Tunneled TLS for Multi-Factor Authentication

Darko Kirovski and Christopher A. Meek
Microsoft Research, Redmond, WA, USA

Contact: darkok@microsoft.com

Technical Report MSR-TR-2009-50
April 2009

Microsoft Research
One Microsoft Way Redmond, WA 98052, USA
http://research.microsoft.com

Tunneled TLS for Multi-Factor Authentication

Darko Kirovski and Christopher A. Meek
Microsoft Research, One Microsoft Way, Redmond, WA 98052

***Abstract*—When logging onto a remote server from a distrusted terminal, one can leak secrets such as passwords and account data to various forms of malware. To address this problem, we take an existing approach of using a trusted personal device as the interface available to users for entering their login credentials. In our proposal, such a device would send the credentials to a server using a tunneled TLS session routed via a distrusted terminal. The tunneling would be done within an existing TLS session established between the terminal's browser and the server. Upon validating the credentials, the server would enable the terminal to access the user account. Consequently, the terminal would never see in plain-text the login credentials. We show that the proposed protocol resists arbitrary key-loggers, phishing agents, cross-site scripting, and invasive virtual machines. As a powerful and surprising application, if the distrusted terminal is at a point-of-sale, the trusted device could use our protocol to execute a payment with it. The user experience for this payment engine is similar to a payment with a traditional credit card.**

I. Introduction

According to a market research report, a surprising 9% of US online consumers have experienced identity theft [1]. Those who fall prey are not online novices: 52% purchased goods online in the past three months; they have an above average online tenure of 5.6 years; and 69% are technology optimists [1]. Another report estimates that roughly half of the US consumers online are *extremely* concerned (5, on a 1-5 scale) about computer malware, online identity theft, and interception of personal data; as a consequence more than 75% of all US consumers chose to deploy free, 24% paid, and 1% no software for malware protection [2]. This does not resolve fear as, for example, roughly one third of US online consumers chose not to use online banking from fear of identity theft; security as a top competitive differentiator is used by 43% of banks to attract the remainder of the online population [3]. The consequences are:

- **direct** – identity theft results in billions of dollars worth of damage to the world economy [4], and
- **indirect** – the threat of identity theft limits consumers' engagement in the online economy, thus, hurting the IT industry as well as numerous businesses that have found a more efficient way to conduct business by going online, e.g., banking, insurance, financial services, etc.

Phishing and key-loggers are simple and popular identity theft methods. Most tools that aim at addressing this problem by detecting such malware, do suffer from being detected themselves and then, circumvented. In addition, some of the available software for protection from malware, is malware itself. The core of the problem lies in the ease of collapsing the circle of trust between the user, the remote server that offers service of interest to the user at cost, and the IT infrastructure that connects them. In this paper, we look at multi-factor authentication as one way to address this problem and opportunistically seek for novel applications – mostly aiming at points-of-sale.

A. The Problem

Informally, the problem statement is simple. A remote server hosting a user account wants to authenticate that a client who is logging under the user's name, is indeed controlled exclusively by the user herself/himself. It is rather difficult to formalize this simple problem description within the existing computing milieux.

The research community has tried to resolve the problem of user authentication from a fully distrusted computer using a broad set of methodologies based upon *multi-factor authentication* (MFA). MFA[1] approaches rely upon at least two of the following three authentication paradigms:

- **"what I know"** – the most commonly deployed mechanism, a password, suffers from the following issues: it is easy to intercept, thus, opens doors to a range of attacks from key-logging to shoulder-surfing, and even modest entropy implies a memorization challenge, i.e., passwords are prone to directory attacks [5].
- **"what I have"** – the user proves to the remote server during authentication that he/she is in possession of a trusted device, i.e., its secrets. This method is used predominantly as an amplifier of trust during authentication. Apart from inadequate security protocols, the only problem related to this authentication paradigm, is the fact that it cannot be used exclusively, i.e., anyone who has access to the device, could impersonate the users whose secrets are stored on it.
- **"what I am"** – scan of user's biometric traits is used as a discriminator during authentication [6]. This method is plagued by many issues: human assistance during verification is necessary because it is rather easy to fool the measurement equipment [7], poor practical error rates [6], cost, privacy, and inconvenience at use.

Considering the strengths of individual authentication paradigms, most proposals rely upon passwords and authentication hardware. The body of research work that introduces protocols that leverage upon these two approaches is fairly diverse and we review it in Section V. We recognize several crucial objectives:

(*i*) **secure login** – the initiative for migrating the input of user credentials from the *distrusted terminal*, $\mathbf{c}$, onto a *personal trusted device*, $\mathbf{p}$, with an objective that $\mathbf{c}$ never observes these credentials in plain-text, [8] and

[1] A solid informal survey of standard MFA techniques can be found on Wikipedia: http://en.wikipedia.org/wiki/Two-factor_authentication.

(*ii*) **write confirmation** – faced with the threat of transaction generators (TG) [9] and cross-site scripting (XSS) attacks [10], the *server*, **s**, would seek a confirmation for each "write" to the user's account state via **p** [11]. Instead of relying on stolen credentials, a TG simply waits for the user to login to her account and then issue transactions on behalf of the user [9]. Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. In recent years XSS surpassed buffer overflows to become the most common of all publicly reported security vulnerabilities [12].

(*iii*) **secure logoff** – a periodic ping from the trusted device [13] as well as an explicit logoff signal issued by the user from **p** are two system actions that can ensure **s** about the proper lifetime of a secure session. Since XSS attacks rely on the existence of established secure sessions at victim's computer, enabling secure logoff is as important as enabling the secure login.

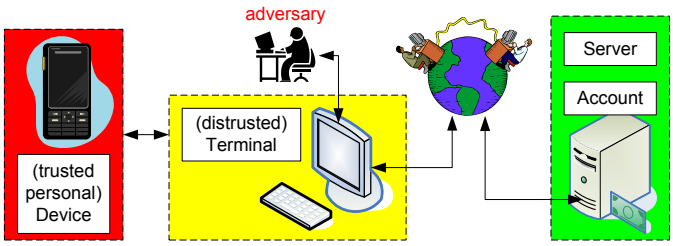

Fig. 1. Parties of interest in multi-factor authentication: a server **s** hosting a user account, a terminal **c** assumed to be fully under the control of the adversary, and a trusted device **p** in the hands of the account owner.

B. The Contribution

We address (*i–iii*) by proposing an authentication system based upon a simple tunneled TLS[2] connection from **p** to **s**. The simplicity of the proposed protocol as well as the fact that it addresses effectively several challenging and important problems in authentication, represent the key contribution of this work.

In our protocol, first **c** authenticates **s** using a standard PKI-based TLS connection [14]. Next, **c** connects to **p** using a local wired or wireless near-field communication channel such as BlueTooth. We *do not* require that this connection is authenticated and/or private, hence cumbersome trusted pairing protocols are avoided. Then, **s** initiates a separate TLS connection tunneled via **c** into **p** in order to achieve the MFA objective: obtain a password, a device identifier, and/or a biometric template from the user. As a consequence, **c** never observes in plain-text the authentication credentials. Upon validating the credentials, **s** grants **c** access to the user account so that all "writes" to the account state (i.e., stock purchase, money transfer, sending an email, etc.) are confirmed by a user action executed on **p**. Finally, **p** periodically (e.g., say every 30 seconds [13]) sends a "ping" signal to **s** over the tunneled TLS to assure of the continuous proximity between **c** and **p**. The user can logoff by either walking out on **c** at which point the proximity between **c** and **p** is broken or by specifically sending a "logoff" signal from **p** to **s**, again, via the tunneled TLS.

The connectivity of the parties of interest in our protocol is illustrated using Figure 1. Section II reviews the construction and properties of the proposed protocol in detail.

We assume a broad threat model where among other assumptions we consider:

- A virtual machine (VM) fully controlled by the adversary, is executing all operations on **c**, i.e., the VM can manipulate the data and control flows of client's browser or any other resource at will.
- A "ghost" user-interface is available to the malicious party who is in control of **c**. The adversary can render an instance of an on-going `https://` session on a "ghost" display – a display hidden from the user, possibly being rendered on another computer, and under the full control of the malicious party.

Within our objectives and considering the above security assumptions further detailed in Section III, we formally verified the proposed protocol using a publicly available, off-the-shelf package, AVISPA [15]. In a brute-force simulation of considered adversarial scenarios, the verifier did not find a single vulnerability. Section IV details this effort as the main experimental result.

C. Preliminary Discussion

Server change. A server must be cognizant of the fact that a specific secure session is launched from a terminal which the user does not trust; hence, the server in its own as well as in the interest of the user, should restrict display of identifying data on such a terminal (e.g., account number, birthdate) as well as ask for a confirmation for each crucial "write" to user's account. Although most related work has pursued authentication systems which *do not* pose demand for server-side alterations (see Section V), we stress that this objective is flawed as servers must address the fact that all data sent to and from the browser that resides in **c**, is assumed public. To that extent, we anticipate that reliable MFA would require changes to **s** and thus, propose a protocol that also asks for additional, minor steps from **s**.

Choice of trusted device. We acknowledge that the design of the trusted personal device, **p**, could be realized so to reach different security vs. cost configurations. The top end would be represented by an intrusion-proof tamper-resistant dedicated personal device with a full user I/O interface (e.g., keyboard and display similar to a mobile phone) and a Web browser with minimal HTML parsing/rendering capabilities (e.g., scripting and dynamic content disabled). One of the least expensive

[2]We refer to the Transport Layer Security protocol and its predecessor, the Secure Sockets Layer, as means of authentication via a public key infrastructure.

Step	Server $\mathbf{s}$	Terminal $\mathbf{c}$	Device $\mathbf{p}$	User
I	$K_{\mathbf{sc}} = TLS(K_{\mathbf{s}}^{-1}, \mathrm{cert}(\mathbf{s}, K_{\mathbf{s}}))$ all further communication between $\mathbf{s} \leftrightarrow \mathbf{c}$ is tunneled via this TLS channel	$K_{\mathbf{sc}} = TLS(K_{\mathbf{t}})$		visits https://www.s.com from $\mathbf{c}$
II	$m_1 = E_{K_{\mathbf{sc}}}(\mathrm{html}) \rightarrow \mathbf{c}$ html = login page with "login via $\mathbf{p}$" option	$\mathrm{html} = D_{K_{\mathbf{sc}}}(\mathrm{m}_1)$ display html		checks "login via $\mathbf{p}$" – installs device discovery app at $\mathbf{c}$
III		pairs with $\mathbf{p}$	pairs with $\mathbf{c}$	controls pairing UI at $\mathbf{p}$ and $\mathbf{c}$
IV	$K_{\mathbf{sp}} = TLS(K_{\mathbf{s}}^{-1}, \mathrm{cert}(\mathbf{s}, K_{\mathbf{s}}))$ all TLS messaging between $\mathbf{s} \leftrightarrow \mathbf{p}$ is passed through the $\mathbf{s} \leftrightarrow \mathbf{c}$ tunnel on its way to/from $\mathbf{p}$. all further messaging between $\mathbf{s} \leftrightarrow \mathbf{p}$ is tunneled via the $\mathbf{s} \leftrightarrow \mathbf{p}$ TLS channel. the $\mathbf{s} \leftrightarrow \mathbf{p}$ TLS channel is always tunneled via the $\mathbf{s} \leftrightarrow \mathbf{c}$ TLS channel on its way from/to $\mathbf{c}$ to/from $\mathbf{s}$.		$K_{\mathbf{sp}} = TLS(K_{\mathbf{t}})$	visits https://www.s.com from $\mathbf{p}$
skip V				
VI	$m_2 = E_{K_{\mathbf{sp}}}(\{\mathrm{html}, x\}) \rightarrow \mathbf{p}$ html = form to enter username, pwd x = challenge $r = D_{K_{\mathbf{sp}}}(\mathrm{m}_3)$ `if` $f^{-1}(r) = \{x, h(\mathrm{password}), \mathrm{ID})\}$ `then` $m_4 = E_{K_{\mathbf{sc}}}(\mathrm{html}) \rightarrow \mathbf{c}$ html = page with account info	$\mathrm{html} = D_{K_{\mathbf{sc}}}(\mathrm{m}_4)$ display html	$\{\mathrm{html}, x\} = D_{K_{\mathbf{sp}}}(\mathrm{m}_2)$ display html $r = f(x, h(\mathrm{pwd}), \mathrm{ID})$ $m_3 = E_{K_{\mathbf{sp}}}(r) \rightarrow \mathbf{s}$	visits https://www.s.com from $\mathbf{p}$, enters credentials pwd = password accesses account at $\mathbf{s}$ from $\mathbf{c}$

TABLE I

DESCRIPTION OF THE TUNNELED TLS PROTOCOL. STEP V WAS INTENTIONALLY SKIPPED. FUNCTIONS f AND f^{-1} REPRESENT RESPONSE AND VERIFICATION FUNCTIONS TO A CHALLENGE x RESPECTIVELY. FUNCTIONS $D_K()$ AND $E_K()$ DECRYPT AND ENCRYPT RESPECTIVELY USING A SYMMETRIC KEY K. A PUBLIC-PRIVATE KEY-PAIR IS REPRESENTED AS K_x AND K_x^{-1} RESPECTIVELY. FUNCTION $TLS()$ EXECUTES ALL STEPS OF THE TLS PROTOCOL WITH SERVER-AUTH ONLY. FUNCTION $\mathrm{cert}(\mathbf{s}, K_{\mathbf{s}})$ REPRESENTS A CERTIFICATE FOR $\mathbf{s}$'S PUBLIC KEY ISSUED BY A TRUSTED AUTHORITY WITH A PUBLIC KEY $K_{\mathbf{t}}$. FINALLY, ID REPRESENTS THE SECRET DEVICE ID UNIQUE TO $\mathbf{p}$.

solutions is a full software implementation on a smartphone[3] with Bluetooth. We consider the last solution rather appealing and in Section II-D pay special attention to an implementation of our protocol on such a device.

Application. The research community has been enamored with authentication solutions that particularly apply to Internet cafés (ICs) (see Section V). Such an objective could not be more restricting considering the fact that there exist roughly 3-4 million ICs worldwide compared to tens of millions of points-of-sale that literally drive the entire world economy. Later, in Section IV-C, we emphasize the benefits and drawbacks of our proposal with respect to a list of applications.

II. TUNNELED TLS

In this section, we provide the objectives and details of the proposed authentication protocol.

A. Preliminaries

ENTITIES. We consider the following entities:

- $\mathbf{s}$ denotes a *server* which manages a database of user accounts. It uses a standard certificate-based public key infrastructure to authenticate itself to an arbitrary client.
- $\mathbf{c}$ denotes a *client* computing platform (personal computer, terminal, or even virtual machine) which is not trusted by its users. The assumption is that the full user interface (e.g., keyboard, mouse, display) could be intercepted, altered, and stored by a malicious party who has unlimited access to $\mathbf{c}$ (e.g., via a Trojan virus or otherwise).
- $\mathbf{p}$ denotes a *personal computing device* such as a mobile phone, trusted by the user. This device can establish a connection to any $\mathbf{c}$ via a common public wireless or wired communication protocol such as Bluetooth or USB. We assume that the incoming networking stack into $\mathbf{p}$ is built to be robust to intrusion attacks. Hence, we assume that the existence of a relevant data-sniffing mechanism on $\mathbf{p}$ is unlikely. Section II-D discusses these assumptions in detail.

[3]See Wikipedia entry, http://en.wikipedia.org/wiki/Smartphone, for informal definition.

DEVICE PAIRING. A wireless connection between $\mathbf{c}$ and $\mathbf{p}$ is of particular importance due to its convenience. We assume that the pairing protocol between $\mathbf{c}$ and $\mathbf{p}$ which initiates their wireless communication, does not guarantee to $\mathbf{p}$ that it has connected to a specific physical unit $\mathbf{c}$. We also assume that $\mathbf{c}$ is previously unknown to $\mathbf{p}$, i.e., they have never had a mutual connection. The fact that security is not imposed during the pairing process simplifies this step as cumbersome trusted pairing protocols are not required.

OBJECTIVE. A user wants to log onto her account at $\mathbf{s}$ so that:

a) access is granted from/to a specific physical terminal $\mathbf{c}$,
b) $\mathbf{s}$ authenticates the user by using the **"what I have?"** (device $\mathbf{p}$) and **"what I know?"** (account password) authentication paradigms,
c) the credentials that answer the authentication questions are never communicated in plaintext via $\mathbf{c}$ – clearly, they are entered using $\mathbf{p}$,
d) a session between $\mathbf{s}$ and $\mathbf{c}$ is alive as long as the user continuously demonstrates that she can answer **"what I have?"** to $\mathbf{s}$, and
e) all critical actions with the account are verified by an action from $\mathbf{p}$.

Since we assume that the system I/O at $\mathbf{c}$ could be intercepted, recorded and/or modified by a malicious party, the best that the account user could hope for is that the adversary:

i) cannot gain access to login credentials, thus cannot access the account at a later time,
ii) cannot initiate or interfere with critical actions during

a $\mathbf{s} \bowtie \mathbf{c}$ session. Note that editing the system I/O in real-time, i.e., terminal screen, is an attack possibility, although difficult – that should not have serious consequences as all critical actions (cash transfers, payments, stock transactions) with the account are verified via $\mathbf{p}$.

A question arises: *if* $\mathbf{p}$ *is a trusted platform, why not access the account entirely from* $\mathbf{p}$*?* Mobile computing devices commonly do not have user interface options that enable comfortable data display and entry – thus, combining the convenience of the desktop user interface with the trust associated with personal computing devices, makes this and similar proposals appealing, secure, and ubiquitous. We review several design choices for $\mathbf{p}$ in Subsection II-D.

B. The Protocol

Let's assume the scenario of logging onto a banking account A using a public computer, $\mathbf{c}$, located at a university computing center. The owner of the account also has a mobile phone, $\mathbf{p}$, at her disposal. By following a standard login procedure, login credentials for A would be exposed to a key-logger installed on $\mathbf{c}$. Here is the proposed remedy – its steps are illustrated in Table I-B.

Step I – TLS handshake $\mathbf{s} \bowtie \mathbf{c}$ w/ server-only auth. Using $\mathbf{c}$'s Web browser[4], the user would establish a TLS[5] session (`https://...`) with the provider of the service A, $\mathbf{s}$. Here, only $\mathbf{c}$ would verify the authenticity of $\mathbf{s}$ using standard public-key cryptography, not the other way around. An established TLS connection assumes a successful symmetric key exchange. The exchanged master secret, $K_{\mathbf{sc}}$, is used for encrypted communication between $\mathbf{s}$ and $\mathbf{c}$. Details of TLS can be found in [16] with a formal verification in [17].

Step II – Login choice. Then, $\mathbf{s}$ would provide a Web page to $\mathbf{c}$ with a log-in fill-out form (`username:` and `password:`) as well as a user-interface to select the type of login: "Web-only" or "via $\mathbf{p}$" that the user can check/select to log via a Bluetooth enabled device. An example is presented in Figure 2(left).

Step III – $\mathbf{c} \bowtie \mathbf{p}$ device discovery. In the latter case (which is of interest here), a small application installed on $\mathbf{c}$ (by $\mathbf{s}$ or some other not necessarily trusted source, such as an OS manufacturer, credit card company, or phone manufacturer) would seek for neighboring Bluetooth devices. The user would then select $\mathbf{c}$ and $\mathbf{p}$ as pairing choices on $\mathbf{p}$ and $\mathbf{c}$ respectively. An example of an interface that would enable the selection of a specific $\mathbf{p}$ on $\mathbf{c}$ is presented in Figure 2(left).

Bluetooth does not offer any cryptographic guarantees that two specific units are connected directly. Fortunately, in our setup such a guarantee is not required. From $\mathbf{c}$'s perspective, the "correct" identity of $\mathbf{p}$ is not important because $\mathbf{c}$ only serves as a user interface to the account that $\mathbf{p}$'s credentials can access. From $\mathbf{p}$'s perspective, connecting to the "correct" $\mathbf{c}$ is important because of usability issues, but irrelevant for security as we already assume in our threat model that the adversary has full access to $\mathbf{c}$. We address the convenience of a "correct" connection to a specific $\mathbf{c}$ in step **V** of our protocol.

Step IV – Tunneled TLS handshake $\mathbf{s} \bowtie \mathbf{p}$ w/ server-only auth. Once $\mathbf{c}$ and $\mathbf{p}$ are connected, $\mathbf{p}$ would use the $\mathbf{c} \bowtie \mathbf{p}$ communication channel to initiate a new TLS handshake tunneled[6] via the existing $\mathbf{s} \bowtie \mathbf{c}$ TLS session. In other words, all the TLS messaging between $\mathbf{s}$ and $\mathbf{p}$ would be encrypted/decrypted using $K_{\mathbf{sc}}$ as $\mathbf{c}$ relays/receives it to/from $\mathbf{s}$. By using tunneling, we ensure that all attacks on our protocol must involve full control over $\mathbf{c}$. We assume that $\mathbf{p}$ holds a "correct" public key of the trusted authority who signed $\mathbf{s}$'s certificate used in the TLS handshake.

Similar to step **I**, $\mathbf{p}$ would authenticate $\mathbf{s}$ but not otherwise. The result of the TLS handshake, a session master secret, $K_{\mathbf{sp}}$, would be used for encrypted communication between $\mathbf{s}$ and $\mathbf{p}$. Similar to the $\mathbf{s} \bowtie \mathbf{p}$ handshake, all further communication between $\mathbf{s}$ and $\mathbf{p}$ already encrypted with $K_{\mathbf{sp}}$ would be additionally encrypted with $K_{\mathbf{sc}}$.

Step V – Verifying $\mathbf{c} \bowtie \mathbf{p}$. Before accessing A, $\mathbf{p}$ wants to ensure that it is connected to the "correct" $\mathbf{c}$. This is a mere convenience feature as the user does not want to connect accidentally to "incorrect" terminals in the computer center. Since all data displayed on $\mathbf{c}$ is considered public, at this point the user understands that her actions with accessing the account could be seen by others. It is the responsibility of $\mathbf{s}$ to anonymize the presented account data in such a way that a malicious observer cannot figure out the identity of $\mathbf{p}$'s owner.

The key idea behind assuring the owner of $\mathbf{p}$ that her device is connected to the "correct" $\mathbf{c}$, is in showing that some secret data or user action is shared on the user interfaces of both $\mathbf{c}$ and $\mathbf{p}$. This could be achieved using several mechanisms, e.g.:

V.1 $\mathbf{p}$ would generate a large random number r, send it to $\mathbf{s}$ and display it on $\mathbf{p}$. Then, $\mathbf{s}$ could display r on $\mathbf{c}$ and ask the user to confirm that $\mathbf{c}$ and $\mathbf{p}$ both show r. The confirmation must be made on $\mathbf{p}$'s user interface.

Step VI – Login. Now, $\mathbf{s}$ sends a Web page to $\mathbf{p}$, asking the user to provide the login credentials: the username and the password. After entering the data, $\mathbf{p}$ sends out the information tagged with its identifier $id(\mathbf{p})$, e.g., a random long number that $\mathbf{s}$ associates with A. This number is stored with $\mathbf{p}$ and $\mathbf{s}$ when the account is open and/or $\mathbf{p}$ registered with $\mathbf{s}$ by the user. The purpose of $id(\mathbf{p})$ is to impose a limitation that A is accessed only by devices approved by the user and $\mathbf{s}$; they could associate several devices with A. The $\mathbf{s} \leftrightarrow id(\mathbf{p}) \leftrightarrow \mathbf{p}$ association procedure is detailed later in this section.

Login credentials (username, password, and $id(\mathbf{p})$) could be sent over the encrypted $\mathbf{p} \bowtie \mathbf{s}$ TLS channel using an arbitrary password authentication protocol including weak ones such as a hash of the plain-text password. In the formal verification of the proposed protocol we used a variant of the Secure Remote Password (SRP) protocol, SRP-6, detailed in [18], [19]. Another elegant way to implement the verification of credentials is via a challenge/response scheme, where $\mathbf{s}$ would send a challenge random number x to $\mathbf{p}$, $\mathbf{p}$ would compute a

[4]In the remainder of this article, when we refer to $\mathbf{c}$, we mean $\mathbf{c}$'s Web browser unless otherwise stated.

[5]Details of the Transport Layer Security (TLS) protocol and its predecessor, Secure Sockets Layer (SSL) protocol can be found at: http://en.wikipedia.org/wiki/Secure_Sockets_Layer.

[6]See Wikipedia entry: http://en.wikipedia.org/wiki/Tunneling_protocol for an informal description of tunneled protocols.

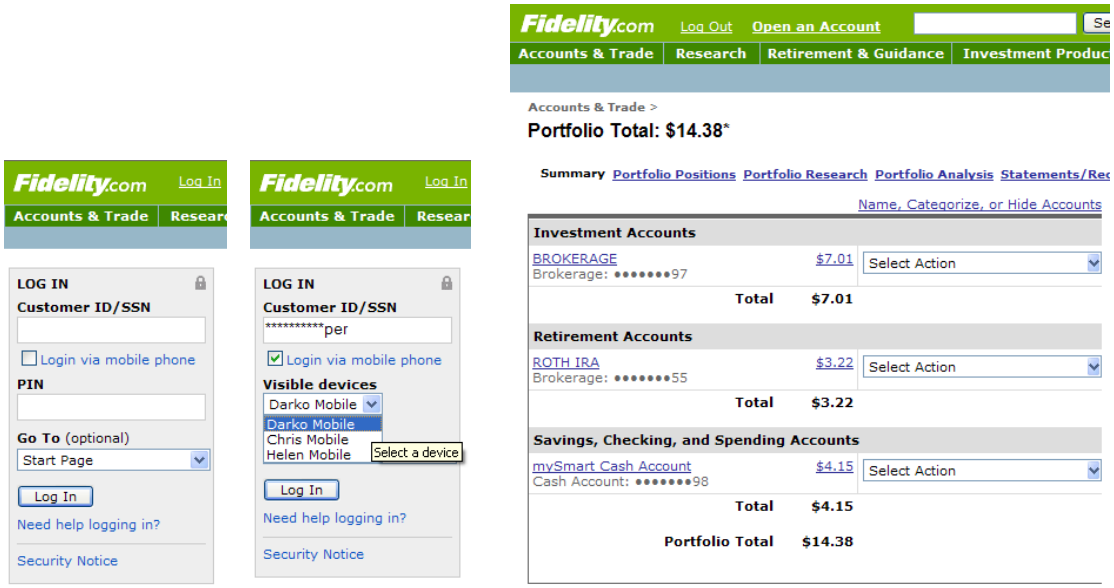

Fig. 2. (left) An example of a user interface offered by the server to authenticate a user using a locally accessible, personal, and trusted mobile device. (right) An example of how the server upon authentication marks account numbers to prevent data sniffing.

response $r = f(id(\mathbf{p}), x, h(\text{password}))$, where x represents a random nonce, $h()$ represents a cryptographic hash function such as SHA-256 that operates over the concatenation of its arguments, and $f()$ is a response function such as $h()$ or a public-key signing function such as RSA with $id(\mathbf{p})$ being used as the private key.

Once $\mathbf{s}$ receives r, since it has access to x, $h(\text{password})$ and $id(\mathbf{p})$ (in case $f = h$) or $id^{-1}(\mathbf{p})$ (in case f is a public-key signature), i.e., the public key that corresponds to $id(\mathbf{p})$, it can validate the authenticity of the credentials by verifying against r.

If $\mathbf{s}$ positively authenticates the information sent from $\mathbf{p}$, it would allow $\mathbf{c}$ access to A. As a precautionary action, $\mathbf{s}$ should present only limited content about A on $\mathbf{c}$ for this logon case specifically. The primary objective is to prevent a potential data sniffing agent at $\mathbf{c}$ to obtain crucial information about A such as the account number and the username. An example of a Web page that avoids presenting crucial account information is presented in Figure 2(right).

Finally, it is important to stress that all data sent between $\mathbf{s}$ and $\mathbf{p}$ is encrypted with $K_{\mathbf{sp}}$. That way, $\mathbf{c}$ cannot fool $\mathbf{p}$ into revealing its credentials to a third party or $\mathbf{c}$. Only a party who owns the private key correspondent to $\mathbf{s}$'s certificate will be able to read this cipher-text. This cipher-text is additionally encrypted with $K_{\mathbf{sc}}$ on its way from $\mathbf{c}$ to $\mathbf{s}$.

Post-auth step VII.a – Write confirmation. Once $\mathbf{c}$ starts accessing information about A at $\mathbf{s}$, under the assumption that the entire I/O to $\mathbf{c}$ is controlled by the adversary, there exists a demand to verify all critical "data writes" (i.e., cash transfers, stock transactions, etc.) into A initiated from $\mathbf{c}$. This is done by asking for explicit approval from $\mathbf{p}$ for each critical "write" into A initiated by $\mathbf{c}$.

Post-auth step VII.b – Read validation. In addition, as we assume that $\mathbf{c}$ is in control of a "ghost" display, it could fool the owner of $\mathbf{p}$ into an action by presenting false information on its terminal, i.e., false stock price or missing credit card transaction. To that extent we allow the user to ask for specific (parts of) Web pages presented on $\mathbf{c}$ to be forwarded by $\mathbf{s}$ to $\mathbf{p}$ for content verification. Such requests could be handled by a simple AJAX script[7] augmented into a Web page served by $\mathbf{s}$ onto $\mathbf{c}$.

Post-auth step VII.c – Continuous "What I have?". Finally, in order to prevent the adversary from browsing data on A when $\mathbf{p}$ loses power or connection to $\mathbf{c}$, $\mathbf{s}$ would continuously ping $\mathbf{p}$ to resume the session with $\mathbf{c}$. This way, $\mathbf{p}$ can logout from a current session by ceasing to respond to pings from $\mathbf{s}$. One simple implementation of a ping could be a challenge to $\mathbf{p}$ to encrypt a counter using $K_{\mathbf{sp}}$. The counter would increment after each challenge and start from a large random value set by $\mathbf{s}$ during the initial TLS handshake in step **IV**. An alternative method for logging off is a specific "log-off" signal sent to $\mathbf{s}$ by $\mathbf{p}$ – this approach achieves an instant termination of the $\mathbf{s} \bowtie \mathbf{c}$ TLS connection.

The described protocol **I-VI+VII.a-c** is specifically constructed to address the objectives presented in Subsection II-A. Key steps of the protocol (all but step **V**) are illustrated in Figure 3. Note that although $\mathbf{c}$ and $\mathbf{p}$ communicate over a "non-secure" BlueTooth connection (E0 used in Bluetooth has been proven vulnerable to several attacks [20]), the platform still achieves its objective. The protocol proposed in this section could be used for numerous other applications such as a universal payment method without any e-cash stored on $\mathbf{p}$, coupons, etc. Other commonly used devices such as PocketPCs, mobile music players, etc., could be used to provide these functionalities.

[7]See Wikipedia entry for Asynchronous JavaScript and XML at: http://en.wikipedia.org/wiki/AJAX.

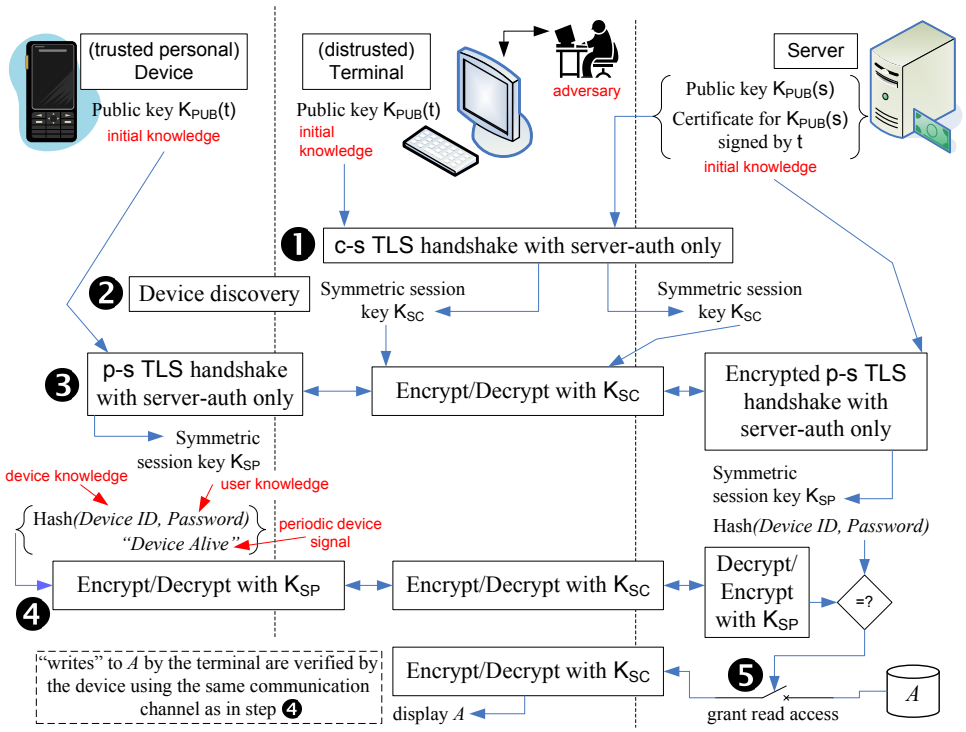

Fig. 3. Block diagram of protocol steps: $\mathbf{c}$ and $\mathbf{s}$ start with a TLS handshake, then $\mathbf{c}$ and $\mathbf{p}$ discover each other, next $\mathbf{p}$ starts a TLS handshake with $\mathbf{s}$ that gets tunneled via $\mathbf{c} \bowtie \mathbf{s}$, finally, $\mathbf{p}$ sends the login credentials so that only $\mathbf{s}$ can read them. Post-authentication signals among the protocol entities include: "device alive", i.e., a continuous "what I have," signal that assures $\mathbf{s}$ that $\mathbf{p}$ is still connected to $\mathbf{c}$, and "A $\leftarrow$ data" command sent by $\mathbf{c}$ to $\mathbf{s}$ via $\mathbf{c} \bowtie \mathbf{s}$ which initiates a "write confirmation" signal from $\mathbf{s}$ to $\mathbf{p}$ via $\mathbf{p} \bowtie \mathbf{s}$; $\mathbf{s}$ would execute "A $\leftarrow$ data" only if $\mathbf{p}$ responds affirmatively.

C. Enrollment and Revocation

It is important to describe how $\mathbf{s}$ enrolls $\mathbf{p}$ to access A. This can be done in several ways: (*i*) when the user accesses A from a trusted computer, then $\mathbf{s}$ could either generate $id(\mathbf{p})$ and send it to $\mathbf{p}$ for permanent storage or $\mathbf{p}$ could generate $id(\mathbf{p})$, store it locally, and register it with $\mathbf{s}$; $id()$ could also be a permanent number hardwired to each device, (*ii*) $\mathbf{p}$ could disclose/receive its $id(\mathbf{p})$ during an SMS or other type of data communication with $\mathbf{s}$ where the user would authenticate herself to $\mathbf{s}$ by demonstrating knowledge of A.

With certain risks, $\mathbf{p}$ could be enrolled from a distrusted terminal. Then, $\mathbf{s}$ would accept a login via $\mathbf{p}$ that is based upon the username and its password only (i.e., a regular login) to get to a point where registration of $id(\mathbf{p})$ is performed. This is an action that should be avoided as it can be done from any $\mathbf{p}$. For example, a shoulder-surfed password would turn any device into a valid $\mathbf{p}$.

Device revocation via an 1-800 phone call or other form of direct connection to $\mathbf{s}$, would entail removing $id(\mathbf{p})$ from the list of devices allowed to access A. The user would be given an opportunity to register a new device using a trusted computer or a timed (say during a phone call to a service center) device registration via a distrusted computer.

D. Design Choices for $\mathbf{p}$

The proposed functionality can be programmed into the mobile phone or onto a smart card (SIM card) used to transfer account features from one mobile phone to another. Compared to general purpose computing platforms, it is arguable whether mobile devices offer better or worse security with respect to viral malware. We review certain design parameters for mobile devices that could be advantageous.

- All or most of the software on a mobile phone is controlled by the service provider. This feature typically redirects the responsibility for virus protection from the owner of the device to its service provider.
- On most mobile phones the real-time hardware stack that handles the voice communication is isolated from the general-purpose programmable unit. The latter hosts the general-purpose OS and resident applications. These "hidden" computing resources offer an inexpensive alternative to smart cards for incorporating security features that adversaries cannot access with malware installed on the programmable unit.
- It is relatively easy to implement a non-programmable, and tamper-resistant operational mode on a mobile phone.
- The system design of a mobile phone is self-contained and allows deployment of more radical intrusion management platforms such as randomized instruction sets [21] or intrusion prevention via code signing [22].
- Finally, it is more difficult to write robust malware for mobile phones as software development kits available for most embedded OSs do not offer full support to all features of the underlying OS. This development hurdle has kept the list of attacks on mobile phones relatively short [23].

Considering the above arguments, one could expect that a mobile phone should achieve better intrusion robustness com-

pared to a general purpose computer, however at additional design costs. Certainly the design strategy for a mobile phone used for MFA should emphasize intrusion prevention as a premier design criterion. Historically, the critical entry points for malware on mobile devices have been Bluetooth, SMS messaging, and e-mail [23].

Finally, the option to use a dedicated hardware device that offers tamper-resistance and custom-built intrusion prevention is certainly real. However, due to the strong recent trend for convergence of most mobile applications (portable MP3 player, digital camera, GPS, WiFi, etc.) onto a single device, our objective to fit the functionality of the proposed protocol onto a mobile phone is appealing.

III. The Threat Model

In this section, we consider several threats and address them informally in our security analysis.

Case 1. Loss of $\mathbf{p}$ and the account password. Since the authentication system is based upon the "What I have?" ($\mathbf{p}$) and "What I know?" (password) authentication principles, it is just to acknowledge that if the user loses $\mathbf{p}$ to a malicious party m who knows her password, then m will have full access to user's account A in a classic case of identity theft. Typically, m will be able to impersonate the user until she notifies $\mathbf{s}$ of her loss of $\mathbf{p}$. All two-factor authentication systems are prone to this attack (see Section V).

Case 2. Intrusion into $\mathbf{p}$. This is a special case of the previous attack, where the adversary penetrates $\mathbf{p}$ with an arbitrary program that enables full visibility of its internal storage. To prevent this attack, the construction of $\mathbf{p}$ must be such that software resident on $\mathbf{p}$ should not be able to read $id(\mathbf{p})$ into $\mathbf{p}$'s general purpose registers – reads and writes of $id(\mathbf{p})$ requested by $\mathbf{s}$ should be done using a hardware-only mechanism. That way, intrusion into $\mathbf{p}$ via a **Trojan virus** would equal to **password leakage**.

In a more complicated version of this attack, if the malicious code installed on $\mathbf{p}$ takes control over its user interface, i.e., it could both read from and write to the variables that constitute the user interface, this code could launch arbitrary malicious actions from $\mathbf{p}$ on behalf of the user. Such an attack should demand great sophistication of the malicious party and is unlikely on most modern mobile phones as their operating systems are custom built and difficult to reverse engineer.

Case 3. Phishing $\mathbf{p}$. A traditional phishing attack onto $\mathbf{p}$ via the $\mathbf{c} \bowtie \mathbf{p}$ connection is not possible by construction. When $\mathbf{p}$ is registered with $\mathbf{s}$, our protocol establishes a device identifier, $id(\mathbf{p})$, and stores it at $\mathbf{p}$ along with the $id(\mathbf{p}) \leftrightarrow \mathbf{s}$ association. At a later time, only a TLS protocol with $\mathbf{s}$ and no other entity is allowed to access $id(\mathbf{p})$, thus, any phishing attempt from $\mathbf{c}$ will be outward rejected by $\mathbf{p}$.

Case 4. Ghost display would probably be the most realistic and unnerving attack to the user. Here, the adversary m would have a virtual display resolution twice as large as $\mathbf{c}$'s display. One half of the virtual display would be connected to $\mathbf{c}$'s screen, the other half to another machine $\mathbf{c}_m$ controlled by m; $\mathbf{c}_m$ would also have a parallel user input augmented into $\mathbf{c}$'s I/O. This attack is relatively easy to do by running a malicious virtual machine on $\mathbf{c}$.

When the user logs onto $\mathbf{s}$ via $\mathbf{c}$, m would open another browser window in the part of the virtual display that's visible only to $\mathbf{c}_m$ and use the parallel I/O to browse the data from A at will. Since every write generated by m would need to be confirmed by the user via $\mathbf{p}$, it is unlikely that m could perform any writes into A's state. By realizing that such an attack is a possibility, $\mathbf{s}$ could offer viewing only part of the data related to A over a tunneled TLS connection that minimizes the damage that a "ghost display" attack could cause.

Case 5. $\mathbf{c}_m \bowtie \mathbf{p}$. The case when the user mistakenly connects $\mathbf{p}$ to a malicious computer $\mathbf{c}_m$ instead of a desired physical unit $\mathbf{c}$, is equivalent in the worst-case to the consequences of the "ghost display" attack. Still, the user is more likely to detect the $\mathbf{c}_m \bowtie \mathbf{p}$ attack as the expected Web pages from $\mathbf{s}$ will not appear on $\mathbf{c}$ although $\mathbf{p}$ would show a successful tunneled TLS connection with $\mathbf{s}$. The user would prevent m from browsing her data by instantly disconnecting $\mathbf{p}$ from $\mathbf{s}$.

Finally, note that all realistic threats in this system pose substantially less danger to the user identity compared to state-of-the-art in modern detection-based anti-virus, anti-phishing, and anti-key-logger technologies. Compared to authentication technologies such as SecurID, we point the Reader to the fact that our system addresses a novel problem where we do not focus on providing token-based one-time strong passwords; instead, we focus on fulfilling a small set of security requirements to enable a personal mobile device as an enabling technology to making an `https://...` or some other type of a login session secure on a fully distrusted client computer/terminal. Due to the wireless nature of the $\mathbf{c} \bowtie \mathbf{p}$ connectivity, our protocol is designed to be robust to man-in-the-middle attacks as opposed to SecurID [24]. Compared to the most related work by Mannan and van Oorschot [11], we do extend their threat model to handle "ghost" displays and we offer a significantly simpler protocol that can be handled by traditional TLS handshakes only. We also introduce several additional functionalities that aim at limiting a malicious party who is in control of $\mathbf{c}$ to read user's account state only at times when $\mathbf{p}$ and $\mathbf{s}$ are connected, i.e., $\mathbf{p}$ responds to $\mathbf{s}$'s pings over the tunneled TLS channel.

IV. Reduction to Practice

A. Formal Verification via AVISPA

In order to formally verify the security of the proposed protocol, we first described it using the High Level Protocol Specification Language [25] which included a description of TLS and SRP adopted from [26]. Descriptions of the adversarial scenarios were more demanding compared to standard two-party protocols because our protocol encompassed four entities including the intruder – this additional complexity contributed to uncharacteristically long and resource consuming verification efforts. Next, we verified the protocol specification using the Automated Validation of Internet Security Protocols and Applications analysis tool (AVISPA) [15], [26]. AVISPA is positioned as an industrial-strength technology for the analysis of large-scale Internet security-sensitive protocols and applications. The `ofmc` theorem prover attached to AVISPA [27] reported a search time of 36.7 seconds, 18207 visited nodes

and a search depth of 15 plies. The alternative `cl-atse` theorem prover [28], also part of the AVISPA distribution, reported analyzing 6092736 states within a space of reachable 699886 states in 156.3 seconds. Both theorem provers found no attacks on the protocol.[8]

B. Implementation

Implementation of the proposed protocol is straightforward as it involves only existing building blocks. TLS-auth can be executed with a variety of public-key cryptosystems such as RSA or elliptic curves which all yield different computational costs at $\mathbf{s}$, $\mathbf{c}$, and $\mathbf{p}$. At the server, each TLS handshake costs approximately one (expensive, if RSA is used) private public-key operation, and at the client each TLS handshake costs approximately two (inexpensive) public operations.

TABLE II
PERFORMANCE OF BASIC PUBLIC-KEY ROUTINES ON A 13MHZ ARM PROCESSOR. WE DISPLAY THE REQUIRED MEMORY FOOTPRINT FOR THE CODE, STACK AND HEAP FOR EACH OF THE ROUTINES AND THE NUMBER OF CYCLES REQUIRED TO EXECUTE IT. ALL PERFORMANCE NUMBERS ARE FOR PETER MONTGOMERY'S IMPLEMENTATION OF RSA (1024-BIT) AND ECC (WTLS 163-BIT CURVE #3).

	Code size	Stack	Heap	Cycles
EC-DH	15KB	1.5KB	1.2KB	2.7M = 150ms
EC-DSA SignVer	16KB	1.5KB	1.2KB	5.4M= 300ms
RSA PubEnc	10KB	0.7KB	4.5KB	3.8M= 210ms
RSA PrivDec	20KB	0.8KB	6.5KB	198M= 11s

We present the performance of basic public-key routines on an outdated 13MHz[9] ARM TDMI720T processor in Table II. We display the required memory footprint for the code, stack and heap for each of the routines and the number of cycles required to execute it. All performance numbers are for Peter Montgomery's implementation of RSA (1024-bit) and ECC (ECDH and ECDSA considered over WTLS 163-bit curve #3).

One can notice that the proposed protocol doubles the computational cost on the server as two TLS sessions are required per login. Still, in most applications crypto-related computations per TLS session are dwarfed by overall computation costs of an average TLS session, hence such an overhead is acceptable considering the benefit of MFA.

C. Applications

In order to evaluate the applicability of our protocol, we outline its disadvantages regardless of comparisons to similar technologies. Our platform induces several changes:

- **server-side** – a slight change to the server is required so that it can handle the tunneled TLS protocol. The fact that the server receives a signal that a specific terminal is not trusted when a user connects via a trusted device, calls for an action to reduce read access to identifying data with the user account. This stresses the point that the server side is likely to change opportunistically rather than due to a necessity to accommodate our protocol.
- **terminal-side** – in order to perform the tunneling, the terminal needs two components. First, it needs to install a browser plug-in, e.g., similar to Adobe Flash or Microsoft SilverLight. Second, the terminal must be able to connect to $\mathbf{p}$ using a wired or wireless connection. In case Bluetooth is used, the terminal would also have to be armed with a Bluetooth transceiver. As opposed to most modern smartphones that do have Bluetooth, many terminals are not equipped with it. Upcoming standards such as the Wireless USB[10] aim at facilitating this scenario.
- **trusted-device** – has to install an application, reminiscent of a stripped down Web browser to handle the user interface to our protocol. Similarly to the server-side, we believe that consumers are likely to act opportunistically rather than due to a necessity.

	$\mathbf{s}$	$\mathbf{c}$	$\mathbf{p}$
$\mathbf{c}$ = personal computer	✓	✓	✓
$\mathbf{c}$ = public terminal	✓	○	✓
$\mathbf{c}$ = point-of-sale	✓	✓	✓

TABLE III
EASE OF USAGE AND DEPLOYMENT OF THE PROPOSED PROTOCOL WITH RESPECT TO A GRID OF POTENTIAL APPLICATION TARGETS VS. SYSTEM COMPONENTS THAT DEMAND ALTERATIONS TO OPERATE THE PROTOCOL. SYMBOL ✓ DENOTES CONVENIENT DEPLOYMENT. SYMBOL ○ DENOTES LIKELY DIFFICULTIES IN USAGE.

It is important to evaluate how the burden on $\mathbf{c}$ imposed by our protocol, would resonate in the main application domains. We identify three main settings for $\mathbf{c}$ and illustrate their fit with respect to our protocol in Table III:

- **personal computers** – include home, office, mobile computing devices with full I/O. These devices are under the control of one or a few users who are not likely to find installation of a browser plug-in difficult. Similarly, those who appreciate the value of MFA, are likely to support their authentication mechanisms with a Bluetooth transceiver (costs less than US$5). We point the Reader to the fact that the cardinality of this setting is in the order of 10^9 machines worldwide.
- **public terminals** – located at public computing centers and Internet cafés could be under a ban on installing any browser plug-ins because of fears for system security. In such a setting, our platform would not gain ground. The cardinality of this setting is in the order of 10^7 machines worldwide.
- **points-of-sale** – could use our protocol to run transactions against their customer's mobile phones. Here, $\mathbf{s}$ would represent a bank or a credit card company, $\mathbf{c}$ a cash register at, say a supermarket, and $\mathbf{p}$ would represent a customer's mobile phone. Instead of typing in a password, a PIN could facilitate the data entry at the consumer's side. The consumer would approve of a transaction after browsing through a downloaded shopping cart and agreeing to the price with an entry of a PIN. The cost of this transaction for both the merchant as well as the credit card company would be significantly lower than when running a transaction via a dedicated communication channel.

[8]For anonymity, we did not include access to the protocol specification for the AVISPA verifier. The final version will certainly include such a pointer.

[9]CPUs in current smartphones are commonly clocked in excess of 400MHz.

[10]See http://en.wikipedia.org/wiki/Wireless_USB for more details.

In addition, this payment mechanism offers a significant advantage over credit cards because of the impossibility of skimming. Although the cardinality of this setting is in the order of 10^8 locations, points-of-sale drive virtually the entire world economy. Thus, the applicability of our scheme to this scenario is of particular importance.[11]

Finally, the type of applications that would benefit from our platform include various financial services (e.g., credit card companies, banks, investment brokers, etc.), account management for on-line bill-payments, vaults for health records, government agencies, storage and application services for cloud computing, personal communication systems such as e-mail or instant messaging, etc. Note that the system that we propose does not establish user identities – our protocol solely enables authentication with substantially improved security especially against nowadays most common attacks such as cross-site scripting, key-loggers, and phishing.

V. Related Work

In light of an industry- and government-driven demand for secure authentication, several schemes that utilize various approaches, have been proposed to date. For example, the US federal banking regulators have concluded that the basic user ID and password are not sufficient to protect against fraud and issued guidelines for banks to implement multi-factor authentication to mitigate risks in online banking [29].

Password-Authenticated Key-Exchange

The first successful password-authenticated key-agreement method was the Encrypted Key Exchange (EKE) protocol [30]. Although several of the first methods were flawed, the surviving and enhanced forms of EKE effectively amplified a shared password into a shared key, which could then be used for encryption and/or message authentication. The first provably-secure PAKE protocols were given in [31] and [32]. These protocols were proven secure in the so-called random oracle model; the first protocols proven secure under standard assumptions were [33] and [34]. In our work, we rely on a traditional PKI/TLS-based key exchange, thus we do not review and compare our work to non-PKI, password-authenticated key-exchange protocols.

One-Time Password Systems

RSA's security key-fob, SecurID, is a dedicated device that uses token-based authentication to enable strong one-time passwords [13], [24]. SecurID consists of a token – a piece of hardware (e.g., USB attachment) or software (e.g., a "soft token" for a cell-phone) – assigned to a computer user that generates an authentication code at fixed intervals[12] using a built-in clock and the card's factory-encoded random key (known as the "seed"). The seed is different for each token, and is loaded into the SecurID server. The token hardware is designed to be tamper-resistant to deter its reverse engineering. The seed is the secret key used to generate one-time passwords. A user authenticating to a network resource needs to enter both a personal identification number and the number being displayed at that moment on their token. The server, which also has a real-time clock and a database of valid cards with the associated seeds, computes what number the token is supposed to be showing at that moment, checks it against what the user entered, and makes the decision to allow or deny access. SecurID is prone to the man-in-the-middle attack [35], [36], [37] as experienced by the Citibank in July 2006 [38]. Passwords generated by time-based generators may face time synchronization issues between a client device and the server [39]. An alternative that does not use tokens, has been deployed in Telcordia's S/KEY[13], a one-time password system based upon Lamport's authentication protocol [40].

Most of the vulnerabilities of traditional one-time password systems could be resolved by using a PKI-based protocol that authenticates the remote server as a starting point to a successful user-server hand-shake. Since we follow this authentication paradigm we do not compare our protocol to one-time passwords.

A. Two-factor Authentication

Researchers have proposed several schemes that use a trusted personal device such as a mobile phone to improve the security of Web surfing. Balfanz and Felten introduced the splitting trust paradigm that partitions an application (e-mail client in their case) between a trusted portable device and a distrusted PC [8]. An application installed on a hand-held computer would prompt the user to selectively approve the process of digitally signing and sending a message to a server. Although not tailored for MFA objectives, their work was the first to use a general purpose portable device as a source of trust. Parno et al. proposed an anti-phishing technology [41] based on server certificates resident on the client.

Jackson et al. introduced the notion of a transaction generator (TG) where instead of relying on stolen credentials, a TG simply waits for the user to log in to his account and then issue transactions on behalf of the user. The authors discuss rootkit-like methods that allow TGs to hide their tracks, and explore a number of mitigation techniques, including transaction confirmation [9].

The *most related research* is that of Mannan and van Oorschot [11] who introduced a protocol which cryptographically separates a user's long-term secret input from $\mathbf{c}$; $\mathbf{c}$ would perform most computations but would have access only to temporary secrets. The user's password would be input through $\mathbf{p}$; $\mathbf{p}$ would provide user's long-term secrets to a $\mathbf{c}$ only after encrypting them using a pre-installed, "correct" public key of $\mathbf{s}$. We expand upon this protocol by proposing a tunneled TLS channel between $\mathbf{p}$ and $\mathbf{s}$ that could be used for verification of connectivity between $\mathbf{p}$, $\mathbf{c}$, and $\mathbf{s}$ in addition to sending credentials. Thus, we provide better protection from "ghost" user interfaces (i.e., transaction generators) for both "reads" and "writes" to the server-managed account state.

[11] We do not review the related work in mobile payment mechanisms, because most methods in use are touch-less, RFID-based and as such, prone to skimming.

[12] Usually 30 or 60 seconds.

[13] Alternatively known as OPIE or OTP.

Wu et al. were the first to propose a proxy-based authentication scheme that employed a trusted proxy server which stored user's credentials [42]. In their informal proposal, a user would use a distrusted machine to send an access request and her username to the trusted proxy which would respond with a random session name. The same session name is then sent to the users mobile phone as an SMS message. The user would compare the displayed session names and accept the session on the mobile phone if there were a match. The proxy would use the stored credentials to login to the desired server. One effective constructive protocol for proxy-based authentication has been proposed by Florencio and Herley [43]. Proxy-based schemes leave a lot to be desired as they require users to trust the proxy, i.e., a sophisticated intrusion into the proxy would reveal users' passwords, and their system cost is linearly proportional to the traffic exhibited over the proxy. Our and related protocols [11] do not suffer from these problems – however, proxy-based authentication could be set up so that the server application is intact [43].

Finally, we review some benefits that our scheme introduces compared to related work. A third party such as a proxy does not participate in our protocol, thus, the user does not need to trust anyone else but $\mathbf{c}$ and to a certain extent, $\mathbf{p}$. To run the protocol, the trusted device does not need to be connected to a global data network via SMS, WiFi, or other data channels. Thus, our protocol is robust to outage of the voice and/or data wireless service. The users in our scheme do not need to type long dynamic passwords onto their personal devices – they also do not need to carry with them at all times print-outs of lists of viable one-time passwords. We allow the protocol to run via the set of functionalities included in most generic smart-phones which greatly facilitates implementation efforts. Loss of the trusted device is not a threat to security as long as the corresponding passwords are not compromised otherwise – the device itself does not store any passwords. Our threat model assumes that the adversary has *full* access to the distrusted terminal. And, last but not least, our protocol can use any *public* communication channel between $\mathbf{c}$ and $\mathbf{p}$ such as Bluetooth.

VI. Summary

We propose a simple TLS-based protocol that enables an off-the-shelf smartphone to assist a password-based user authentication from a fully distrusted terminal. The set of appealing applications includes a mobile payment platform that is robust to skimming and loss of the payment device, while being convenient for use. In the general computing milieux, our protocol prevents phishing, key-loggers, cross-site scripting, invasive virtual machines, and transactional generators from achieving their objectives.

References

[1] K. Delhagen, et al. Consumer Concerns Over Identity Theft And Fraud. Forrester Research report, June 21, 2004.

[2] N. Lambert, et al. Consumers Combat Their Internet Fears With Free Protection Measures. Forrester Research report, June 19, 2008.

[3] B. Cundiff, et al. Online Banking. Jupiter research report, BNK05-C03, April 1, 2005.

[4] Wikipedia. Identity theft. http://en.wikipedia.org/wiki/Identity_theft#Spread_and_impact

[5] D.V. Klein. Foiling the Cracker; A Survey of, and Improvements to Unix Password Security. USENIX Security Workshop, 1990.

[6] A.K. Jain, et al. Handbook of Biometrics. Springer, 2007.

[7] T. Matsumoto, et al. Impact of artificial gummy fingers on fingerprint systems. SPIE, Vol.4677, 2002.

[8] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. USENIX Security Symposium, 1999.

[9] C. Jackson, et al. Transaction generators: root kits for the web. USENIX Workshop on Hot Topics in Security, 2007.

[10] J. Rafail. Cross-Site Scripting Vulnerabilities. CERT Coordination Center, Carnegie Mellon University, 2001. http://www.cert.org/archive/pdf/cross_site_scripting.pdf

[11] M. Mannan and P.C. van Oorschot. Using a personal device to strengthen password authentication from untrusted computer. FC, 2007.

[12] S. Christey and R.A Martin. Vulnerability Type Distributions in CVE (version 1.1). MITRE Corporation, May 22, 2007. http://cwe.mitre.org/documents/vuln-trends/index.html

[13] RSA SecurID: Securing Your Future with Two-Factor Authentication. http://www.rsa.com/node.aspx?id=1156

[14] T. Dierke and E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.2. IETF draft TLS-RFC5246, 2008. http://tools.ietf.org/html/rfc5246

[15] A. Armando, et al. The AVISPA Tool for the automated validation of internet security protocols and applications. International Conference on Computer Aided Verification, Vol.3576, pp.281-285, 2005.

[16] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0, January 1999. Status: Proposed Standard.

[17] L.C. Paulson. Inductive analysis of the internet protocol TLS. ACM Trans. on Computer and System Security, Vol.2, no.3, pp.332–351, 1999.

[18] T. Wu. The Secure Remote Password Protocol. Internet Society Network and Distributed System Security Symposium, pp.97–111, 1998.

[19] T. Wu. SRP-6: Improvements and Refinements to the Secure Remote Password Protocol. IEEE P1363 Working Group, 2002. http://srp.stanford.edu/design.html

[20] Y. Lu, et al. The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. CRYPTO, 2005.

[21] G.S. Kc, et al. Countering code-injection attacks with instruction-set randomization. ACM Conference on Computer and Communications Security, pp.272–280, 2003.

[22] D. Kirovski and M. Drinic. A Hardware-Software Platform for Intrusion Prevention. IEEE MICRO 37, 2004.

[23] N. Leavitt. Mobile phones: the next frontier for hackers? Computer, Vol.38, no.4, pp.20–23, 2005.

[24] Wikipedia: RSA SecurID. http://en.wikipedia.org/wiki/SecurID

[25] The High Level Protocol Specification Language. AVISPA project, 2003. http://www.avispa-project.org/delivs/2.1/d2-1.pdf

[26] Avispa a tool for Automated Validation of Internet Security Protocols. http://www.avispa-project.org/package/user-manual.pdf.

[27] D.A. Basin, et al. Ofmc: A symbolic model checker for security protocols. International Journal on Information Security, Vol.4, no.3, pp.181-208, 2005.

[28] M. Turuani. The cl-atse protocol analyser. RTA, Vol.4098, pp.277-286, 2006.

[29] Federal Financial Institutions Examination Council. Authentication in an internet banking environment. http://www.ffiec.gov/pdf/authenticationguidance.pdf

[30] S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. IEEE Security and Privacy, Oakland, May 1992.

[31] M. Bellare, et al. Authenticated Key Exchange Secure against Dictionary Attacks. Eurocrypt, 2000.

[32] V. Boyko, et al. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. Eurocrypt, 2000.

[33] O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. Crypto, 2001.

[34] J. Katz, et al. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. Eurocrypt, 2001.

[35] A. Biryukov, et al Cryptanalysis of the alleged SecurID hash function. SAC, 2003.

[36] A. Biryukov, et al. Recent attacks on alleged SecurID and their practical implications. Computers & Security, Vol.24, no.5, pp.364-370, 2005.

[37] S. Contini and Y.L. Yin. Fast software-based attacks on SecurID. Fast Software Encryption Workshop, pp.454-471, 2004.

[38] Netcraft. Phishing attacks continue to grow in sophistication, January 2007. http://news.netcraft.com/archives/2007/01/15/phishingattackscontinuetogrowinsophistication.html.
[39] G.G. Xie, et al. Quantifying effect of network latency and clock drift on time-driven key sequencing. ICDCS, 2002.
[40] L. Lamport. Password Authentication with Insecure Communication. Communications of the ACM, Vol.24, no.11, pp.770-772, 1981.
[41] B. Parno, et al. Authentication and fraud detection: Phoolproof phishing prevention. FC, 2006.
[42] M. Wu, et al. Secure web authentication with mobile phones. DIMACS Workshop on Usable Privacy and Security Software, 2004.
[43] D. Florencio and C. Herley. One-time Password Access to Any Server Without Changing the Server. ISC 2008.