_P)$*-secure PTR scheme and* Π *be a* $(q_S, q_U, T_\Pi, \epsilon_\Pi)$*-secure PAKE protocol that is also* ϵ_{KC}*-wKCI-resistant, then the DE-PAKE scheme* C *that uses the composition of both protocols is a* $(q_S, q_U, q'_U, q_D, T_C, \epsilon_C)$*-secure DE-PAKE protocol where* $\epsilon_C = \epsilon_\Pi + (3q'_U + 1)\epsilon_P + \epsilon_{KC} + \frac{q_U + q_S}{2^{\tau-1}}$.

[6] This attack recovers pwd with 2^d impersonation attempts (of S) against U and it only requires one value $F_{k'}(\mathsf{pwd})$ used by U as its PAKE password. This does not imply a break of the DE-PAKE scheme, since for each impersonation attempt against U, A needs to perform a rogue activation of U in PTR. If q'_U is the number of rogue activation of U in PTR and q_U is the number of rogue calls to U in PAKE, then the probability of successful impersonation is at most $\min\{q_U, q'_U\}/2^d$. This implies that *it is insecure for* U *to cache the value retrieved from* D *for use in multiple sessions* - doing so allows the above attack without A having to act as a MitM between U and D in each DE-PAKE session.

[7] DH_k is not defined over elements outside G hence one needs to check the input to the oracle - it can be done by an explicit group membership check or by co-factor exponentiation.

5. INSTANTIATIONS AND EXTENSIONS

In this section, we discuss several instantiations and extensions of our PTR-PAKE scheme showing the practicality and flexibility of our approach. We first present a full and detailed instantiation of PTR-PAKE that is secure in the PKI-free setting and which we use as the basis for our implementation and evaluation work reported in Section 6. Then, we show how to provide transparent DE-PAKE support to currently deployed web services, namely, armoring an existing service against online and offline attacks *without changing the server*. Finally, we comment on extensions that provide defenses against client-side and phishing attacks.

PKI-Free DE-PAKE. Figure 3 describes a full instantiation of a PKI-free PTR-PAKE protocol using the FK-PTR scheme from Figure 2 and a PKI-free PAKE protocol with resistance against wKCI attacks adapted from the threshold PAKE (TPAKE) protocol of [22, 23]. More precisely, the PAKE protocol we use is an adaptation of the variant of the TPAKE protocol from [22, 23] with resistance against wKCI attacks, proven secure in the PKI-free (or CRS) model, to the single-server case (i.e., a (1,1)-TPAKE).

The protocol as described in Figure 3 also requires a key-exchange mechanism (the "KE formula") to set a session key between server and user (in particular for the sake of mutual authentication). Different protocols can be used here, for example, based on shared keys or public keys, with or without forward secrecy, etc. However, we note that in order to achieve security against server compromise (needed to provide the maximal security of a PTR-PAKE scheme) one must use a public key mechanism. Otherwise, the server would be storing a secret authentication key for the user which would allow an attacker to impersonate the user to the server in case of server compromise. Thus, while we allow for different key exchange mechanisms through a general KE formula, we do require these to be based on public keys for both parties (we also accommodate ephemeral keys if forward secrecy is desired). For illustration, and as a concrete and efficient instantiation that preserves a minimal number of messages and provides forward secrecy, we define the key computation formulas corresponding to the HMQV protocol [27], where $e_u = H(X_u, \mathsf{S}), e_s = H(X_s, \mathsf{U})$:

$$\text{For S:} \quad K = \mathsf{KE}(p_s, x_s, P_u, X_u) = H\left((X_u P_u^{e_u})^{x_s + e_s p_s}\right)$$

$$\text{For U:} \quad K = \mathsf{KE}(p_u, x_u, P_S, X_S) = H\left((X_s P_s^{e_s})^{x_u + e_u p_u}\right)$$

The user attempts to log-in to a web service (S) from a browser running at the client (U), using her smartphone (D) to map her password pwd to a random rwd. S and U run an instantiation of PAKE protocol on input rwd to authenticate the user. Although the scheme can be extended to other devices with same capabilities as smartphones, we targeted latter as the most common device.

Server-Transparent DE-PAKE. An important implication of the modularity of our PTR-PAKE scheme is that the user can use any secure PTR protocol to derive a hardened password rwd from her nominal password pwd, and then register rwd as her actual password with an existing server, where the latter implements *any* wKCI-resistant password authentication protocol. Since the password authentication protocol is independent of the hardening procedure, it can be the standard password-over-TLS protocol used in the de-facto standard PKI setting. (Note that if the server stores a salted password hash then the password-over-TLS authentication is wKCI-resistant.) In such case the login phase of the PTR-PAKE protocol consists of the user typing her password pwd, the client terminal and the device executing the PTR protocol to compute the hardened password rwd, and the client terminal sending rwd to the server over a TLS session. In this setting, no modification to an existing service is required. We refer to this mechanism as Server-Transparent DE-PAKE.

There are several advantages of this setting: (1) the user can simply remember the short nominal pwd but register with a strong high-entropy password that significantly increases resistance to online and offline guessing attacks (in particular, offline-only attacks on a compromised server are not possible); (2) nominal password pwd can be the same or reused among multiple services, but the OPRF key associated with each service stored on the device can be different (hence also rwd would be different), and therefore the compromise of the password rwd at one server will not reveal the actual password pwd and will not compromise the user's accounts with other services; (3) rather than asking the user to frequently change the password and memorize the updated password, only the key on the device can be changed, which improves the usability.

We are currently studying the application of this server-transparent FK-PTR-PAKE scheme to building a secure password manager.

Resisting Client-Side & Phishing Attacks. Malicious code and keyloggers remain a threat to browsers in spite of browser security enhancements. Because we use a keyed password hardening scheme, an attacker who learns pwd by a key-logger or shoulder surfing can *not* authenticate to the service without interacting with the device. However, an attacker who compromises a client terminal can obtain rwd. By using service-specific keys at the device we guarantee that an attacker who obtains rwd can only compromise the particular service associated with it; even if pwd is used for multiple services, the rwd values derived for each service are random and independent .

Still, one can reduce the threat of the malware attack to the by combining our scheme with the traditional two-factor authentication (TFA) mechanism, i.e., having D generate a PRF on a time value or a nonce under a key that D shares with S. Note that in a traditional TFA mechanism, compromising the client allows the attacker to hijack the current login session of the user, but does not allow the attacker to login in future sessions (due to the use of "one-time" PIN codes). Integrating our DE-PAKE protocol with traditional TFA could provide the same level of security in the event of client compromise, while providing all the other security properties of our DE-PAKE scheme.

Resistance to phishing attacks can be achieved if rwd is computed on a concatenatation of pwd and the URL being accessed, i.e., if $\mathsf{rwd} = F_{K_d}(\mathsf{pwd}|url)$. This is similar to the PwdHash approach [28] except that in PwdHash, the attacker that obtains the randomized password through phishing can mount a dictionary attack to find the user's password while in our case this is not feasible.

6. DEVELOPMENT AND EVALUATION

We evaluate the feasibility and performance of our DE-PAKE scheme, we pursue a proof of concept system development and evaluation of protocol FK-PTR-PAKE described in Figure 3. The FK-PTR-PAKE protocol implements a secure DE-PAKE solution with full resilience to online and offline attacks, and without reliance on PKI (except for initialization).

6.1 FK-PTR-PAKE System Design

We implemented parties S, D, U as follows: S is a web-server hosted on a Debian 7.6 server running PHP 5.4.4; D is an LG G3 Verizon VS985 Android 5.1 phone running our Java code developed for Android API 22; and U logs in from a client running Google Chrome 40.0 browser on a MacBook Air laptop with a 1.3 GHz Intel Core i5 and 4GB memory. At the client side, all computations are performed in our Chrome app. At the device side,

Parties: User U, Device D, Server S.

Public Parameters and Components

- *Group* G of prime order q with generator g.
- *Hash functions* H, H' with ranges $\{0,1\}^{2\tau}, G$ and Z_q, respectively, for τ a security parameter.
- *Pseudorandom function (PRF)* f with range $\{0,1\}^{2\tau}$.
- *OPRF function* $F_k(x) = H(x, (H'(x))^k)$ for key $k \in Z_q$.
- *Key exchange formula* KE: on input long-term and ephemeral private-public keys outputs shared key $K \in \{0,1\}^\tau$.

Initialization Phase *(assumed to be executed over secure links)*

- FK-PTR Initialization: Run FK-PTR initialization of Fig. 2 to choose pwd, device's OPRF key k_d, and compute $\mathsf{rwd} = F_{k_d}(\mathsf{pwd})$.
- PAKE Initialization:
  1. S chooses $p_s \in_R Z_q$ and sends to U the public key $P_s = g^{p_s}$ (P_s can be used with all of S's users).
  2. U chooses $z \in_R \{0,1\}^\tau, k_s \in_R Z_q$;
     sets values $c = z \oplus F_{k_s}(\mathsf{rwd}),\ r = f_z(0),\ C = H(r, \mathsf{rwd}, c),\ p_u = f_z(1) \bmod q$;
     computes $P_u = g^{p_u}$ and $m_u = f_z(2, P_u, P_s)$; and sends to S the values $c,\ C,\ k_s,\ P_u,\ m_u$.
  3. S stores $c,\ C,\ k_s, P_u, m_u$ in its U-associated storage (if P_s is user-specific, it also stores P_s and p_s).

Login Phase

- **User-Device Interaction (FK-PTR)**
  Follows the FK-PTR protocol as per Fig. 2 to obtain rwd on input pwd from U and input k_d from D.
- **User-Server Interaction (PAKE)**
  1. U chooses $\rho, x_u \leftarrow Z_q$; initiates a key exchange session with S by sending its identity U, the value $\alpha = (H'(\mathsf{rwd}))^\rho$ and $X_u = g^{x_u}$.
  2. S proceeds as follows:
     (a) Checks that $\alpha \in G$;
     (b) Retrieves (c, C, k_s, P_u, m_u) from its U-associated storage;
     (c) Picks $x_s \in_R Z_q$ and computes $\beta = \alpha^{k_s},\ X_s = g^{x_s}$.
     (d) Sends to U: $\beta, c, C, P_u, m_u, P_s, X_s$.
     (e) Computes $K = \mathsf{KE}(p_s, x_s, P_u, X_u)$ and outputs session key $SK = f_K(0)$.
  3. U proceeds as follows:
     (a) Sets $z = c \oplus H(\mathsf{rwd}, \beta^{1/\rho}),\ r = f_z(0),\ p_u = f_z(1) \bmod q$.
     (b) Aborts unless the following conditions hold: $\beta \in G,\ C = H(r, \mathsf{rwd}, c),\ m_u = f_z(2, P_u, P_s)$.
     (c) Computes $K = \mathsf{KE}(p_u, x_u, P_s, X_s)$ and outputs session key $SK = f_K(0)$.
- **Explicit Authentication**
  If explicit authentication of the parties is required then S adds the value $f_K(1)$ to its message and U adds a third message with value $f_K(2)$. Each party verifies the value received from the other party.

Figure 3: Instantiation of FK-PTR-PAKE with PKI-free PAKE protocol from [22, 23]

all computation is performed by our Android app. The server uses a Java communication application to run the single-server PAKE of [22] (see Figure 3) and PHP scripts to authenticate the user.

While client U and server S communicate over the Internet, the device-client (U−D) communication depends on the features of the device. Our implementation allows D to communicate either via the cellular data network or the WiFi internet, but if such channels are unavailable, e.g. because device D looses cellular data and/or WiFi internet coverage, we allow for a fall-back to a *localized* device-client channel, which in our current implementation is realized by an *ad hoc* WiFi connection between U and D.

Hash Functions: The modular structure of the the DE-PAKE protocol gives us the flexibility to pick any hash function in the implementation. The "Hash onto Elliptic Curve" function, denoted by H', maps an arbitrary-length string into an element of the group G of prime order q; in our implementation G is an elliptic curve NIST P-256 group. In this hashing function, SHA-256 of the input (concatenated with the iteration number) is computed and truncated into an element in Z_q, the computed value is considered as the x coordinate of a point on the curve only if a y value associated with it is a quadratic residue (i.e., x and y satisfy the curve equation). Otherwise, the same computation is repeated in the next iteration until a curve element is obtained[8]. Such a point on the curve is the output of the hash function H'. The other hash function in our implementation, denoted by H, is a SHA256 hash function. The PRF f used in computing r, p_u, and m_u in Figure 3 is implemented using HMAC-SHA256 with τ-bit keys and τ-bit outputs.

OPRF Function: OPRF is defined as $F_k(x) = H(x, (H'(x))^k)$ with input x from the client and k from the server/device [22]. The OPRF works over group G of prime order p, which in our implementation is an elliptic curve NIST P-256 group.

[8]We note that the number of iterations in this hunt-and-peck procedure (as used in our prototype) can leak information on the password. A safer approach is to use a curve such as 25519 with the Elligator 2 hashing technique [11] which provides better side-channel protection and better performance.

Table 2: Performance Analysis of the FK-PTR-PAKE Protocol; *mean (standard deviation) of delay is reported in milliseconds*; The interaction of the user with the system is exuded from this evaluation. The primary $\mathsf{D}-\mathsf{U}$ channel implementation is over Cellular Data (12Mbps 4G LTE).

	Device	**Client**	**Server**	$\mathsf{S}-\mathsf{U}$ **Channel**	$\mathsf{D}-\mathsf{U}$ **Channel**	**Total**
PTR	183.30 (0.02)	45.37 (1.20)	–	–	36.81 (0.10) (4G LTE)	265.48ms
PAKE	–	195.05 (0.03)	27.85 (0.00)	26.76 (2.45)	–	249.66ms
Overall Time						515.14ms
# of EC Exponentiations	1	7	4		WiFi Internet (10 Mbps)	48.09 (0.23)
Single EC Exponentiation Cost	167.41(0.03)	10.65 (0.02)	1.39 (0.00)		Localized WiFi (54 Mbps)	7.42 (0.01)

Key Computation: The Key Computation function, KE, is from the HMQV protocol [27] described in Section 5 and is computed at the client's Chrome application and the server application.

Commitment: During initialization, commitment C is computed as $H(r, \mathsf{rwd}, c)$ at the client side, and is sent to the server to be stored at the server. During authentication, the server sends back C, which is compared against $H(r, \mathsf{rwd}, c)$ to verify that the computed values at the client are valid. The client aborts the protocol if the commitment is not verified.

Message Authentication Code: m_u is the message authentication code that is sent to the server from the client during initialization, and is stored at the server. During the authentication phase, the server sends m_u back to the client, which is compared against $f_z(2, P_u, P_s)$ at the client side. If the two do not match, it means the received m_u is not the same as the one sent to the server during the initialization and therefore the client aborts the protocol, otherwise the received public key P_s is valid (i.e., the server is legitimate) and client can continue running the protocol.

Chrome Extension: We developed a Chrome JavaScript application to implement user U on a client browser. The user needs to run this application during the initialization and authentication phases (if not already running). This application interacts with D to reconstruct rwd, and with S to construct K and session key SK, and to authenticate U to S. For all Elliptic Curve computations, we used the Stanford EC library, and for SHA-256 and OPRF functions, we used CryptoJS APIs. We used Stanford JSBN and JSBN2 libraries for all BigInteger computations on the Chrome App. We used Chrome.socket API to send and receive data between U and D in the PTR protocol, and between U and S in the PAKE protocol.

Server Application: Server side PHP scripts are the initial and last point of interaction with the user. Moreover, the application on the server-side is responsible for storing user specific information, and participating in the PAKE protocol to authenticate the user. We developed a Java program using native Java Socket, Java Security and Bouncy Castle libraries to run the PAKE protocol.

Device Application: The Android app running on the device is responsible for storing OPRF k_d, receiving $\alpha = (H'(\mathsf{pwd}))^{\rho}$ from the client, checking the group membership of α, and sending $\beta = \alpha^{k_d}$ to the client. Although this application can theoretically run on the device without any interaction with the user, we require the user to confirm the transfer of β. The user needs to launch the application (if not already running) and wait for the app to prompt for the consent. Then, she can approve the communication, to let the device send β to the client. This design choice is made to ensure that the device does not respond to unauthorized requests (without user's awareness). All elliptic curve functions in our app are based on Java Security and Spongy Castle on Android 5.1.

Device-Client Communication. Since the device-client communication in our scheme does not require to be confidential or authenticated (i.e., does not need to be protected), we can use any channel to run the protocol. Today, almost all smartphones have Internet connectivity (cellular data or WiFi), which can serve as an effective device-client communication channel. In addition to performance and bandwidth, such channels have the advantage that the user does not need to be involved in the channel set-up phase (unlike, for example, a Bluetooth channel that requires device pairing by the user). Although such channels are not cost-free (especially cellular data), currently most of the carriers offer unlimited text/voice/data plan with as less as $25 per month on some of the plans. With these type of plans, we believe that device-client channels may not impose additional cost for the user. At home or at work, for example, users often have WiFi Internet connectivity available.

Due to the above advantages our implementation is based on a device-client "data channel". On occasions this channel becomes unavailable, e.g., when the user loses network coverage, our implementation allows fall-back to the localized wireless ad-hoc channels at the cost of reduced usability, since some user effort is required for establishing localized channels. Assuming such a fall-back is not required frequently, this "hybrid model" of device-client communication may provide a high level of usability. (For other case studies using localized WiFi channel see e.g. [14, 29].)

6.2 Performance Evaluation

The overall execution time of FK-PTR-PAKE implementations is evaluated over $10,000$ iterations, and the averaged results are reported in Table 2 in settings described in the previous subsection.

The communication between U and S is timed over the Internet (web server and client with 10Mbps Internet connection), but the communication between U and D is tested in three different settings: (1) The primary setting is when the communication channel between D and the internet is a *Cellular Data* link, which in our tests was a 12Mbps Verizon 4G LTE link; (2) The second setting we tested is when D has a *WiFi Internet* connection (10Mbps), which is another popular Internet connection setting for mobile devices, commonly used as an alternative to cellular data communication; (3) Finally, we timed the protocol in the *Localized WiFi* setting, namely a point-to-point 54Mbps wireless device-client link (ad hoc WiFi) established between U and D. As we argued in the previous subsection, the localized WiFi channel is an important fall-back option in case the other two channels are unavailable. Table 2 shows overall costs for setting (1), but at the bottom of the right-most column we show the D-U channel delay in settings (2) and (3).

Based on our evaluation, for all parties, the most costly computation is Elliptic Curve (EC) exponentiation (10.65ms on the Chrome app, 167.41ms on the Android app, and 1.39ms on the server with the mentioned libraries). In one round of the protocol with different device-client links, the most costly communication is around 50ms between the client and the device over 10Mbps WiFi Internet. The overall execution time of FK-PTR-PAKE protocol, excluding the manual task of password entry, is around 515ms, which seems reasonably efficient. On an average, the PTR part of the protocol takes around 265ms to complete (with the major delay due to computation on the device), and the PAKE part of the protocol takes

249ms to complete (with the major delay due to computation on the client). Since the major computation is performed by the client due to EC exponentiations (total number of 7 Exponentiation, 2 in PTR and 5 in PAKE protocol), optimizing the exponentiation code at the client might help to further improve the performance. Also, one can reduce the number of exponentiations for both server and user by almost one exponentiation by implementing the HMQV computation via a multi-exponentiation technique.

Communication delays are a smaller component of the overall cost, hence the choice of a particular $D - U$ channel does not significanly affect the overall performance: Using WiFi Internet instead of Cellular Data adds 11.28ms, causing only a 2% increase in the overall time, while using Localized WiFi cuts 29.39ms, a 6% speed-up in the overall time.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we considered the problem of armoring password protocols against online guessing attacks as well as offline dictionary attacks in the event of server or device compromise. We proposed a novel, efficient and modular device-enhanced password protocol (DE-PAKE) and formally analyzed its security. In contrast to previous work on this subject, our protocol does not require the presence of a public key infrastructure or the availability of authenticated public keys (except, possibly, for initial password registration) thus relaxing the concerns regarding PKI failures or compromises. At the same time, when an authentic and uncompromised public key of the server is available, our protocol further guarantees resilience to server impersonation even when the user's password is disclosed. Remarkably, we can achieve these benefits without necessitating service-side changes.

Finally, we note that, thanks to our modular architecture, one can further increase the resistance to server compromise by using a threshold-PAKE protocol (e.g., [22]), in which case an attacker needs to compromise a threshold of servers *in addition to* the device before being able to mount an offline dictionary attack.

We demonstrate practical viability of our solution by building a DE-PAKE system based on a concrete instantiations of our protocol (with and without server-side changes), utilizing an automated data channel between the device and the client (falling back to localized WiFi communication only when such a channel is to become unavailable). Our performance evaluation demonstrate the promising feasibility of our schemes, although a systematic investigation of their overall usability is left for future work.

Acknowledgments

This research is partially supported by ONR Contract N00014-14-C-0113, NSF CICI Award #1547435 and #1547350, and NSF CNS Award #1209280.

8. REFERENCES

[1] Anonymous hackers claim to leak 28,000 PayPal passwords on global protest day. Available at: http://goo.gl/oPv2h.

[2] Blizzard servers hacked; emails, hashed passwords stolen. Available at: http://goo.gl/OTNWJC.

[3] Hackers compromised nearly 5M Gmail passwords. Available at: http://goo.gl/IRu07u.

[4] LinkedIn Confirms Account Passwords Hacked. Available at: http://goo.gl/UBWuY0.

[5] RSA breach leaks data for hacking securid tokens. Available at: http://goo.gl/tcEoS.

[6] RSA SecurID software token cloning: a new how-to. Available at: http://goo.gl/qkSFY.

[7] Russian Hackers Amass Over a Billion Internet Passwords. Available at: http://goo.gl/aXzqj8.

[8] T. Acar, M. Belenkiy, and A. Kupcu. Single password authentication. *Computer Networks*, 57(13):2597 – 2614, 2013.

[9] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. 16(3):185–215, June 2003.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – Eurocrypt*, 2000.

[11] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. pages 967–980, 2013.

[12] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, pages 30–45, London, UK, UK, 1997. Springer-Verlag.

[13] X. Boyen. Hidden credential retrieval from a reusable password. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 228–238, New York, NY, USA, 2009. ACM.

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

[15] I. Dacosta, M. Ahamad, and P. Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In *European Symposium on Research in Computer Security*, 2012.

[16] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia prf service. In *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[17] W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2000.

[18] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography*. 2005.

[19] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology-CRYPTO*. 2006.

[20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating ssl certificates in non-browser software. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[21] D. P. Jablon. Password authentication using multiple servers. In *The Cryptographer's Track at RSA Conference (CT-RSA)*. 2001.

[22] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology–ASIACRYPT*. 2014.

[23] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly Efficient and Composable Password-Protected Secret Sharing. In *1st IEEE European Symposium on Security and Privacy (EuroS&P)*. 2015.

[24] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena. Device-enhanced password protocols with optimal online-offline protection. IACR Cryptology ePrint Archive: Report 2015/1099 available at http://eprint.iacr.org/2015/1099, December 2015.

[25] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Theory of Cryptography*. 2009.

[26] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks*. 2010.

[27] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Advances in Cryptology–CRYPTO*, 2005.

[28] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *Usenix security Symposium*, 2005.

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

[30] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *USENIX Security Symposium*, 2009.