Leveraging Personal Devices for Stronger Password Authentication from Untrusted Computers*

Mohammad Mannan, University of Toronto, Canada, and
P.C. van Oorschot, Carleton University, Canada

Abstract

Internet authentication for popular end-user transactions, such as online banking and e-commerce, continues to be dominated by passwords entered through end-user personal computers (PCs). Most users continue to prefer (typically untrusted) PCs over smaller personal devices for actual transactions, due to usability features related to keyboard and screen size. However most such transactions and their existing underlying protocols are vulnerable to attacks including keylogging, phishing, and pharming, which can extract user identity and sensitive account information allowing account access. We propose a simple approach called Mobile Password Authentication (*MP-Auth*) to counter such attacks, which cryptographically separates a user's long-term secret input (typically low-entropy password) from the client PC, and offers transaction integrity. The PC continues to be used for most of the interaction and computations but has access only to temporary secrets, while the user's long-term secret is input through an independent personal trusted device such as a cellphone which makes it available to the PC only after encryption under the intended far-end recipient's public key. MP-Auth is intended to safeguard passwords from PC malware, phishing/DNS attacks, and to provide transaction integrity against session hijacking. MP-Auth expects users to input passwords only to a personal device, and be vigilant while confirming transactions from the device. To facilitate a comparison to MP-Auth, we also provide a comprehensive survey of web authentication techniques that use an additional factor of authentication such as a cellphone or hardware token; this survey may be of independent interest. A proof sketch of MP-Auth using the Protocol Composition Logic (PCL) is also provided.

1 Introduction

In the current Internet environment, most consumer computers are infected with one or more forms of spyware or malware. Internet connected PCs are not 'safe' anywhere; an improperly patched home or public computer generally survives only minutes.[1] Semantic attacks such as phishing also widely target general Internet users. Software keyloggers are typically installed on a user PC along with common malware and spyware [48]. An increasing number of phishing sites also install keyloggers on user PCs, even when users do not explicitly download or click any link on those sites [56]. Many of these attacks attempt to extract user identity and sensitive account information for unauthorized access to users' financial accounts; for example, user names and passwords for thousands of bank accounts have been found on an online storage site reportedly gathered by a botnet [57]. Passwords enjoy ubiquitous use for online authentication even in such an environment, although many more secure (typically also more complex and costly) authentication protocols have been proposed. Due to the usability and ease of deployment, most North American financial transactions over the Internet are still authenticated through only a password. Hence passwords are a prime target of attackers, for economically-motivated exploits including those targeting online bank accounts and identity theft.

As one example of highly sensitive Internet services, online banking often requires only a bank card number (as *userid*) and password for authentication. Users input these credentials to a bank website to access their accounts. An attacker can easily collect these long-term secrets by installing a keylogger program on a client PC, or embedding a JavaScript keylogger [58] on a compromised website. As plaintext sensitive information is input to a client PC, malware on the PC has instant access to these (reusable) long-term secrets. We argue (as do others – e.g., see Laurie and Singer [37], Kursawe and Katzenbeisser [36]) that for some common applications, passwords are too important to input directly to a typical user PC on today's Internet; and that the user PC should no longer be trusted with such

*Version: February 1, 2010. This paper extends an earlier publication [38], and is published with permission of IFCA, the copyright holder of the preliminary paper. This work was carried out while M. Mannan was a PhD student at Carleton University.

[1] An average time between attacks of 3 minutes, as of Jan. 18, 2010, is reported at `http://isc.sans.org/survivaltime.html`.

plaintext long-term secrets, which are intended to be used for user authentication to a remote server. Additionally, phishing attacks can collect plaintext reusable userid-password pairs even if a user's PC is malware-free (through e.g., domain name hijacking [29], or DNS flaws [34, 14]).

To safeguard a long-term password, we build on the following simple idea: use a hand-held personal device, e.g., a cellphone or PDA (personal digital assistant) to encrypt the password (combined with a server generated random challenge) under the public key of an intended server, and relay through a (possibly untrusted) PC only the encrypted result in order to login to the server website. This simple challenge-response effectively turns a user's long-term password into a one-time password in such a way that long-term passwords are not revealed to phishing websites, or keyloggers on the untrusted PC.

The resulting protocol, called *MP-Auth* (short for *M*obile *P*assword *Auth*entication), is proposed primarily to protect a user's long-term password input through an untrusted (or untrustworthy) client PC. The use of a mobile device in MP-Auth is intended to protect user passwords from easily being recorded and forwarded to malicious parties, e.g., by keyloggers installed on untrustworthy commodity PCs. For usability and other reasons, the client PC is used for the resulting interaction with the website, and performs most computations (e.g., session encryption, HTML rendering etc.) but has access only to temporary secrets. The capabilities we require from a mobile device include encryption, alpha-numeric keypad, short-range network connection (wire-line or Bluetooth), and a small display. Although we highlight the use of a cellphone, the protocol can be implemented using any similar "trustworthy" device (e.g., PDAs or smart-phones), i.e., one free of malware. There are known attacks against mobile devices [26], but the trustworthiness of such devices is currently more easily maintained than a PC (see Section 3.3 for further discussion on mobile device security). Note that, despite our use of a personal device in conjunction with a PC, MP-Auth is essentially a single-factor (password-only) authentication protocol; we do not use (or store) any other secrets from the device.

To protect a password from being revealed to a phishing site, the password is encrypted with the "correct" public key of an intended website (e.g., a bank). Thus MP-Auth protects passwords from keyloggers and various forms of phishing attacks (including deceptive malware, DNS-based attacks or pharming, as well as false bookmarks). MP-Auth also protects against session hijacking, by providing transaction integrity through a transaction confirmation step. Unlike standard two-factor techniques, MP-Auth does not store any secret on the mobile device.

Phishing attacks have been discussed in the literature since 1997 (see [22]); however, few, if any, anti-phishing solutions exist today that are effective in practice. In addition to several anti-phishing proposals (e.g., [58]), there exist many software tools for detecting spoofed websites (e.g., eBay toolbar, SpoofGuard, Spoofstick, Netcraft toolbar). However, most of these are susceptible to keylogging attacks,[2] and phishing toolbars are barely effective in reality [73]. On the other hand, several authentication schemes which use a trusted personal device, generally prevent keyloggers, but do not help against phishing or session hijacking attacks. New malware attacks (*bank-stealing Trojans*, e.g., Win32.Grams [10], Trojan.Silentbanker [62]; see also web-rootkits [32]) attempt to perform fraudulent transactions in real-time after a user has logged in, instead of collecting userids and passwords for later use. Most existing or proposed techniques are susceptible to these new attacks, including e.g., Phoolproof [53] and two-factor authentication such as a password and a passcode generator token (e.g., SecurID). In contrast, the primary goals of MP-Auth are twofold: protect passwords from both keyloggers and phishing sites, and provide transaction integrity.

Our contributions. We propose MP-Auth, a protocol for online authentication using a personal device such as a cellphone in conjunction with a PC. The protocol provides the following benefits without requiring a trusted proxy, or storing long-term secrets on a cellphone (cf. [12, 53]).

1. Keylogging Protection. A client PC does not have access to long-term user secrets. Consequently keyloggers (software or hardware) on the PC cannot access critical passwords.
2. Phishing Protection. Even if a user is directed to a spoofed website, the website will be unable to decrypt a user password. Highly targeted phishing attacks (*spear phishing*) are also ineffective against MP-Auth.
3. Pharming Protection. In the event of domain name hijacking [29, 34], MP-Auth does not reveal a user's long-term password to attackers. It also protects passwords when the DNS cache of a client PC is poisoned.
4. Transaction Integrity. With the transaction confirmation step (see Section 2) in MP-Auth, a user can detect any (critical) unauthorized transaction during a login session, even when an attacker has complete control over the user PC.

We also provide a comprehensive survey of related authentication schemes used in practice and/or proposed to date, and compare these to MP-Auth; this survey may be of independent interest. MP-Auth has been analyzed using

[2] PwdHash [58] can protect passwords from JavaScript keyloggers, but not software keyloggers on client PCs.

AVISPA [4]; no attacks were found. A formal proof sketch of MP-Auth using the Protocol Composition Logic (PCL) [15, 28, 60] is also provided. A prototype of MP-Auth for performance testing has also been implemented.

Organization. The MP-Auth protocol, threat model and operational assumptions are discussed in Section 2. A brief informal analysis of protocol messages, discussion on how MP-Auth prevents common attacks, and circumstances under which MP-Auth fails to provide protection are outlined in Section 3. Discussion on usability and deployment issues related to MP-Auth are provided in Section 4. The performance and basic implementability of MP-Auth is discussed in Section 5. Related work, including commercial one-time password generators, and a number of web authentication techniques proposed in the literature, is discussed in Section 6. Section 7 concludes. Appendix A provides AVISPA test code for MP-Auth and related discussion. The PCL analysis of MP-Auth is provided in Appendix B.

2 MP-Auth: A Protocol for Online Authentication

In this section, we describe the MP-Auth protocol, including threat model assumptions.

Threat model and operational assumptions. The primary goals of MP-Auth are to protect user passwords from malware and phishing websites, and to provide transaction integrity. We assume that a bank's "correct" public key is available to users (see below for discussion on public key installation), and mobile devices are malware-free. Public keys of each target website must be installed on the device. (We assume that there are only a few financially critical websites that a typical user deals with.) A browser on a PC uses a bank's SSL certificate to establish an SSL connection with the bank website (as per common current practice). The browser may be duped to go to a spoofed website, or have a wrong SSL certificate of the bank or the verifying certificating authority. The protocol does not protect user privacy (of other than the user's password) from an untrusted PC; the PC can record all transactions, generate custom user profiles etc. Visual information displayed to a user on a PC screen is also not authenticated by MP-Auth, i.e., a malicious PC can display misleading information to a user without being (instantly) detected. Denial-of-service (DoS) attacks are not addressed. A communication channel between a personal device and PC is needed, in such a way that malware on the PC cannot infect the personal device.[3] However, MP-Auth does not rely on the security of this channel, i.e., attackers can modify or insert messages between the device and PC.

U, M, B, S	User, a cellphone, a browser on the untrusted user PC, and the server, respectively.
ID_S, ID_U	Server ID and user ID, respectively. ID_U is unique in the server domain.
P	Long-term (pre-established) password shared between U and S.
R_S, R_M	Random nonces generated by S and M, respectively.
$\{data\}_K$	Symmetric (secret-key) authenticated encryption (see e.g., [25], [7]) of *data* using key K.
$\{data\}_{E_S}$	Asymmetric (public-key) encryption of *data* using S's long-term public key E_S.
$X.Y$	Concatenation of X and Y.
K_{BS}	Symmetric encryption key shared between B and S (e.g., an SSL key).
$f(\cdot)$	A cryptographically secure hash function.

Table 1: Notation used in MP-Auth

Protocol steps in MP-Auth. For notation see Table 1. Before the protocol begins, we assume that user U's cellphone M is connected to B (via wire-line or Bluetooth). The protocol steps are described below (see also Fig. 1).

1. U launches a browser B on the untrusted PC, and visits the bank website S.
2. B and S establish an SSL session; let K_{BS} be the established SSL secret key.
3. S generates a random nonce R_S, and sends the following message to B.

$$B \leftarrow S: \ \{ID_S.R_S\}_{K_{BS}} \tag{1}$$

4. B decrypts message (1) and forwards it to M.

$$M \leftarrow B: \ ID_S.R_S \tag{2}$$

5. M displays ID_S, and prompts the user to input the userid and password for S. A userid (e.g., bank card number) may be stored on the cellphone for convenience; the password should not be stored or auto-remembered.

[3]The first *crossover* virus was reported [45] in February 2006.

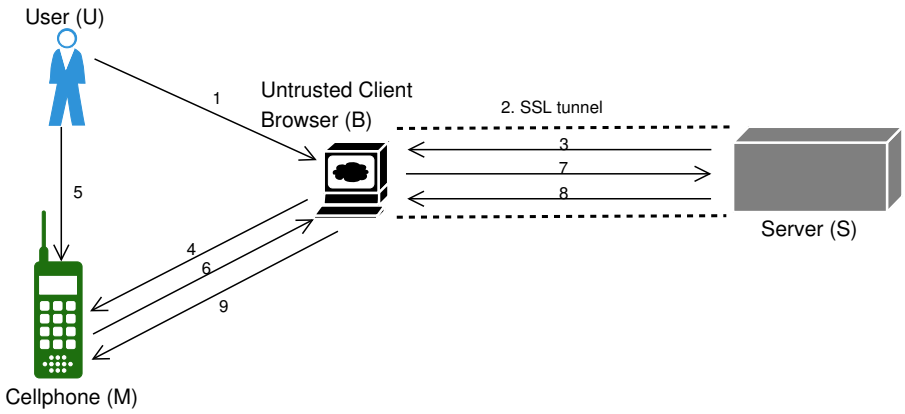

Figure 1: MP-Auth protocol steps

6. M generates a random secret nonce R_M and encrypts R_M using E_S. M calculates the session key K_{MS} and sends message (4) to B (here, the userid ID_U is, e.g., a bank card number).

$$K_{MS} = f(R_S.R_M) \tag{3}$$

$$M \rightarrow B: \ \{R_M\}_{E_S}.\{f(R_S).ID_U.P\}_{K_{MS}} \tag{4}$$

7. B (via SSL) encrypts message (4) with K_{BS}, and forwards the result to S.
8. From message (4), after SSL decryption, S decrypts R_M using its corresponding private key, calculates the session key K_{MS} (as in equation (3)), decrypts the rest of message (4), and verifies P, ID_U and R_S. Upon successful verification, S grants access to B on behalf of U. S sends the following message for M to B (indicating login success).

$$B \leftarrow S: \ \{\{f(R_M)\}_{K_{MS}}\}_{K_{BS}} \tag{5}$$

9. B forwards $\{f(R_M)\}_{K_{MS}}$ to M. M decrypts to recover $f(R_M)$ and verifies its local copy of R_M. Then M displays success or failure to U.

Transaction integrity confirmation. In MP-Auth, M and S establish a session key K_{MS} known only to them; malware on a user PC has no access to K_{MS}. Attackers may modify or insert transactions through the untrusted PC. To detect and prevent such transactions, MP-Auth requires explicit transaction confirmation by U (through M). The following messages are exchanged (after step 9) for confirmation of a transaction with summary details T (R_{S1} is a server generated random nonce, used to prevent replay).

$$M \xleftarrow{\{T.R_{S1}\}_{K_{MS}}} B \xleftarrow{\{\{T.R_{S1}\}_{K_{MS}}\}_{K_{BS}}} S \tag{6}$$

$$M \xrightarrow{\{f(T.R_{S1})\}_{K_{MS}}} B \xrightarrow{\{\{f(T.R_{S1})\}_{K_{MS}}\}_{K_{BS}}} S \tag{7}$$

M displays T to U in a human-readable way (e.g., "Pay \$10 to Vendor V from checking account C"), and asks for confirmation (yes/no). When the user confirms T, the confirmation message (7) is sent from M to S (via B). From message (7), S retrieves $f(T.R_{S1})$, and verifies with its local copy of T and R_{S1}. Upon successful verification, T is committed. Instead of initiating a confirmation step after each transaction, transactions may be confirmed in batches (e.g., four transactions at a time); then, T will represent a batch of transactions in the above message flows. The user-interface (UI) design of the confirmation step is extremely important to reduce dangerous errors; a user study [2] on SMS-based transaction authorization methods reported that 21% of participants failed to detect modified transaction details as displayed on a cellphone. Uzun et al. [65] provides a comparative study and user-testing of different approaches to such UI design in the context of secure device pairing.

Some transactions may not require confirmation from the mobile device. For example, adding a new user account or setting up an online bill payment for a phone company should require user confirmation, but when paying a monthly bill to that account, omitting the confirmation step would seem to involve little risk. Similarly, fund

transfers between user accounts without transaction confirmation may be an acceptable risk. A bank may configure the set of sensitive transactions that will always require the confirmation step. Deciding to omit the requirement of explicit confirmation should be done with hesitation. As reported in a Washington Post article [68], attackers compromised customers' trading accounts at several large U.S. online brokers, and used the customers' funds to buy thinly traded shares. The goal is to boost the price of a stock they already have bought and then to sell those shares at the higher price. This incident indicates that seemingly innocuous transactions may be exploited by attackers if transactions requiring confirmation are not diligently selected by banks.

Password setup/renewal. In order to secure passwords from keyloggers during password renewal, we require that the password is entered through the cellphone keypad. We assume that the initial password is set up via a trustworthy out-of-band method (e.g., regular phone, postal mail), and U attempts a password renewal after successfully logging into S (i.e., K_{MS} has been established between M and S). The following message is forwarded from M to S (via B) during password renewal (P_{old} and P_{new} are the old and new passwords respectively).

$$M \xrightarrow{X, \text{ where } X = \{ID_U.P_{old}.P_{new}\}_{K_{MS}}} B \xrightarrow{\{X\}_{K_{BS}}} S \tag{8}$$

Public key installation. One of the greatest practical challenges of deploying public key systems is the distribution and maintenance (e.g., revocation, update) of public keys. MP-Auth requires a service provider's public key to be distributed (and updated as needed) and installed into users' cellphones. The distribution process may vary depending on service providers; we recommend that it not be primarily Internet-based. Considering banking as an example, we visualize the following key installation methods (but emphasize that we have not user-tested these for usability):

1. at a bank branch, during an account setup (see Section 4 for usability issues);
2. through in-branch ATM interfaces (hopefully free of "fake" ATMs);
3. through a cellphone service (authenticated download) as data file transfer; and
4. for web-only banks: through removable flash memory card for cellphones (e.g., microSD card) mailed to users, containing the public key. In this case, one might consider the cost of an attack involving fake mailings o users.

A challenge-response protocol or integrity cross-checks (using a different channel, e.g., see [66]) should ideally be used to verify the public key installed on a cellphone, in addition to the above procedures. For example, the bank may publish its public key on the bank website, and users can cross-check the received public key, e.g., comparing *visual hashes* [54] or *public passwords* [27]; to reduce usability issues, an automated cross-check would be preferable. The public key may also be distributed as a trusted third-party-signed certificate (e.g., similar to current SSL certificates as verified by root certificates pre-installed in browsers). However, we do not expect users to "verify" a certificate on their own, e.g., through UI cues as used in PC browsers. The public key, as a key file or certificate, must be installed in a secure fashion (as outlined in the above example methods).

Authentication without a personal device. It may happen that while traveling, a user may lose her cellphone, and cannot use MP-Auth to log in to her bank. As a secondary authentication technique in such emergency situations, one-time codes could be used. For example, banks may provide users a list of pass-codes (e.g., 10 digit numbers) printed on a paper which can be used for secondary login; i.e., a userid and pass-code entered directly on a bank login page allows emergency access to a user's account. However, such logins should be restricted to a limited set of transactions, excluding any sensitive operations such as adding a payee or changing postal address (cf. TwoKind authentication [5]).

3 Security and Attack Analysis

In this section, we first consider an informal security analysis of MP-Auth. We motivate a number of design choices in MP-Auth messages and their security implications, and discuss several attacks that MP-Auth is resistant to. Successful but less likely attacks against MP-Auth are also listed.

As a confidence building step, we have tested MP-Auth using the AVISPA (Automated Validation of Internet Security Protocols and Applications) [4] analysis tool, and found no attacks. AVISPA is positioned as an industrial-strength technology for the analysis of large-scale Internet security-sensitive protocols and applications. AVISPA test code for MP-Auth and discussion are provided in Appendix A. The test code is also available online.[4] A formal proof sketch of MP-Auth using the Protocol Composition Logic (PCL) [15, 28, 60] is also provided in Appendix B.

[4] `http://people.scs.carleton.ca/~mmannan/mpauth/`

3.1 Partial Message Analysis and Motivation

Here we provide motivation for various protocol messages and message parts. In message (1), S sends a fresh R_S to B, and B forwards ID_S, R_S to M. ID_S is included in message (2) so that M can choose the corresponding public key E_S. When U starts a session with S, a nearby attacker may start a parallel session from a different PC, and grab M's response message (4) (off-the-air, from the Bluetooth connection) to login as U. However, as S generates a new R_S for each login session (i.e., U and the attacker receive different R_S from S), sending message (4) to S by any entity other than B would cause a login failure.

The session key K_{MS} shared between M and S, is known only to them. Both M and S influence the value of K_{MS} (see equation (3)), and thus a sufficiently random K_{MS} is expected if either of the parties is honest (as well as capable of generating secure random numbers); i.e., if a malicious party modifies R_S to be 0 (or other values), K_{MS} will still be essentially a random key when M chooses R_M randomly (i.e., R_M has enough entropy). To retrieve P from message (4), an attacker must guess K_{MS} (i.e., R_M) or S's private key. If both these quantities are sufficiently large (e.g., 160-bit R_M and 1024-bit RSA key E_S) and random, an offline dictionary attack on P becomes computationally infeasible. We encrypt only a small random quantity (e.g., 160-bit) by E_S, which should always fit into one block of a public key cryptosystem (including elliptic curve). Thus MP-Auth requires only one public key encryption. Browser B does not have access to K_{MS} although B helps M and S establish this key. With the transaction integrity confirmation step, all (important) transactions must be confirmed from M using K_{MS}; therefore, any unauthorized (or modified) transaction by attackers will fail as attackers do not have access to K_{MS}.

Analysis of simplified authentication messages. For the authentication phase in MP-Auth, the browser simply forwards messages between the personal device and the web server. Therefore for analysis, we simplify the MP-Auth authentication messages (steps 1 through 9 in MP-Auth, see Section 2) in the following way.

$$M \leftarrow S : ID_S.R_S \tag{9}$$

$$M \rightarrow S : \{R_M\}_{E_S}.\{f(R_S).ID_U.P\}_{K_{MS}}, \text{ where } K_{MS} = f(R_S.R_M) \tag{10}$$

$$M \leftarrow S : \{f(R_M)\}_{K_{MS}} \tag{11}$$

We assume that before the protocol run, M and S establish a shared secret P, and M has an authentic copy of S's public key E_S (e.g., obtained using an out-of-band method). At the end of the protocol run, the goals are to achieve mutual authentication between M and S, and establish a fresh session key known only to M and S. We assume that all protocol messages can be intercepted, modified, and stored by an attacker. Also the attacker can attempt to impersonate either M or S. However, none other than M and S knows P, and the private key corresponding to E_S is known only to S.

At the end of message (9), S believes that it has sent a fresh nonce R_S to M. M receives R_S, but it has no assurance of the true identity of S. For message (10), M generates a fresh nonce R_M, and an encryption key K_{MS} from R_M and R_S. M believes that K_{MS} is a fresh encryption key as R_M is freshly generated by M. M encrypts R_M using E_S so that none other than S can generate K_{MS} and learn P. When S verifies P from message (10), i.e., P matches the expected pre-shared secret, M is authenticated to S. Also, $f(R_S)$ indicates to S that the current protocol run is fresh, as S knows that R_S is freshly generated. S believes that K_{MS} is a fresh encryption key as R_S is freshly generated by S in the current protocol run. S also believes that M knows K_{MS} as M is able to encrypt P with the key. Hence for S the protocol goals have been established, i.e., S learns the authenticated identity of M (i.e., the user U), and S and M share a fresh encryption key K_{MS}. When M receives message (11) and verifies $f(R_M)$, M believes the following: the communication involves the authenticated (true) party S, as none other than S can retrieve R_M in message (10); that S knows K_{MS}; and that the current protocol run is fresh. Hence the protocol goals for M are also established. Messages (9) and (10) are cryptographically linked by R_S, and messages (10) and (11) are cryptographically linked by R_M. This chaining prevents replay and interleaving attacks [17].

In message (9), S generates R_S and stores it for future use in decrypting message (10). Attackers can initiate a large number of login attempts (e.g., through a botnet), and force S to generate and store many unwanted values of R_S, and thus exhaust resources on S. To limit this attack, a suitable time-out value must be chosen after which S will drop a protocol run (i.e., stop waiting for message (10)).

We now make a few additional comments regarding possible alternatives. Apparently $f(R_S)$ could be removed from message (10), as verification of P requires use of K_{MS}, thereby indicating freshness of the current protocol run to S; however, verification of P would generally require a database access (where userid and password pairs are stored). Using $f(R_S)$ in message (10) enables S to determine freshness of the current protocol run directly from this message; thus the preference to retain $f(R_S)$ in message (10). Note that the use of $f(R_M)$ in message (11) is intended to allow verification by M that S knows K_{MS}; thus alternately $f(R_M)$ might be replaced by any constant

in this message. Also, while calculating $f(R_S)$ and $f(R_M)$, $f(\cdot)$ can be as simple as an identity function instead of a cryptographically secure hash function.

3.2 Unsuccessful Attacks Against MP-Auth

We list several potential attacks against MP-Auth, and discuss how MP-Auth prevents them. We also discuss some MP-Auth steps in greater detail, and further motivate various protocol components/steps.

a) Remote desktop attacks. A malicious browser B can collect message (4) and then deny access to U. B can use message (4) to login to S, and provide an attacker a remote desktop, e.g., a Virtual Network Computing (VNC) terminal, in real-time to the user PC. However, this attack will be detected by the transaction integrity confirmation step of MP-Auth.

b) Session hijacking attacks. In a session hijacking attack, malware may take control of a user session after the user successfully establishes a session with the legitimate server; e.g., B may leak K_{BS} to malware. The malware may actively alter user transactions, or perform unauthorized transactions without immediately being noticed by the user. However, such attacks will be detected by the transaction integrity confirmation step of MP-Auth.

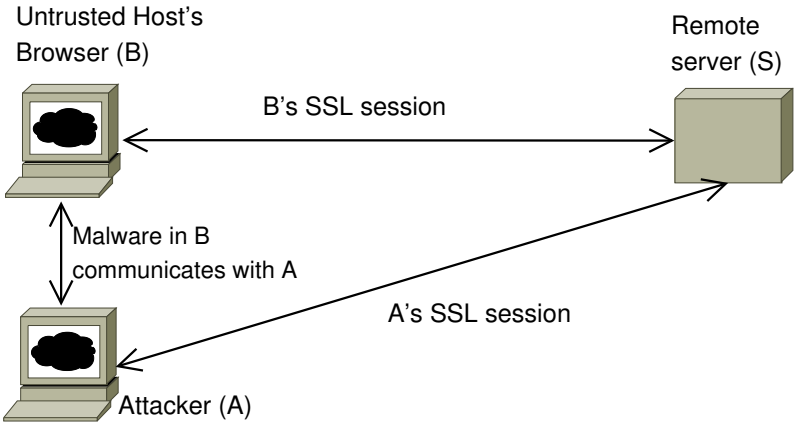

Figure 2: Setup for a parallel session attack

c) Parallel session attacks. In a parallel session attack [17], messages from one protocol run are used to form another successful run by running two or more protocol instances simultaneously. Generally a parallel session attack may effectively be prevented through the proper chaining of protocol messages. However, in MP-Auth, there is no authentication between M and B, making such an attack possible even when protocol messages are linked correctly. An attack against MP-Auth is the following (see Fig. 2). When U launches B to visit S's site, malware from U's PC notifies a remote attacker A. A starts another session with S as U, and gets message (1) from S, which the attacker relays to U's PC. The malware on U's PC drops the message (1) intended for U when B attempts to send the message to M, and forwards the attacker's message to M instead. The malware then relays back U's response (i.e., from M) to A. Now A can login as U for the current session, although A is unable to learn P. However, the transaction integrity confirmation step in MP-Auth makes such parallel session attacks view-only.

3.3 Remaining Attacks Against MP-Auth

Although MP-Auth can protect user passwords from malware installed on a PC or phishing websites, here we discuss some other possible attacks against MP-Auth which, if successful, may expose a user's plaintext password.

a) Mobile malware. We have stated the requirement that the personal (mobile) device be trusted. An attack could be launched if attackers can compromise mobile devices, e.g., by installing a (secret) keylogger. Malware in mobile networks is increasing as high-end cellphones (smart-phones) contain millions of lines of code. For example, a Sept. 2006 study [26] reported that the number of known malware programs targeting mobile devices is nearly 162; in comparison, the number of such programs has reportedly crossed the one million mark in the PC world in 2007 [13]. Worms such as Cabir [19] are designed to spread in smart-phones by exploiting vulnerabilities in embedded operating systems. Regular cellphones which are capable of running J2ME MIDlets have also been targeted, e.g., by

the RedBrowser Trojan [20]. However, currently cellphones remain more trustworthy than PCs, thus motivating our proposal.

The number of applications available for cellphones is rapidly increasing as evident from the popularity of application stores for iPhone, Android, BlackBerry and Symbian devices. Perhaps due to the closed (or less open, compared to PC) nature of software distribution in these platforms, malware threat is still limited. In the future, as mobile devices increasingly contain much more software, the requirement of trustworthy cellphones may become more problematic, and their use for sensitive purposes such as online banking may make them a more attractive target. Limited functionality devices (with less software, implying more trustworthy, see e.g., Laurie and Singer [37], ZTIC [69]) may then provide an option for use with MP-Auth. Even if MP-Auth is implemented in such a special-purpose or lower functionality device (e.g., an EMV CAP reader[5]), the device can hold several public keys for different services; in contrast, users may require a separate passcode generator for each service they want to access securely in standard two-factor authentication proposals. Another possibility of restricting mobile malware may be the use of ARM's TrustZone[6] security technology (allowing e.g., secure PIN/password entry, DRM) as available in certain ARM processors. A microkernel-based embedded hypervisor such as OKL4[7] which allows running multiple mobile operating systems (as virtual machines) on the same device can also be used; users can choose any popular OS with most features for their regular use, and switch to a custom OS (e.g., provided by a bank) for secure applications. Additionally, industry standards such as Trusted Computing Group's (TCG's) specification for trusted mobile module [47] may positively influence secure applications in mobile devices.

In version 9 of the Symbian OS (a widely used cellphone OS), Symbian has introduced *capabilities* and *data caging* [61]. A capability allows access to a set of APIs for an application, which is managed through certification, e.g., Symbian Signed.[8] About 60% of APIs are available to all applications without any capabilities. Some capabilities are granted at installation time by a user. Some sensitive APIs (e.g., ReadDeviceData, TrustedUI) are granted only after passing Symbian Signed testing. Capabilities such as DRM must be granted by the device manufacturer, e.g., Nokia. Controlled capabilities may restrict functionality of unauthorized applications (or malware). Access to the file system for applications and users is restricted through data caging. Caging enforces data privacy so that an application can access only its private directories and directories marked as 'open'.

Enforcement of capabilities and data caging is done by Symbian Trusted Computing Base (TCB). TCB is a collection of software including the kernel, file system, and software installer. However, TCB will become an attractive attack target, and it may contain bugs in itself. Secure hardware, e.g., Trusted Platform Module (TPM) may help achieve goals of Symbian Platform Security. Similar or more advanced security features are available in other popular mobile phone operating systems, e.g., iPhone, Android [18].

Anti-virus software for mobile platforms (e.g., Trend Micro) may also help maintain trustworthiness of cellphones. Malware targeting mobile phones is still limited, and leveraging the experience of working to secure traditional PC platforms may help us achieve a relatively secure mobile computing environment. However, considering the current state of mobile phone security, MP-Auth would perform better on devices whose software upgrade is tightly controlled (e.g., only allowing applications which are digitally signed by a trustworthy vendor).

b) Common-password attacks. Users often use the same password for different websites. To exploit such behavior, in a common-password attack, attackers may break into a low-security website to retrieve userid/password pairs, and then try those in financially critical websites, e.g., for online banking. MP-Auth itself does not address the common-password problem (but see e.g., PwdHash [58]).

c) Social engineering. Some forms of social engineering remain a challenge to MP-Auth (and apparently, other authentication schemes using a mobile device). For example, malware might prompt a user to enter the password directly into an untrusted PC, even though MP-Auth requires users to enter passwords only into a cellphone. In a "mixed" phishing attack, emails are sent instructing users to call a phone number which delivers, by automated voice response, a message that mimics the target bank's own system, and asks callers for account number and PIN. Fraudsters may also exploit transaction integrity confirmation using similar payee names, e.g., Be11 Canada instead of Bell Canada, or modifying a transaction amount e.g., from $100 to $1000. User habit or user instruction may provide limited protection against these. However, we argue that phishing attacks against transaction confirmation

[5]EMV is a commercial standard (Europay, MasterCard and Visa) for interoperation of chip cards. EMV CAP (Chip Authenticator Program) is a two-factor authentication system for bank customers with chip cards where a card is used to generate a one-time password; see `https://emvcap.com`. A chip card is inserted into a small CAP reader (which includes a small display, keypad and a low-end processor), and using the PIN associated with the card, a user can generate one-time passwords, respond to a server's challenge and MAC over transaction data.

[6]`http://www.arm.com/products/security/trustzone/`

[7]`http://www.ok-labs.com/products/okl4-microvisor`

[8]`www.symbiansigned.com`

may be more easily detected than those attacks against financial sites; for example, users may better understand the consequences of "pay $1000 to party X" than the security cues of a given site (e.g., detecting correct URLs, and comprehending SSL site certificates).

d) Private key disclosure. It would be disastrous if the private key of a bank is compromised. This would require, e.g., that the bank generate and distribute a new public key. However, this threat also exists for currently deployed SSL (server site) certificates, and root keys present in current browsers. If a user has multiple bank accounts that use MP-Auth, compromising one of those bank private keys may expose passwords for other accounts. The attack[9] may work in the following way. Assume a user has accounts in banks $S1$ and $S2$ with server IDs ID_{S1} and ID_{S2}, and the private key for $S1$ has been compromised. The user goes to $S2$'s website for online banking. Malware in the user's PC forwards ID_{S1} to the cellphone while displaying $S2$'s website on the PC. If the user inputs the userid and password for her $S2$ account without carefully checking the displayed server ID on her cellphone, the attacker can now access the user's password for $S2$ (using $S1$'s private key). Displaying a distinct image of the requesting server on the cellphone may reduce such risks.

e) Shoulder surfing attacks. A nearby attacker may observe (shoulder surf) while a user enters a password to a mobile device. Video recorders or cellphones with a video recording feature can also easily record user passwords/PINs in a public location, e.g., in an ATM booth. MP-Auth does not stop such attackers. Methods resilient against shoulder surfing have been proposed (e.g., [59]), and may be integrated with MP-Auth, although their practical viability remains an open question.

f) Online password guessing. Since MP-Auth assumes passwords as the only shared secret between a user and a server, online password guessing attacks can be launched against MP-Auth. Current techniques, e.g., locking online access to an account after a certain number of failed attempts, or CAPTCHA challenges [55, 67] can be used to restrict such attacks. Alternatively, tools for improving entropy of user-chosen passwords (e.g., ObPwd [39]) may also be used.

4 Usability and Deployment

In this section, we discuss usability and deployment issues related to MP-Auth. Usability is a great concern for any protocol supposed to be used by general users, e.g., for Internet banking and ATM transactions. In MP-Auth, users must connect a cellphone to a client PC. This step is more user-friendly when the connection is wireless, e.g., Bluetooth, than wire-line. Then the user browses to a bank website, and enters into the cellphone the userid and password for the site (step 5 in MP-Auth, see Section 2). We also assume that typing a userid and password on a cellphone keypad is acceptable in terms of usability, as many users are accustomed to type SMS messages or have been trained by BlackBerry/Treo experience (especially, with devices featuring a full QWERTY keypad). However, verification of transactions may be challenging to some users. We have not conducted any user study to this end.

During authentication the cryptographic operations a cellphone is required to perform in MP-Auth include: one public key encryption, one symmetric encryption and one decryption, one random number generation, and three cryptographic hash operations. The most expensive is the one public key encryption, which is a relatively cheap RSA encryption with short public exponent in our application; see Section 5 for concrete results.

For authentication in MP-Auth, a bank server performs the following operations: one private key decryption, one symmetric key encryption and one decryption, three cryptographic hash operations, and one random number generation. The private key decryption will mostly contribute to the increment of the server's computational cost. Verification of one-time passcodes generated by hardware tokens or SMS passcodes (as deployed in many two-factor authentication schemes) also incurs extra processing and infrastructure costs.

Banks may also hesitate in distributing software programs for a user's PC and mobile device as required by MP-Auth, possibly due to software maintenance issues. Standardization of such software APIs might enable interoperable independent tools development, and thus reduce maintenance burdens. If a specialized device like the EMV CAP reader is used for MP-Auth, banks may pre-package all require software on the device and relieve users from installing anything on a personal device. However, users and banks may still need to deal with software for communications between a PC and personal device.

We now discuss other usability and deployment aspects which may favor MP-Auth (see also Section 6).

1. MP-Auth expects the following from a user: (i) to input the password only to the personal device (i.e., not to provide any MP-Auth-enabled sites' passwords to a PC, even when phished or otherwise social-engineered

[9] An anonymous FC 2007 referee pointed us this attack.

to do so); and (ii) to verify transaction information diligently when confirming a transaction from the device. The former is required in each login session, and the later only infrequently. While users can still fail in these actions, we believe these requirements are simpler than existing practice, which expects users e.g., to keep a PC patched with all OS and other software updates, to install and run different anti-malware programs, and to check visual (SSL) cues on a PC browser.

2. As it appears from the current trend in online banking (see Section 6.1), users are increasingly required to use two-factor authentication (e.g., with a separate device such as a SecurID passcode generator) for login. Hence using an existing mobile device for online banking relieves users from carrying an extra device. Also, a user might otherwise require multiple hardware tokens (e.g., SecurID) for accessing different online accounts (from different banks).

3. MP-Auth offers cost efficiency for banks – avoiding the cost of providing users with hardware tokens (as well as the token maintenance cost). The software modification at the server-end is relatively minor; available SSL infrastructure is used with only three extra messages (between a browser and server) beyond SSL. MP-Auth is also compatible with the common SSL setup, i.e., a server and a client authenticate each other using a third-party-signed certificate and a user password respectively.

4. Several authentication schemes involving a mobile device store long-term secrets on the device. Losing such a device may pose substantial risk to users. In contrast, losing a user's cellphone is inconsequential to MP-Auth assuming no secret (e.g., no "remembered password") is stored on the phone.

5. Public key distribution and renewal challenges usability in any PKI. Key updating is also troublesome for banks. However, key renewal is an infrequent event; we assume that users and banks can cope with this process once every two to three years. If key updates are performed through the mobile network or selected ATMs (e.g., within branch premises), the burden of key renewal is largely distributed. For comparison, hardware tokens (e.g., SecurID) must be replaced approximately every two to five years.

6. One usability problem of MP-Auth is that users must deal with two devices (a trusted device and a PC) for online banking. Since usability of smartphones is increasing with the adoption of a full QWERTY keypad and a relatively large (e.g., 320 x 240 pixel) color screen, it would be better if MP-Auth could be used directly from such a device (i.e., without requiring a PC). However, we do not recommend such an integration as it may be vulnerable to phishing attacks when a phishing website mimics MP-Auth's user-interface for password input.

Although we have not tested MP-Auth for usability, the above suggests that compared to available two-factor authentication methods (see Section 6.1), MP-Auth may be as usable or better. However, we hesitate to make strong statements without usability tests (cf. [11]).

5 Implementation and Performance

We developed a prototype of the main authentication and session key establishment parts of MP-Auth to evaluate its performance. The prototype consists of a web server, a Firefox Extension, a desktop client, and a MIDlet on the cellphone. We set up a test web server (bank), and used PHP OpenSSL functions and the *mcrypt* module for the server-side cryptographic operations. The Firefox Extension communicates between the web server and desktop client. The desktop client forwards messages to and from the cellphone over Bluetooth. We did not have to modify the web server or Firefox browser for MP-Auth besides adding PHP scripts to the login page (note that Phoolproof [53] requires browser modifications). We used the BlueZ Bluetooth protocol stack for Linux, and Rococosoft's Impronto Developer Kit for Java. We developed a MIDlet – a Java application for Java 2 Micro Edition (J2ME), based on the Mobile Information Device Profile (MIDP) specification – for a Nokia E62 phone (commercially available circa Sept. 2006, running Symbian Series 60 r3 on a Texas Instruments processor at 235 MHz). For cryptographic operations on the MIDlet, we used the Bouncy Castle Lightweight Crypto API.

To measure login performance, we used MP-Auth for over 200 successful logins, and recorded the required login time, i.e., the time to complete steps 1 through 9 in (see Section 2; excluding userid and password input in step 5). We carried out similar tests for regular SSL logins. The results are summarized in Table 2. Table 3 summarizes other implementation choices. Although regular SSL login is almost eight times faster than MP-Auth, on average, it takes less than a second for MP-Auth login. This added delay may be acceptable, given that entering a userid and password takes substantial additional time. Source code for the prototype can be made available.

	Avg. Time (s)	[Min, Max] (s)
MP-Auth	0.62	[0.34, 2.28]
Regular SSL	0.08	[0.06, 0.22]

Table 2: Performance comparison between MP-Auth and regular SSL login excluding user input time

Public key encryption	RSA 1024-bit
Symmetric encryption	AES-128 (CBC)
Hash function	SHA-1 (160-bit)
Source of randomness	`/dev/urandom`, `SecureRandom`

Table 3: Cryptosystems and parameters for MP-Auth

6 Survey of Related Work

In this section, we summarize and provide extended discussion of related online authentication methods used in practice or proposed in the literature, and compare MP-Auth with these techniques.

6.1 Online Authentication Methods

We first discuss several online authentication methods commonly used (or proposed for use) by banks, and briefly discuss their security.

a) Password-only authentication. Most bank websites authenticate customers using only a password over an SSL connection. This is susceptible to keyloggers and phishing. Banks' reliance on SSL certificates does not stop attackers. Attackers have used certificates – both self-signed, and real third-party signed certificates for sound-alike domains, e.g., `visa-secure.com` – to display the SSL lock on phishing websites. In 2005, over 450 phishing websites were reported to deploy SSL [49]. A trojan (with rootkit capabilities) has been reported [24] to inject a spoofed HTML form (from the local PC) inside a SSL-protected bank website for collecting banking and identity information; the form appears to be served by the real bank site, and the browser displays the correct SSL site certificate when verified by the user. Also the bank site remains uncompromised in this attack.

In a cross-site/cross-frame scripting attack, vulnerable website software is exploited to display malicious (phishing) contents within the website, making such attacks almost transparent to users. Past vulnerable websites include Charter One Bank, MasterCard, Barclays and Natwest.[10] In a March 2006 phishing attack, attackers broke into web servers of three Florida-based banks, and redirected the banks' customers to phishing websites.[11] In another high-profile phishing attack, attackers manipulated a U.S. government website to forward users to phishing websites.[12]

Reliance on SSL itself also leads to problems. For example, only one in 300 customers of a New Zealand bank [49] chose to abandon the SSL session upon a browser warning indicating an expired SSL site certificate; the bank accidentally allowed a certificate to expire for a period of 12 hours. A user study by Dhamija et al. [16] also notes that standard (visual) security indicators on websites are ineffective for a significant portion of users; over 90% participants were fooled by phishing websites in the study.

As front-end (client-side) phishing solutions are failing in many instances, some banks are putting more resources at back-end fraud detection to counter phishing threats. For example, HSBC in Brazil uses the PhishingNet[13] back-end solution. PhishingNet uses user machine identification, and monitors online account activities, without requiring any user registration or software downloads. Such a solution is almost transparent to end-users, and may help detect fraudulent transactions. The RSA Adaptive Authentication for Web[14] also provides similar back-end fraud detection capabilities. However, in case of session hijacking attacks when fraudulent transactions are performed from a user's own machine, back-end solutions may not help much.

The above suggests password-only web authentication over SSL is inadequate in today's Internet environment. This is motivating financial organizations towards two-factor authentication methods.

b) Two-factor authentication. Traditionally, authentication schemes have relied on one or more of three factors: something a user *knows* (e.g., a password), something a user *has* (e.g., a bank card), and something a user *is* (e.g., biometric characteristics). Properly designed authentication schemes that depend on more than one factor are more reliable than single-factor schemes. Note that the authentication scheme used in ATMs through a bank card and PIN is two-factor; but, an online banking authentication scheme that requires a user's bank card number (not necessarily the card itself) and a password is single-factor, i.e., both are something known. As a step toward multi-

[10] `http://www.spamfo.co.uk/component/option,com_content/task,view/id,83/Itemid,2/`
[11] `http://news.netcraft.com/archives/2006/03/27/phishers_hack_bank_sites_redirect_customers.html`
[12] `http://www.eweek.com/c/a/Security/Tax-Scam-Preys-on-RefundHungry-Public-with-Real-Gov-Site/`
[13] `http://www.the41st.com/products.asp`
[14] `http://www.rsa.com/node.aspx?id=3018`

factor authentication, banks are providing users with devices like one-time password generators, to use along with passwords for online banking, thus making the authentication scheme rely on two independent factors. Examples of two-factor authentication in practice are given below.

1. Several European banks attempt to secure online banking through e.g., passcode generators.
2. U.S. federal regulators have provided guidelines for banks to implement two-factor authentication by the end of 2006 for online banking [21].
3. The Association of Payment and Clearing Systems (APACS) in the U.K. is developing a standard[15] for online and telephone banking authentication. Most major U.K. banks and credit-card companies are members of APACS. The standard provides users a device to generate one-time passwords using a chip card and PIN. The one-time password is used along with a user's regular password.

Two-factor web authentication methods may make the collection of passwords less useful to attackers and thus help restrict phishing attacks. However, these methods raise deployment and usability issues, e.g., cost of the token, requirement to carry the token. Also malware on a client PC can record the device-generated secret (which a user inputs directly to a browser), and log on to the bank website before the actual user. This is recognized as a classic man-in-the-middle (MITM) attack. Apparently showing a pre-selected user image or phrase on the login page, or "device fingerprinting" (information specific to a user PC, e.g., client browser, OS, CPU type, screen resolution) are considered as a second factor by several U.S. banks; see O'Connor [51] for how easily these 'second' factors can be defeated by traditional phishing/MITM attacks.

In an interesting real attack [64] against a one-time password scheme implemented by a Finnish bank, the bank provided users a scratch sheet containing a certain number of one-time passwords. By setting up several phishing sites, attackers persuaded users to give out a sequence of one-time passwords in addition to their regular passwords. This attack is made more difficult if one-time passwords expire after a short while (e.g., 30 to 60 seconds in SecurID); then the collected one-time passwords must be used within a brief period of time from a user's login attempt. A July 2006 phishing attack [50], attackers collected userid, password, as well as one-time password (OTP) generated by time-based passcode generators from Citibank customers, and launched a real-time MITM attack against compromised accounts. Also, such time-based passcode generators, e.g., SecurID, typically have time synchronization problems between a client device and the server [72], and expire in 2-5 years. Other security issues of such devices (e.g., [70]) are not directly relevant to our discussion; we assume that any weaknesses could be repaired by superior algorithms or implementations overtime, albeit with the usual practical challenges, e.g., backwards compatibility.

Note that, even when a one-time password is used along with a user's (long-term) regular password, gathering long-term passwords may be still be of offline use to an attacker. For example, if flaws are found in a one-time key generator algorithm (e.g., differential adaptive chosen plaintext attack [9]) by which attackers can generate one-time keys without getting hold of the hardware token, keylogging attacks to collect user passwords appear very useful.

Instead of gathering passwords, attackers can simply steal money from user accounts in real-time, immediately after a user completes authentication [32, 62]. One Trojan program [23] even alters banking statements when displayed in an infected PC browser to avoid detection by the user. Therefore, transaction security becomes critical to restrict such attacks.

c) Transaction security and complimentary mechanisms. To protect important transactions, and make users better able to detect break-ins to their accounts, some banks have deployed security techniques which are generally complementary to authentication schemes. Examples include:

1. Two New Zealand banks require online users to enter a secret from a cellphone (sent as an SMS message to the phone) for transfers over $2500 from one account to another [63].
2. Customers of the Commonwealth Bank of Australia[16] must answer (pre-established) identification questions when performing sensitive transactions. Email alerts are sent to users to confirm when users' personal details have been changed, or modifications to user accounts are made.
3. Bank of America uses SiteKey and SafePass[17] to strengthen online authentication and transaction authorization. If a user PC is recognized by the bank, a secret pre-shared SiteKey picture is displayed; upon successful verification of the SiteKey picture, the user enters her password. A confirmation question is asked if the user PC is not recognized, and the SiteKey picture is displayed when the user answers the question correctly. The SiteKey picture provides evidence that the user is entering her password to the correct website. When registered with SafePass, users get a six-digit OTP through an SMS message which is used for authorizing critical transactions; OTPs can also be generated using a wallet-sized card that users can buy from the bank.

[15] http://www.chipandpin.co.uk/

[16] http://www.commbank.com.au/Netbank/faq/security.asp

[17] http://www.bankofamerica.com/privacy/index.cfm?template=learn_about_safepass

In principle, the above mechanisms (as well as MP-Auth's transaction integrity confirmation) are similar to integrity cross-checks by a second channel [66]. Attackers may be able to defeat some of these techniques; e.g., if a bank requires SMS verification on large transactions, attackers can commit several relatively small transactions (e.g., $10 instead of $1000) to avoid the verification step. In one instance [30], cellphones are reprogrammed with a targeted user's phone number; this enables attackers to receive SMS messages including mTANs (mobile transaction authentication numbers) sent to the user's mobile phone number by her bank as required for performing important transactions in certain European countries. Also, SMS verification requires access to cellphone networks which is a problem when a phone network is not available (e.g., while traveling).

d) Using a cellphone alone for important Internet services. One proposed solution to keyloggers is to perform all critical work through a cellphone browser, or through a PDA. However, a combination of the following usability and security issues may restrict such proposals being widely deployed.

1. The display area of a cellphone/PDA is much smaller than a PC, limiting usability for web browsing.
2. Users may still reveal passwords to phishing sites controlled by malicious parties (through e.g., domain name hijacking [29], Kaminsky DNS-flaw [34]). Thus even a trusted browser in a trusted device may not stop phishing attacks; i.e., such a setup may allow a 'secure' pipe directly to phishing sites.
3. Some mobile browsers, e.g., Opera Mini,[18] use proxy servers (called transcoders) to provide a faster web experience from a smart-phone; transcoders retrieve a web page on behalf of a user, and reformat the content to fit in the mobile browser. As a consequence, SSL connections from a mobile browser to a site cannot provide end-to-end encryption; the transcoders can learn all user credentials. This is problematic in several ways: a transcoder-provider can be malicious or compromised, and users may be unknowingly violating banking agreements which mandate not to share passwords with others.
4. In many parts of the world, airtime costs money. So Internet browsing through a mobile network remains, at least presently, far more expensive than wire-line Internet connections.

e) Comparing MP-Auth with existing online authentication methods. In contrast to two-factor authentication methods, by design MP-Auth does not provide attackers any window of opportunity when authentication messages (i.e., collected regular and one-time passwords of a user) can be replayed to login as the legitimate user and perform transactions on the user's behalf. The key observation is that, through a simple challenge-response, message (4) in MP-Auth (Section 2) effectively turns a user's long-term static password into a one-time password in such a way that long-term passwords are not revealed to phishing websites, or keyloggers on an untrusted PC. In contrast to transaction security mechanisms, MP-Auth can protect both large and small transactions as long as users diligently check integrity confirmation messages, and transactions are prudently labled for user confirmation from the device; for example, even small transactions to an unknown/unregistered party should be categorized as sensitive. Also, MP-Auth does not require text or voice communications airtime for web authentication or transaction security. (See also Section 4 for more comparison on usability and deployment issues.)

6.2 Academic Proposals

Here we summarize selected academic proposals for authentication from an untrusted PC using a mobile device. MP-Auth shares several design goals with these, and is influenced by the ideas and experiences of these past proposals. We also compare MP-Auth to these in terms of technical merits and usability.

a) Splitting trust paradigm. Abadi et al. [1] envisioned an ideal smart-card (with an independent keyboard, display, processor) as early as 1990, and designed protocols using such a device to safeguard a user's long-term secrets from a potentially malicious computer. In 1999, Balfanz and Felten [6] proposed a scheme to deliver smart-card functionality through a PalmPilot assuming the availability of user-level public key systems. They introduced the splitting trust paradigm to split an application between a small (in size and processing power) trusted device and an untrusted computer. Our work is based on such a paradigm where we provide the long-term password input through widely available cellphones, and use the untrusted computer for computationally intensive processing and display. However, we do not use any user-level PKI.

b) Phoolproof phishing prevention. Parno, Kuo and Perrig [53] proposed a cellphone-based technique to protect users against phishing with less reliance on users making secure decisions. With the help of a pre-shared secret – established using an out-of-band channel, e.g., postal mail – a user sets up an account at the intended service's website. The user's cellphone generates a key pair $\{K_U, K_U^{-1}\}$, and sends the public key to the server. The user's private key and server certificate are stored on a cellphone for logins afterward. During login (see Fig. 3), a user

[18] `http://www.opera.com/mini/help/faq/#security`

provides userid and password to a website on a browser (as usual), while in the background, the browser and server authenticate (using SSL mutual authentication) through the pre-established client/server public keys in an SSL session; the browser receives the client public key from the cellphone. (See also the Personal Transaction Protocol (PTP) [46] for a similar approach from leading mobile phone manufacturers.)

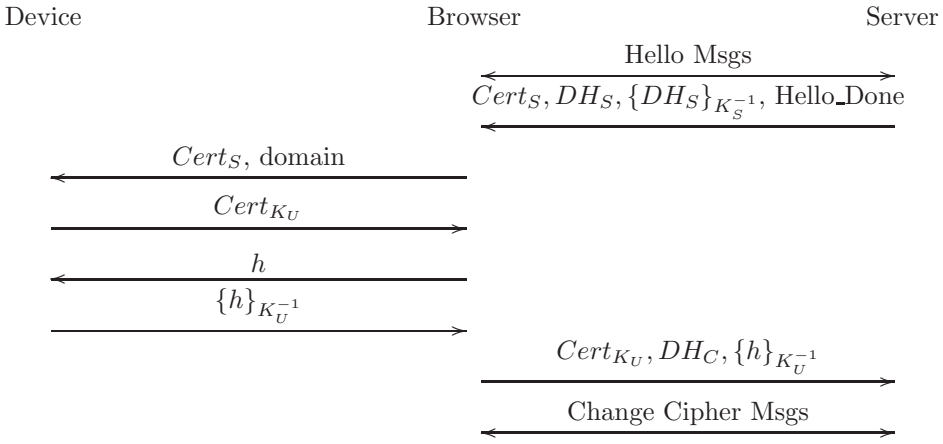

Figure 3: Phoolproof login process (adapted from [53]

In Figure 3, DH_S, DH_C represent the Diffie-Hellman public key parameters for the server and client browser respectively, and h is a secure hash of all previous SSL handshake messages of the current session. As noted [53], attackers may hijack account setup or (user) public key re-establishment. Phoolproof assumes that users can correctly identify websites at which they want to set up an account. Public key creation in Phoolproof happens in the background and is almost transparent to users. However, users must revoke public/private key pairs in case of lost or malfunctioning cellphones, or a replacement of older cellphone models. Expecting non-technical users (e.g., typical bank customers) to understand concepts of revocation and renewal of public keys may not be practical yet.

It is also assumed in Phoolproof that the (Bluetooth) channel between a browser and cellphone is secure. Seeing-is-believing (SiB) [42] techniques are proposed to secure local Bluetooth channels, requiring users to take snapshots using a camera-phone, an thus increasing complexity to users. If malware on a PC can replace h (when the browser attempts to send h to the cellphone) with an h value from an attacker, the attacker can login as the user (recall parallel session attacks in Section 3.2). Also, Phoolproof is not designed to provide protection against session hijacking attacks, which are becoming more common, and easy to develop and deploy [32]. MP-Auth achieves such protection at the (human interaction) cost of transaction integrity confirmation.

c) Bumpy: Safe passage for passwords. McCune et al. proposed Bumpy [44] to enable safe forwarding of user-specified (e.g., via a secure attention sequence @@) sensitive information including passwords from the user to a server, bypassing untrustworthy legacy OS and other applications. Bumpy requires a trusted device for displaying critical information (e.g., URL, favicon of the input-receiving site), input-encrypting keyboard and mouse, and a proposed system called Flicker [41]. Using TPM and the *late launch* feature available in certain recent CPUs, Flicker provides hardware-supported isolation for security-sensitive code from the OS and other applications. Flicker is used for obtaining sensitive user input which is released to the legacy OS/browser after being encrypted for the receiving server. Bumpy's goal is to ensure a safe input mechanism in an untrusted platform, but not to protect user secrets from phishing sites. A long or look-a-like URL and the target site's favicon when used in a phishing attack may easily fool users into divulging their secrets to unwanted sites.

d) Zone Trusted Information Channel (ZTIC). ZTIC [69] has been designed as a special-purpose USB device primarily to expose and confirm online transactions (genuine or otherwise), initiated from an untrusted PC. The device is equipped with a small display and two buttons (OK, Cancel), stores user credentials (e.g., bank-issued user-level X.509 certificates, as in Phoolproof [53]), and hosts a TLS engine, proxy server and HTTP parser. The server and device establish a secure channel using SSL mutual authentication, and the device acts as a proxy and parses all interactions between the browser and server. When a critical transaction is detected, it is displayed on the device and performed only when confirmed from the device (i.e., by pressing the OK button). The HTTP parser and bank server must be kept in sync to ensure proper extraction and display of transaction detail. The device can store multiple user/server credentials and thus be used to access different ZTIC-enabled services. ZTIC does not aim to protect user passwords; instead it relies on user-level certificates and PKI.

e) Bump in the Ether. Bump in the Ether (BitE) [43] proxies sensitive user input to a particular application via a trusted mobile device, bypassing the Linux X-windowing system. Users receive verifiable evidence regarding the integrity of the host kernel and whether the intended user application has been loaded. Only the target application receives user input from the mobile device through a user-verifiable trusted tunnel (between the device and application). BitE assumes the OS kernel is trustworthy, and the BIOS and OS are TPM-enabled and perform integrity check of code loaded for execution. BitE requires a user's mobile device to be cryptographically paired with the OS kernel. For establishing a trusted tunnel between the user device and an application, (symmetric) cryptographic keys for each BitE-aware application must be shared beforehand (e.g., during application installation/registration). Keystrokes from the trusted device is then sent encrypted from the device to the target application.

BitE can protect user input against user-space malware. However, BitE does not protect user inputs from a phishing website, or compromised (e.g., Trojaned) user applications. BitE also stores cryptographic keys to the mobile device, which are subject to compromise if the device is lost or left unattended (if not protected otherwise, e.g., through TPM). Session hijacking is also not addressed by BitE.

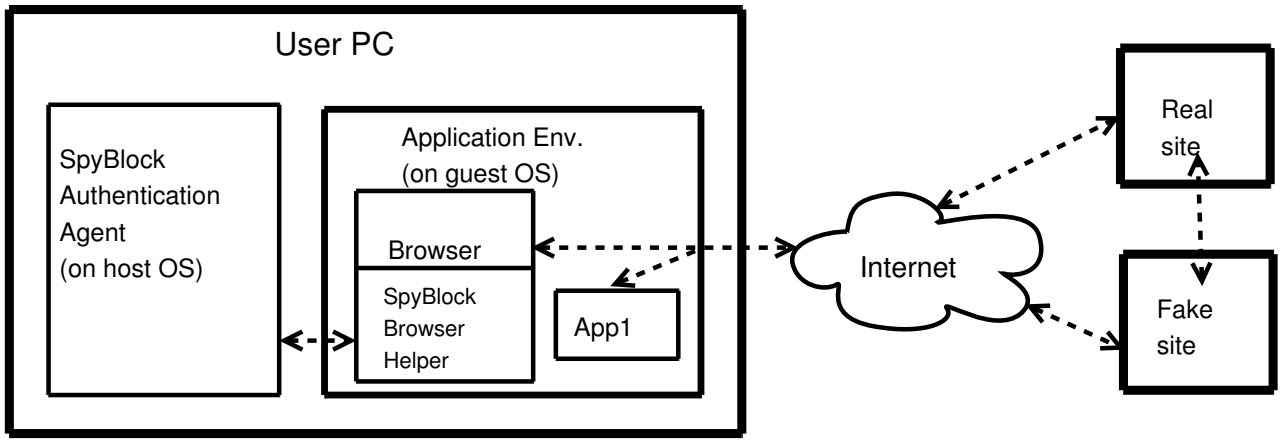

Figure 4: SpyBlock setup (adapted from [31])

f) SpyBlock. Jackson, Boneh and Mitchell propose SpyBlock [31] (see also [32]) to provide spyware-resistant web authentication using a virtual machine monitor (VMM). The SpyBlock authentication agent runs on a host OS (assumed to be trusted), and user applications including a web browser with a SpyBlock browser helper run inside a guest VM (assumed to be untrusted) on the trusted host OS. See Figure 4.

A user authenticates to a website with the help of the SpyBlock authentication agent. The site password is given only to the authentication agent which supports several authentication techniques, e.g., password hashing, strong password authentication, transaction integrity confirmation. The authentication agent provides a trusted path to the user through a pre-shared secret picture.

SpyBlock does not require an additional hardware device (e.g., a cellphone). However, a VMM must be installed. Also, users must know when they are communicating with the authentication agent; user interface design in such a setting appears quite challenging. Another assumption in SpyBlock is that the host OS is trusted. In reality, maintaining trustworthiness of any current consumer OS is very difficult (which is in part why secure web authentication is so complex).

g) Three-party secure remote terminal protocol. Oprea et al. [52] proposed a three-party protocol (see Fig. 5) to provide secure access to a home computer from an untrusted public terminal. A trusted device (e.g., PDA) is used to delegate temporary credentials of a user to an untrusted public computer, without revealing any long-term secret to the untrusted terminal. Two SSL connections are established in the protocol: one from the trusted PDA and another from the untrusted terminal to the home PC using a modified Virtual Network Computing (VNC) system. The PDA authenticates normally (using a password) to the home PC, and forwards temporary secret keys to the untrusted terminal. A user can control how much information from the home PC is displayed to the untrusted PC. Control messages to the home VNC, e.g., mouse and keyboard events, are only sent from the PDA.

This protocol safeguards user passwords only when users access a PC (or application) that they control, e.g., a home PC. Also, the trusted device must have SSL capabilities, and is required to maintain a separate SSL channel from the PDA to the home PC.

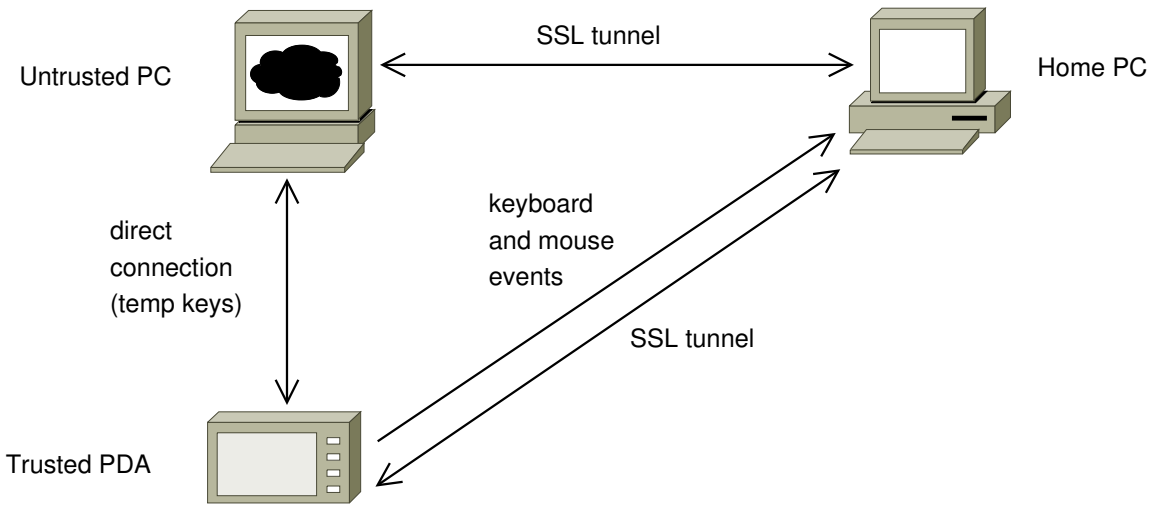

Figure 5: Three-party VNC protocol (adapted from [52])

h) Camera-based authentication. Clarke et al. [12] proposed a technique using camera-phones for authenticating visual information (forwarded by a trusted service) in an untrusted PC. This method verifies message authenticity and integrity for an entire user session; i.e., it authenticates the content displayed on a PC screen for every web page or only critical pages in a user session. A small area on the bottom of a PC screen is used to transmit security parameters (e.g., a nonce, a one-time password, or a MAC) as an image, with a strip of random-looking data. Figure 6 outlines the proposed protocol.

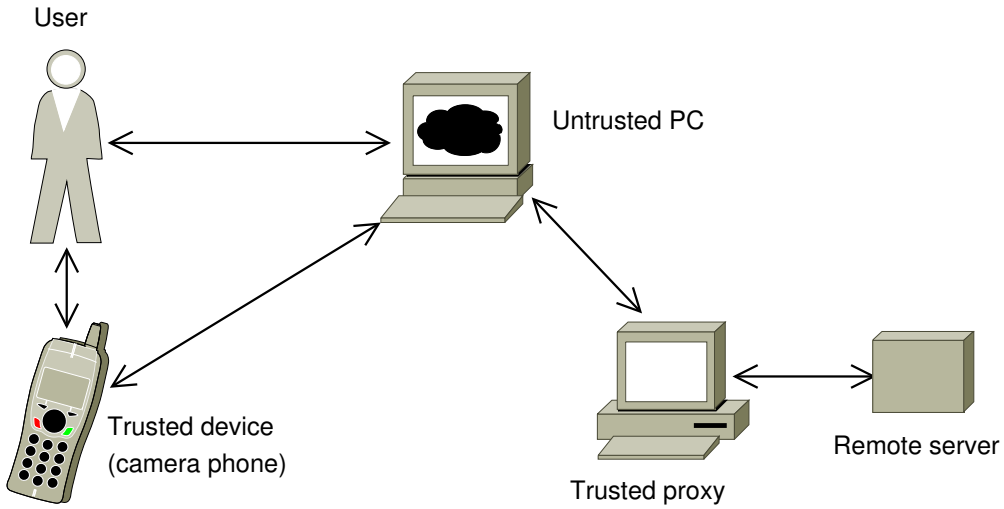

Figure 6: Camera-based authentication

To access a service from the Internet through an untrusted PC, this scheme requires a trusted proxy. A user's long-term keys are stored on the camera-phone, protected by a PIN or biometric measurement. With a stolen phone, an attacker may successfully impersonate the user or retrieve the stored long-term keys from the phone. Camera-based authentication also creates a much different user experience: users are expected to take snapshots and visually verify (cross-check) images in terms of colors and shades. A calibration phase may also be required to construct a mapping between PC screen pixels and camera pixels (in one implementation, reported to take about 10 seconds). It attempts to authenticate content of a visual display, which is apparently useful in a sense that we can verify what is displayed on the screen.

i) Secure web authentication with cellphones. Figure 7 shows the secure web authentication proposed by Wu et al. [71]. User credentials (userid, password, mobile number etc.) are stored on a trusted proxy server. The

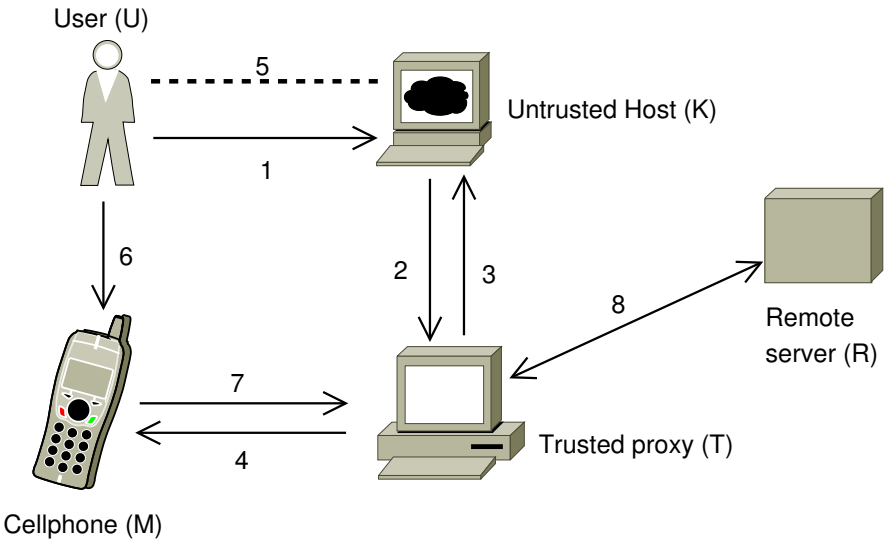

Figure 7: Web authentication with a cellphone

protocol involves the following steps (see Fig. 7 for symbol definitions).

1. U launches a web browser at K, and goes to T's site.
2. U types her userid and K sends it to T.
3. T chooses a random session name, and sends it to K.
4. T sends this session name to M as an SMS message.
5. U checks the displayed session name at K.
6. U verifies the session name at M.
7. If session names match, the user accepts the session.
8. If U accepts the session, then T uses U's stored credentials to login to R, and works as a web proxy.

This protocol requires a trusted proxy, which if compromised, may readily expose user credentials to attackers. A well-behaved proxy may also be tricked to access a service on behalf of a user. Hence the proxy may become a prime target of attacks. Also, losing the cellphone is problematic, as anyone can access the trusted proxy using the phone, at least temporarily. Delegate [33] is another similar trusted proxy based solution for secure website access from an untrusted PC that also provides protection against session hijacking.

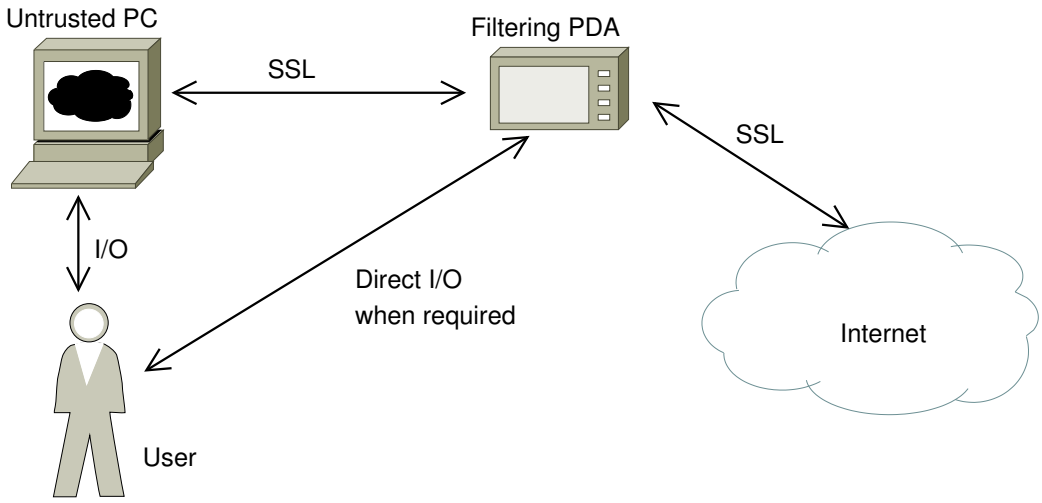

Figure 8: Guardian setup (adapted from [40])

j) Guardian privacy control framework. The Guardian [40] framework has been designed with an elaborate threat model in mind. Its focus is to protect privacy of a mobile user,[19] including securing long-term user passwords and protecting sensitive information, e.g., personal data from being recorded (to prevent identity profiling). Guardian works as a personal firewall but placed on a trusted PDA. In effect, the PDA acts as a portable privacy proxy. See Fig. 8.

Guardian keeps passwords and other privacy sensitive information out of the reach of keyloggers and other malware installed on an untrusted PC. However, phishing attacks still may succeed. Guardian attempts to manage a large set of sensitive user details, e.g., PKI certificates, SSL connections, and cookies as well as real-time content filtering. Thus its implementation appears to be complex, and requires intelligent processing from the PDA.

k) Comparing MP-Auth with existing literature. Table 4 summarizes a comparison of MP-Auth with several anti-phishing proposals from the literature. An (✗) means a special requirement is needed. An empty box indicates the stated protection is not provided (first three columns) and the stated requirement is not needed (last four columns). NA denotes non-applicability. (All ✓and no ✗ would be optimal.) For example, Phoolproof [53] provides protection against phishing and keylogging, but it is vulnerable to session hijacking; it requires a malware-free mobile and stores long-term secrets on the mobile, but does not require a trusted proxy or trusted PC OS. We acknowledge that although this table may provide useful high-level overview, it is not an apple-to-apple comparison. Several solutions listed here require a trusted proxy, thus introduce an extra deployment burden, and present an attractive target to determined attackers. Also, fraudsters may increasingly target mobile devices if long-term secrets are stored on them.

	Protection against			Requirement			
	Session-hijacking	Phishing	Key-logging	Trusted proxy	On-device secret	Trusted PC OS	Malware-free mobile
MP-Auth	✓	✓	✓				✗
Phoolproof [53]		✓	✓		✗		✗
Bumpy [44]			✓				✗
ZTIC [69]	✓	✓	✓		✗		✗
BitE [43]			✓		✗	✗	✗
SpyBlock [31]	✓	✓	✓		NA	✗	
Three-party [52]	NA	NA	✓		✗		✗
Camera-based [12]	✓	✓	✓	✗	✗		✗
Web-Auth [71]		✓	✓	✗	✗		✗
Guardian [40]			✓		✗		✗

Table 4: Comparing MP-Auth with existing academic proposals

7 Concluding Remarks

We propose MP-Auth, a protocol for web authentication which is resilient to keyloggers (and other malware including rootkits), phishing websites, and session hijacking. Recently, many new small-scale, little-known malware instances have been observed that install malicious software launching keylogging and phishing attacks; these are in contrast to large-scale, high-profile worms like Slammer. One reason for this trend might be the fact that attackers are increasingly targeting online financial transactions. For example, according to one report [3], 91% of all phishing sites in the third quarter of 2009 targeted online financial, payment, auction and retail services. Furthermore, such attacks are fairly easy to launch; for example, attackers can gain access to a user's bank account simply by installing (remotely) a keylogger on a user PC and collecting the user's banking access information (userid and password). MP-Auth is designed to prevent such attacks. MP-Auth primarily focuses on online banking but can be used for general web authentication systems as well as at ATMs. Our requirement for a trustworthy personal device (i.e., free of malware) is important, and becomes more challenging over time, but as discussed in Section 3.3, may well remain viable. In our MP-Auth implementation, cryptographic computations and Bluetooth communications took less than a second for login (excluding the user input time), which we believe to be an acceptable delay for the added security. Despite a main objective of preventing phishing and keylogging attacks, MP-Auth as presented remains one-factor

[19] A user who uses several different public terminals to access critical online services, e.g., banking.

authentication; thus an attacker who nonetheless learns a user password can impersonate that user. Consequently, the server side of MP-Auth must be trusted to be secure both against insider attack and break-in.

Users often input reusable critical identity information to a PC other than userid/password, e.g., a passport number, social security number, driver's license number, or credit card number. Such identity credentials are short, making them feasible (albeit tedious) to enter from a cellphone keypad. In addition to protecting a user's userid/password, MP-Auth may easily be extended to protect other identity credentials from the reach of online attackers, and thereby might be of use to reduce online identity theft. Additionally, MP-Auth may be suitable for use in ATMs (automated teller machines), if an interface is provided to connect a cellphone, e.g., a wire-line or Bluetooth interface. This can be a step towards ending several types of ATM fraud. We believe that the very simple approach on which MP-Auth is based – using a cellphone or similar device to asymmetrically encrypt passwords and one-time challenges – is of independent interest for use in many other applications, e.g., traditional telephone banking directly from a cellphone, where currently PINs are commonly transmitted in-band without encryption.

We reiterate that although based on a very simple idea, MP-Auth has yet to be user-tested for usability; this is an architecture, analysis and state-of-the-art paper. We encourage the security community to pursue alternate proposals for password-based online authentication which simultaneously address phishing, keylogging and session hijacking rootkits.

Acknowledgments. We thank anonymous referees for their constructive comments which improved the presentation of this work, Bryan Parno for allowing us to access and build on source code of his Phoolproof [53] implementation, Anupam Datta for help with the PCL analysis, and Masud Khan for providing a Nokia E62 smartphone. The first author is supported in part by an NSERC CGS and ISSNet (Internetworked Systems Security Network – an NSERC Strategic Network Grant) post-doctoral fellowship. The second author is Canada Research Chair in Internet Authentication and Computer Security, and is supported in part by an NSERC Discovery Grant, the Canada Research Chairs Program, and NSERC ISSNet.

References

[1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. *Science of Computer Programming*, 21(2):91–113, Oct. 1993.

[2] M. AlZomai, B. AlFayyadh, A. Jøsang, and A. McCullag. An exprimental investigation of the usability of transaction authorization in online bank security systems. In *Australasian Information Security Conference (AISC'08)*, Wollongong, Australia, Jan. 2008.

[3] Anti-Phishing Working Group. Phishing Activity Trends Report, Q3/2009. `http://www.antiphishing.org/reports/apwg_report_Q3_2009.pdf`.

[4] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of Internet security protocols and applications. In *Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, 2005. Project website, `http://www.avispa-project.org`.

[5] K. Bailey, A. Kapadia, L. Vongsathorn, and S. W. Smith. TwoKind authentication: Protecting private information in untrustworthy environments. In *ACM Workshop on Privacy in the Electronic Society (WPES'08)*, Alexandria, VA, USA, Oct. 2008.

[6] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 1999.

[7] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *AsiaCrypt*, Kyoto, Japan, Dec. 2000.

[8] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO'93*, Santa Barbara, CA, USA, Aug. 1993.

[9] A. Biryukov, J. Lano, and B. Preneel. Cryptanalysis of the alleged SecurID hash function. In *Selected Areas in Cryptography (SAC)*, volume 3006 of *LNCS*, Ottawa, Canada, Aug. 2003.

[10] CA Virus Information Center. Win32.Grams.I, Feb. 2005.

[11] S. Chiasson, P. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *USENIX Security Symposium*, Vancouver, Canada, Aug. 2006.

[12] D. E. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. van Dijk, S. Devadas, and R. L. Rivest. The untrusted computer problem and camera-based authentication. In *Pervasive Computing*, volume 2414 of *LNCS*, Zurich, Switzerland, Aug. 2002.

[13] Computerworld.com. Malware count blows past 1M mark. News article (Apr. 8, 2008).

[14] D. Dagon, M. Antonakakis, X. Luo, C. P. Lee, W. Lee, and K. Day. Recursive dns architectures and vulnerability implications. In *Network and Distributed System Security Symposium (NDSS'09)*, San Diego, CA, USA, Feb. 2009.

[15] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol composition logic (PCL). *Electronic Notes in Theoretical Computer Science (ENTCS)*, 172:311–358, Apr. 2007.

[16] R. Dhamija, J. Tygar, and M. Hearst. Why phishing works. In *CHI*, Montreal, Canada, Apr. 2006.

[17] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[18] W. Enck, M. Ongtang, and P. McDaniel. Understanding Android security. *IEEE Security & Privacy Magazine*, 7(1):50–57, 2009.

[19] F-Secure. F-Secure virus descriptions: Cabir, June 2004.

[20] F-Secure. F-Secure trojan information pages: Redbrowser.A, Mar. 2006.

[21] Federal Financial Institutions Examination Council (FFIEC). FFIEC guidance: Authentication in an Internet banking environment, Oct. 2005. `http://www.fdic.gov/news/news/financial/2005/fil10305.html`.

[22] E. W. Felten, D. Balfanz, D. Dean, and D. S. Wallach. Web spoofing: An Internet con game. In *National Information Systems Security Conference*, Oct. 1997.

[23] Finjan Malicious Code Research Center. Cybercrime intelligence report: Cybercriminals use Trojans & money mules to rob online banking accounts. Online article (issue no. 3, 2009). `http://www.finjan.com/GetObject.aspx?ObjId=679`.

[24] Finjan Malicious Code Research Center. Web security trends report – Q3/2007. `http://www.finjan.com/GetObject.aspx?ObjId=506`.

[25] V. D. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Workshop on Fast Software Encryption (FSE'01)*, Yokohama, Japan, Apr. 2001.

[26] A. Gostev and A. Shevchenko. Kaspersky security bulletin, January - June 2006: Malicious programs for mobile devices, Sept. 2006. `http://www.viruslist.com`.

[27] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and Systems Security (TISSEC)*, 2(3):230–268, Aug. 1999.

[28] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *ACM Computer and Communications Security (CCS'05)*, Alexandria, VA, USA, Nov. 2005.

[29] ICANN Security and Stability Advisory Committee. Domain name hijacking: Incidents, threats, risks, and remedial actions, July 2005. `http://www.icann.org`.

[30] IDG News Service. Investigators replicate Nokia 1100 online banking hack. News article (May 21, 2009). `http://www.thestandard.com/news/2009/05/21/investigators-replicate-nokia-1100-online-banking-hack`.

[31] C. Jackson, D. Boneh, and J. Mitchell. Spyware resistant web authentication using virtual machines. Technical report (2006). `http://crypto.stanford.edu/spyblock`.

[32] C. Jackson, D. Boneh, and J. Mitchell. Transaction generators: Root kits for web. In *USENIX Workshop on Hot Topics in Security (HotSec'07)*, Boston, MA, USA, Aug. 2007.

[33] R. C. Jammalamadaka, T. van der Horst, S. Mehrotra, K. Seamons, and N. Venkatasuramanian. Delegate: A proxy based architecture for secure website access from an untrusted machine. In *Annual Computer Security Applications Conference (ACSAC'06)*, Miami Beach, FL, USA, Dec. 2006.

[34] D. Kaminsky. Black Ops 2008 – It's the end of the cache as we know it. In *Black Hat USA*, Las Vegas, NV, USA, Aug. 2008.

[35] D. Kuhlman, R. Moriarty, T. Braskich, S. Emeott, and M. Tripunitara. A correctness proof of a mesh security architecture. In *IEEE Computer Security Foundations Symposium (CSF)*, Pittsburgh, PA, USA, June 2008.

[36] K. Kursawe and S. Katzenbeisser. Computing under occupation. In *New Security Paradigms Workshop (NSPW'07)*, New Hampshire, USA, Sept. 2007.

[37] B. Laurie and A. Singer. Choose the red pill and the blue pill. In *New Security Paradigms Workshop (NSPW'08)*, Lake Tahoe, CA, USA, Sept. 2008.

[38] M. Mannan and P. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security (FC'07)*, volume 4886 of *LNCS*, Lowlands, Scarborough, Trinidad and Tobago, Feb. 2007.

[39] M. Mannan and P. van Oorschot. Digital objects as passwords. In *USENIX Workshop on Hot Topics in Security (HotSec'08)*, San Jose, CA, USA, July 2008.

[40] N. B. Margolin, M. K. Wright, and B. N. Levine. Guardian: A framework for privacy control in untrusted environments, June 2004. Technical Report 04-37 (University of Massachusetts, Amherst).

[41] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *The European Conference on Computer Systems (EuroSys'08)*, Glasgow, Scotland, UK, Apr. 2008.

[42] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2005.

[43] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the Ether: A framework for securing sensitive user input. In *USENIX Annual Technical Conference*, Boston, MA, USA, 2006.

[44] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *Network and Distributed System Security Symposium (NDSS'09)*, San Diego, CA, USA, Feb. 2009.

[45] Mobile Antivirus Researchers Association. Analyzing the crossover virus: The first PC to Windows handheld cross-infector, 2006. `http://www.informit.com`.

[46] Mobile electronic Transactions (MeT) Ltd. Personal Transaction Protocol Version 1.0 (Draft Specification), Jan. 2002. `http://www.mobiletransaction.org/`.

[47] Mobile Phone Work Group. TCG mobile trusted module specification. Specification version 1.0, Revision 6, June 26, 2008.

[48] A. Moshchuk, T. Bragin, S. D. Gribble, and H. Levy. A crawler-based study of spyware in the web. In *Network and Distributed System Security (NDSS'06)*, San Diego, CA, USA, Feb. 2006.

[49] Netcraft. More than 450 phishing attacks used SSL in 2005, Dec. 2005. `http://news.netcraft.com`.

[50] Netcraft. Fraudsters attack two-factor authentication, July 2006. `http://news.netcraft.com`.

[51] B. O'Connor. Greater than 1: Defeating "strong" authentication in web applications. In *Defcon 15*, Las Vegas, NV, USA, Aug. 2007.

[52] A. Oprea, D. Balfanz, G. Durfee, and D. Smetters. Securing a remote terminal application with a mobile trusted device. In *Annual Computer Security Applications Conference (ACSAC'04)*, Tucson, AZ, USA, Dec. 2004.

[53] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *Financial Cryptography and Data Security (FC'06)*, volume 4107 of *LNCS*, Anguilla, British West Indies, Feb. 2006.

[54] A. Perrig and D. Song. Hash visualization: A new technique to improve real-world security. In *Cryptographic Techniques and E-Commerce (CrypTEC'99)*, Hong Kong, July 1999.

[55] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *ACM Computer and Communications Security (CCS'02)*, Washington, DC, USA, Nov. 2002.

[56] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iFRAMEs point to us. In *USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[57] Redmondmag.com. Coreflood trojan stole 500G of personal financial data. News article (Aug. 7, 2008). `http://redmondmag.com/news/article.asp?editorialsid=10111`.

[58] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, Baltimore, MD, USA, 2005.

[59] V. Roth, K. Richter, and R. Freidinger. A PIN-entry method resilient against shoulder surfing. In *ACM Computer and communications Security (CCS'04)*, Washington, DC, USA, Oct. 2004.

[60] A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. *Secrecy Analysis in Protocol Composition Logic*. Formal Logical Methods for System Security and Correctness, IOS Press, 2008. Volume based on presentations at Summer School Marktoberdorf, Germany, 2007.

[61] M. Shackman. Platform security - a technical overview, Nov. 2006. Symbian Developer Network article. http://developer.symbian.com/main/downloads/papers/plat_sec_tech_overview/platform_security_a_technical_overview.pdf.

[62] Symantec Security Response. Banking in silence. News article (Jan. 14, 2008). http://www.securityfocus.com/blogs/485.

[63] The Sydney Morning Herald. NZ bank adds security online. News article (Nov. 8, 2004). http://www.smh.com.au/.

[64] TheRegister.com. Phishing attack targets one-time passwords. News article (Oct. 12, 2005). http://www.theregister.co.uk/2005/10/12/outlaw_phishing/.

[65] E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Workshop on Usable Security (USEC'07)*, Lowlands, Scarborough, Trinidad/Tobago, Feb. 2007.

[66] P. van Oorschot. Message authentication by integrity with public corroboration. In *New Security Paradigms Workshop, (NSPW'05)*, Lake Arrowhead, CA, USA, Sept. 2005.

[67] P. van Oorschot and S. Stubblebine. On countering online dictionary attacks with login histories and humans-in-the-loop. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):235–258, Aug. 2006.

[68] WashingtonPost.com. Hackers zero in on online stock accounts. News article (Oct. 24, 2006).

[69] T. Weigold, T. Kramp, R. Hermann, F. Hörin, P. Buhle, and M. Baentsch. The Zurich trusted information channel – an efficient defence against man-in-the-middle and malicious software attacks. In *Conference on Trusted Computing and Trust in Information Technologies (TRUST'08)*, Villach, Austria, Mar. 2008.

[70] I. Wiener. Sample SecurID token emulator with token secret import, 2000. BugTraq post. http://archives.neohapsis.com/archives/bugtraq/2000-12/0428.html.

[71] M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *DIMACS Workshop on Usable Privacy and Security Systems*, July 2004.

[72] G. G. Xie, C. E. Irvine, and T. E. Levin. Quantifying effect of network latency and clock drift on time-driven key sequencing. In *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'02)*, Vienna, Austria, July 2002.

[73] Y. Zhang, S. Egelman, L. F. Cranor, and J. Hong. Phinding phish: An evaluation of anti-phishing toolbars. In *Network and Distributed System Security Symposium (NDSS'07)*, San Diego, CA, USA, Feb. 2007.

A AVISPA Test Code

Protocol: MP-Auth

As noted in Section 3, we include here results on our AVISPA [4] analysis of an idealized version (see below) of the MP-Auth protocol from Section 2.

Protocol Purpose

Authentication and key exchange between a mobile device M and a remote server S. More specifically, goals are (see Section 2, Table 1 for notation):

- M and S achieve mutual authentication (using P and E_S)
- M and S establish a secret (symmetric) session key for later use in encryption

How We Tested Using AVISPA

We used AVISPA Web interface available at http://www.avispa-project.org/web-interface/. We copied the HLPSL code (below) to the Web interface, and ran the relevant tests. Applicable tests to MP-Auth are: On the Fly Model Checker (OFMC), Constraint Logic-based Attack Searcher (CL-AtSe), and SAT-based Model Checker (SATMC). The Tree Automata based on Automatic Approximations for Analysis of Security Protocols (TA4SP) results are omitted from the AVISPA output below as the TA4SP back-end was not supported for our setup.

Idealization of MP-Auth

In MP-Auth, the browser B acts like a relaying party between M and S during the authentication and key exchange phase. Therefore B was removed from our idealized HLPSL model (and thus also, the SSL encryption between B and S). Also, the human user U was merged with M, as U only provides the password P to M. Hence the idealized MP-Auth is a two-party protocol, which is much simpler to analyze for AVISPA back-end protocol analyzers. As we have omitted party B, session ID verification is not required. The transaction integrity confirmation messages use K_{MS} established in the authentication phase. The confirmation messages have not been included in our model; we assume the secrecy of K_{MS} implicitly protects those messages. The idealized version of MP-Auth is given below.

```
M <- S: Rs
M -> S: {Rm}_Es.{f(Rs).M.P}_Kms, where Kms = f(Rs.Rm)
M <- S: {f(Rm)}_Kms
```

Results of the AVISPA Tests

No attacks have been reported by AVISPA on the idealized protocol. Results from the AVISPA back-end protocol analyzers are given below.

OFMC.

```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  /home/avispa/web-interface-computation/./tempdir/workfileP2NEkh.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 2.58s
  visitedNodes: 798 nodes
  depth: 10 plies
```

CL-AtSe.

```
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL
PROTOCOL
  /home/avispa/web-interface-computation/./tempdir/workfileP2NEkh.if
GOAL
  As Specified
BACKEND
  CL-AtSe
STATISTICS
  Analysed   : 5548 states
  Reachable  : 3529 states
  Translation: 0.01 seconds
  Computation: 0.14 seconds
```

SATMC.

```
SUMMARY
```

```
  SAFE
DETAILS
  STRONGLY_TYPED_MODEL
  BOUNDED_NUMBER_OF_SESSIONS
  BOUNDED_SEARCH_DEPTH
  BOUNDED_MESSAGE_DEPTH
PROTOCOL
  workfileP2NEkh.if
GOAL
  %% see the HLPSL specification..
BACKEND
  SATMC
COMMENTS
STATISTICS
  attackFound               false     boolean
  upperBoundReached         true      boolean
  graphLeveledOff           4         steps
  satSolver                 zchaff    solver
  maxStepsNumber            11        steps
  stepsNumber               5         steps
  atomsNumber               1196      atoms
  clausesNumber             5705      clauses
  encodingTime              1.12      seconds
  solvingTime               0.1       seconds
  if2sateCompilationTime    0.21      seconds
ATTACK TRACE
  %% no attacks have been found..
```

HLPSL Specification

```
role mobile (M, S: agent,
      Es: public_key,
      F, KeyGen: hash_func,
      P: text,
      SND, RCV: channel (dy)) played_by M def=

  local State : nat,
  Rm, Rs: text,
  Kms: message

  init State := 1

  transition
    2.  State  = 1 /\ RCV(Rs') =|>
        State':= 3 /\ Rm' := new()
                   /\ Kms':= KeyGen(Rs'.Rm')
                   /\ SND({Rm'}_Es.{F(Rs').M.P}_Kms')
                   /\ witness(M,S,rm,Rm')
                   /\ secret(Kms', sec_kms1, {M,S})

    3.  State  = 3 /\ RCV({F(Rm)}_Kms) =|>
        State':= 5 /\ request(M,S,rs,Rs)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role server(S: agent,
      Es: public_key,
      F, KeyGen: hash_func,
      Agents: (agent.text) set,
      SND, RCV: channel (dy)) played_by S def=
```

```
  local State : nat,
  Rm, Rs, P: text,
  Kms: message,
  M: agent

  init State := 0

  transition
    1.  State  = 0 /\ RCV(start) =|>
        State'= 2 /\ Rs' := new()
                  /\ SND(Rs')

    2. State  = 2  /\ RCV({Rm'}_Es.{F(Rs).M'.P'}_KeyGen(Rs.Rm'))
                   /\ in(M'.P', Agents) =|>
       State'= 4  /\ Kms' := KeyGen(Rs.Rm')
                  /\ SND({F(Rm')}_Kms')
                  /\ secret(Kms', sec_kms2, {M',S})
                  /\ request(S,M',rm,Rm')
                  /\ witness(S,M',rs,Rs)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(M, S: agent,
       Es: public_key,
       F, KeyGen: hash_func,
       P: text,
       Agents: (agent.text) set) def=

  local SS, RS, SM, RM: channel (dy)

  composition
     mobile (M,S,Es,F,KeyGen,P,SM,RM)
  /\ server (S,Es,F,KeyGen,Agents,SS,RS)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

    local Agents: (agent.text) set
    const m, s: agent,
    es: public_key,
    f, keygen: hash_func,
    rm, rs, sec_kms1, sec_kms2 : protocol_id,
    pm, pi: text

    init Agents := {m.pm, i.pi}
    intruder_knowledge = {m,s,f,keygen,pi,es,rs}

    composition
        session(m,s,es,f,keygen,pm,Agents)
     /\ session(m,s,es,f,keygen,pm,Agents)
     /\ session(i,s,es,f,keygen,pi,Agents)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal
```

```
  secrecy_of sec_kms1, sec_kms2
  authentication_on rm
  authentication_on rs

end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

environment()
```

B A PCL Proof Sketch for MP-Auth

In this section we discuss a proof sketch of MP-Auth using the Protocol Composition Logic (PCL) [15, 28, 60]. We assume that readers are familiar with the PCL proof system. See Appendix B.5 for a quick reference to frequently-used PCL axioms, rules, and definitions. We first outline the PCL setup for MP-Auth, and then provide the PCL analysis of mutual authentication and key secrecy.

B.1 PCL Setup

For the proof here, we use the following simplified version of MP-Auth. As the browser in MP-Auth only forwards messages between the web server and personal device, we remove the browser's role here. For simplification of the proof, we also replace $\{f(R_M)\}_{K_{MS}}$ with $[1]_{K_{MS}}$ (i.e., now the proof of ownership of the session key K_{MS} is provided through a MAC instead of an encryption). Also, to reduce confusion between PCL roles (generally upper case) and variables (generally lower case), we make necessary case transformation here.

$$
\begin{aligned}
&M \leftarrow S : ID_S.r_s \\
&M \rightarrow S : \{r_m\}_{E_S}.\{f(r_s).ID_U.P\}_{k_{ms}}, \text{ where } k_{ms} = f(r_s.r_m) \\
&M \leftarrow S : [1]_{k_{ms}}
\end{aligned}
$$

This simplified protocol is defined by 'roles' {**Init**, **Resp**} in Fig. 9, written using the protocol programming language as used in PCL. Each role specifies a sequence of actions to be executed by an honest principal in MP-Auth. An honest principal can execute one or more copies of its own role concurrently. Note that, roles are asymmetric in MP-Auth; for example, the server and client authenticate each other using a public key and a password, respectively, and an honest server does not impersonate a client. Here, a *thread* X refers to a principal $\hat{X}$ executing a particular instance of a role. Actions inside a thread include nonce generation, encryption, hash calculation, network communication and pattern matching (e.g., decryption). Each thread contains one or more 'basic sequences.' A basic sequence is a series of actions excluding any blocking actions (e.g., receive) except as the first action. Each role in MP-Auth consists of two basic sequences. PCL proofs use modal formulas of the form $\psi[P]_X\varphi$ which informally means that if X starts from a state where ψ holds, and executes the program P, then the resulting state is guaranteed to hold the security property φ, irrespective of the actions of a Dolev-Yao attacker and other honest principals. Let $idp := \langle \hat{M}.ID, \hat{M}.P \rangle$ (i.e., the userid-password pair of the user operating the mobile device $\hat{M}$), and $ids := \hat{S}.ID$ (the server's ID).

Init $=(\hat{M}, S, idp, ids)$ [
- `new` r_s;
- `send` $\hat{S}.\hat{M}.ids.r_s$;
- `receive` $\hat{M}.\hat{S}.t$;
- `match` $t/\langle encrm, encidp \rangle$;
- r_m := `pkdec` $encrm$, $\hat{S}$;
- k_{ms} := `hash` $r_s.r_m$;
- $decval$:= `symdec` $encidp$, k_{ms};
- `match` $decval/\langle hrs', idp \rangle$;
- `match` $idp/\langle \hat{M}.ID, \hat{M}.P \rangle$;
- hrs := `hash` r_s;
- `match` hrs'/hrs;
- $mac1$:= `hash` $1, k_{ms}$;
- `send` $\hat{S}.\hat{M}.mac1$;

]$_S$

Resp $=(M, idp)$ [
- `receive` $\hat{S}.\hat{M}.ids.r_s$;
- `new` r_m;
- $encrm$:= `pkcnc` r_m, $\hat{S}$;
- hrs := `hash` r_s;
- k_{ms} := `hash` $r_s.r_m$;
- $symterm$:= $hrs.idp$;
- $encidp$:= `symenc` $symterm$, k_{ms};
- `send` $\hat{M}.\hat{S}.encrm.encidp$;
- `receive` $\hat{S}.\hat{M}.mac1$;
- `verifyhash` $mac1, 1, k_{ms}$;

]$_M$

Figure 9: MP-Auth server (**Init**) and client (**Resp**) programs

Let $\mathcal{K} = \{\bar{k}_{\hat{S}}\}$, the private key of server $\hat{S}$. The public and private key pair for $\hat{S}$ is $(k_{\hat{S}}, \bar{k}_{\hat{S}})$. We use the following abbreviations for MP-Auth messages (see Fig. 9 for terms definitions):

$$msg1 := \hat{S}.\hat{M}.ids.r_s$$
$$msg2 := \hat{M}.\hat{S}.encrm.encidp$$
$$msg3 := \hat{S}.\hat{M}.mac1$$

Invariants. The 'honesty' rule in PCL is "an invariance rule for proving properties about the actions of principals that executes roles of a protocol" [15]. An honest principal in PCL is the one who follows one or more roles of the protocol. The honesty rule is used to reason in a deductible manner about the actions of the other party in the protocol. Formulas derived by the application of this rule are called 'invariants'. We use the following invariants of MP-Auth for our authentication and secrecy proofs.[20]

Γ_{mp1} $\quad \texttt{Honest}(\hat{S}) \wedge \texttt{Send}(S, msg) \supset \neg\texttt{Contains}(msg, idp)$

Γ_{mp2} $\quad \texttt{Honest}(\hat{M}) \supset \texttt{PkEnc}(M, r_m, k_{\hat{S}}) \supset$
$(\texttt{Receive}(M, msg1) < \texttt{New}(M, r_m) < \texttt{PkEnc}(M, r_m, k_{\hat{S}}) < \texttt{Send}(M, msg2))$

Γ_{mp3} $\quad \texttt{Honest}(\hat{M}) \wedge \texttt{Receive}(M, msg1) \wedge \texttt{Send}(M, msg2) \supset \texttt{FirstSend}(M, r_m, msg2)$

Γ_{mp4} $\quad \texttt{Honest}(\hat{M}) \wedge \texttt{Send}(M, msg) \supset \neg\texttt{Contains}(msg, HASH[k_{ms}](1))$

Γ_{mp5} $\quad \texttt{Honest}(\hat{S}) \wedge \texttt{New}(S, r_s) \wedge \texttt{Send}(S, msg1) \supset \texttt{FirstSend}(S, r_s, msg1)$

Γ_{mp1} states that the server $\hat{S}$ does not send any message containing idp. This essentially prohibits a server to execute the role of a client (mobile device). Otherwise, $\hat{S}$ could impersonate $\hat{M}$ which is false given that $\hat{S}$ is honest. Γ_{mp4} implies that an honest $\hat{M}$ does not send any message containing $HASH[k_{ms}](1)$, although $\hat{M}$ also knows k_{ms}. Only the server $\hat{S}$ sends such a term to prove the knowledge of k_{ms}.

Secrecy of password. As assumed in MP-Auth, the userid-password pair idp is unique for each user, and P is a shared secret between M and S. This assumption is formalized as follows.

$$\phi_{secp} ::= \texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \wedge \texttt{Has}(\hat{Z}, idp) \supset (\hat{Z} = \hat{M} \vee \hat{Z} = \hat{S})$$

Additionally, we now show that idp is not sent in the clear by any role (**Init, Resp**) of MP-Auth. The proof is straightforward: $\hat{S}$ does not send any message with idp, and $\hat{M}$ sends out idp only encrypted under k_{ms}. Assume that $\mathcal{K}' = \{k_{ms}\}$, and k_{ms} is the secret session key shared between $\hat{M}$ and $\hat{S}$ (see Section B.3).

$$\textbf{SAF2} \quad \texttt{SafeMsg}(E_{sym}[k_{ms}](hrs.idp), idp, \mathcal{K}') \tag{12}$$

$$12, \textbf{P1, S1, NET3} \quad \texttt{SafeNet}(idp, \mathcal{K}')[\textbf{Resp}]_M \ \texttt{SendsSafeMsg}(M, idp, \mathcal{K}') \tag{13}$$

$$\Gamma_{mp1}, \textbf{SAF0} \quad \texttt{SafeNet}(idp, \mathcal{K}')[\textbf{Init}]_S \ \texttt{SendsSafeMsg}(S, idp, \mathcal{K}') \tag{14}$$

From 13, 14, and the application of the **NET** rule and the **POS** axiom, we conclude that $\texttt{SafeNet}(idp, \mathcal{K}')$ is always true, and $\texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \wedge \texttt{Has}(\hat{Z}, idp) \supset (\hat{Z} = \hat{M} \vee \hat{Z} = \hat{S})$.

Security Properties of MP-Auth. MP-Auth requires the following authentication and secrecy properties to be satisfied by any successful protocol run.

1. *Server-side authentication.* At the end of a protocol run, both parties must agree on each other's identity, protocol completion status, and the secret session key generated from exchanged nonces. The authentication property of MP-Auth is formulated in terms of matching conversations [8]. The basic idea is that on execution of the server role (**Init**), we prove the existence of the intended client role (**Resp**) with a corresponding view of the messages exchanged. Matching conversations for server $\hat{S}$ and corresponding client $\hat{M}$ is formulated as follows:

$$\begin{aligned}\phi_{auth,\hat{S}} ::= \texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \supset \exists M.\texttt{Has}(M, k_{ms}) \wedge \\ ((\texttt{Send}(S, msg1) < \texttt{Receive}(M, msg1)) \wedge \\ (\texttt{Receive}(M, msg1) < \texttt{Send}(M, msg2)) \wedge \\ (\texttt{Send}(M, msg2) < \texttt{Receive}(S, msg2)) \wedge \\ (\texttt{Receive}(S, msg2) < \texttt{Send}(S, msg3)))\end{aligned}$$

[20] For the application of the honesty rule, the invariants must be preserved by all the basic sequences in MP-Auth. These proofs are straightforward, and thus we omit them here.

Note that the server receives no acknowledgement for the last message sent; i.e., the corresponding receive action is not a part of the authentication guarantee.

2. *Secrecy of k_{ms}.* The secret session key K_{ms} must not be known to any principal other than the server and client. This secrecy property of MP-Auth is formulated as follows:

$$\phi_{seckms} \ ::= \texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \wedge \texttt{Has}(\hat{Z}, k_{ms}) \supset (\hat{Z} = \hat{M} \vee \hat{Z} = \hat{S})$$

3. *Client-side authentication.* In MP-Auth, the client and server roles are not symmetric; i.e., the server is authenticated by showing the proof of ownership of the decryption key corresponding to the public key as used by the client role. On the other hand, the client prove its identity to the server by showing the knowledge of the shared secret P. Thus the client's view of mutual authentication is different than that of the server. For client $\hat{M}$, communicating with server $\hat{S}$, matching conversations is defined as follows.

$$\begin{aligned}\phi_{auth,\hat{M}} \ ::= \texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \supset \exists S.\texttt{Has}(S, k_{ms}) \wedge \\ ((\texttt{Send}(S, msg1) < \texttt{Receive}(M, msg1)) \wedge \\ (\texttt{Receive}(M, msg1) < \texttt{Send}(M, msg2)) \wedge \\ (\texttt{Send}(M, msg2) < \texttt{Receive}(S, msg2)) \wedge \\ (\texttt{Receive}(S, msg2) < \texttt{Send}(S, msg3)) \wedge \\ (\texttt{Send}(S, msg3) < \texttt{Receive}(M, msg3)))\end{aligned}$$

B.2 Server-side authentication

We use the secrecy of password (ϕ_{secp}, Section B.1), protocol invariants $\Gamma_{mp1}, \Gamma_{mp2}$ and Γ_{mp3}, to argue that there must be a thread of client $\hat{M}$ which must have performed certain actions corresponding to the client role **Resp** (see Fig. 9). Properties of nonces and encryption are also used. The proof sketch is summarized by the following steps below. Each step consists of three components: (i) the axioms, invariants and/or previous steps used, (ii) actions performed, and (iii) the resulting predicate. For example, the first of the proof below uses axioms **AA1, P1, AA4** (see Section B.5) to establish that the server performed certain actions in sequence.

AA1, P1, AA4	$[\textbf{Init}]_S \ \texttt{Send}(S, msg1) < \texttt{Receive}(S, msg2) < \texttt{Send}(S, msg3)$	(15)
AA1	$[\texttt{receive} \ \hat{M}.\hat{S}.t]_S \ \texttt{Receive}(S, \hat{M}.\hat{S}.t)$	(16)
AR1	$\texttt{Receive}(S, \hat{M}.\hat{S}.t)[\texttt{match} \ t/\langle encrm, encidp\rangle]_S \ \texttt{Receive}(S, \hat{M}.\hat{S}.encrm.encidp)$	(17)
REC	$\texttt{Receive}(S, \hat{M}.\hat{S}.encrm.encidp) \supset \texttt{Has}(S, \hat{M}.\hat{S}.encrm.encidp)$	(18)
PROJ	$\texttt{Has}(S, \hat{M}.\hat{S}.encrm.encidp) \supset \texttt{Has}(S, encrm) \wedge \texttt{Has}(S, encidp)$	(19)
AR3	$\texttt{Has}(S, encrm) \ [r_m := \texttt{pkdec} \ encrm, \hat{S};]_S \ \texttt{Has}(S, E[k_{\hat{S}}](r_m))$	(20)
DEC	$\texttt{Has}(S, E[k_{\hat{S}}](r_m)) \supset \texttt{Has}(S, r_m)$	(21)
$16, 17, 18, 19, 20, 21,$ **S1, P1**	$[\textbf{Init}]_S \ \texttt{Has}(S, r_m)$	(22)
AA1	$[\texttt{new} \ r_s]_S \ \texttt{New}(S, r_s)$	(23)
ORIG	$\texttt{New}(S, r_s) \supset \texttt{Has}(S, r_s)$	(24)
$23, 24,$ **S1, P1**	$[\textbf{Init}]_S \ \texttt{Has}(S, r_s)$	(25)
HASH0'	$[k_{ms} := \texttt{hash} \ r_s.r_m]_S \ \texttt{Has}(S, k_{ms})$	(26)
AR3	$\texttt{Has}(S, encidp) \ [decval := \texttt{symdec} \ encidp, k_{ms};]_S \ \texttt{Has}(S, E_{sym}[k_{ms}](decval))$	(27)
AR1	$\texttt{Has}(S, E_{sym}[k_{ms}](decval)) \ [\texttt{match} \ decval/\langle hrs', idp\rangle;]_S \ \texttt{Has}(S, E_{sym}[k_{ms}](hrs'.idp))$	(28)
ENC4	$\texttt{SymDec}(S, E_{sym}[k_{ms}](hrs'.idp), k_{ms}) \supset \exists Y.\texttt{SymEnc}(Y, hrs'.idp, k_{ms})$	(29)
ENC3	$\texttt{SymEnc}(Y, hrs'.idp, k_{ms}) \supset \texttt{Has}(Y, hrs'.idp) \wedge \texttt{Has}(Y, k_{ms})$	(30)
PROJ	$\texttt{Has}(Y, hrs'.idp) \supset \texttt{Has}(Y, hrs') \wedge \texttt{Has}(Y, idp)$	(31)
$31, \phi_{secp}$	$\texttt{Has}(Y, idp) \supset \hat{Y} = \hat{M} \vee \hat{Y} = \hat{S}$	(32)
$\Gamma_{mp1}, 27, 28, 29, 30, 31, 32,$ **S1, P1**	$[\textbf{Init}]_S \ \exists Y.\texttt{Has}(Y, idp) \supset \hat{Y} = \hat{M}$	(33)

$$\Gamma_{mp2}, 33 \quad [\mathbf{Init}]_S \ \exists M.(\texttt{Receive}(M, msg1) < \texttt{Send}(M, msg2)) \tag{34}$$

$$\mathbf{AN3, FS1, S1, P1} \quad [\mathbf{Init}]_S \ \texttt{FirstSend}(S, r_s, msg1) \tag{35}$$

$$33, 35, \mathbf{FS2} \quad [\mathbf{Init}]_S \ \texttt{Receive}(M, msg1) \wedge \hat{M} \neq \hat{S} \supset \texttt{Send}(S, msg1) < \texttt{Receive}(M, msg1) \tag{36}$$

$$\mathbf{AA1, S1, P1} \quad [\mathbf{Init}]_S \ \texttt{Receive}(S, msg2) \tag{37}$$

$$\Gamma_{mp3}, 37, \mathbf{FS2} \quad [\mathbf{Init}]_S \ \texttt{Honest}(\hat{M}) \wedge \hat{M} \neq \hat{S} \wedge \texttt{Receive}(M, msg1) \wedge \texttt{Send}(M, msg2) \supset \texttt{Send}(M, msg2) < \texttt{Receive}(S, msg2) \tag{38}$$

$$15, 34, 36, 38 \quad [\mathbf{Init}]_S \ \texttt{Honest}(\hat{M}) \wedge \hat{M} \neq \hat{S} \supset \phi_{auth,\hat{S}} \tag{39}$$

B.3 Secrecy of session key

We show that honest principals do not perform any actions that compromise the secrecy of session key k_{ms} through induction on the basic protocol sequences (see below for definitions of $\mathbf{Resp}_1, \mathbf{Resp}_2, \mathbf{Init}_1$, and $\mathbf{Init}_2$). Each induction step informally states that if k_{ms} has not already been compromised at the beginning of a basic sequence (i.e., $\texttt{SafeNet}(k_{ms}, \mathcal{K})$ is true), then the actions performed in that basic sequence by a thread X do not compromise k_{ms} (i.e., $\texttt{SendsSafeMsg}(X, k_{ms}, \mathcal{K})$ is true). For the basic sequence $\mathbf{Resp}_2$, the proof is straightforward: there is no `send` action. For $\mathbf{Init}_2$, the terms sent out by $\hat{S}$ do not contain k_{ms} in the clear (k_{ms} is used as a MAC key).

$$\begin{aligned}\text{Let } [\mathbf{Resp}_2]_{M'} : \ & [\texttt{receive } \hat{S}'.\hat{M}'.mac1'; \\ & \texttt{verifyhash } mac1', 1, k'_{ms};]_{M'}\end{aligned}$$

$$\mathbf{S1, NET1} \quad \texttt{SafeNet}(k_{ms}, \mathcal{K})[\mathbf{Resp}_2]_{M'} \ \texttt{SafeMsg}(\hat{S}'.\hat{M}'.mac1', k_{ms}, \mathcal{K}) \tag{40}$$

$$40, \mathbf{NET2} \quad \texttt{SafeNet}(k_{ms}, \mathcal{K})[\mathbf{Resp}_2]_{M'} \ \texttt{SendsSafeMsg}(M', k_{ms}, \mathcal{K}) \tag{41}$$

$$\begin{aligned}\text{Let } [\mathbf{Init}_2]_{S'} : \ & [\texttt{receive } \hat{M}'.\hat{S}'.t'; \\ & \texttt{match } t'/\langle encrm', encidp'\rangle; \\ & r'_m \ := \ \texttt{pkdec } encrm', \ \hat{S}'; \\ & k'_{ms} \ := \ \texttt{hash } r'_s.r'_m; \\ & decval' \ := \ \texttt{symdec } encidp', \ k'_{ms}; \\ & \texttt{match } decval'/\langle hrs'', idp'\rangle; \\ & \texttt{match } idp'/\langle \hat{M}'.ID, \hat{M}'.P\rangle; \\ & hrs' \ := \ \texttt{hash } r'_s; \\ & \texttt{match } hrs''/hrs'; \\ & mac1' \ := \ \texttt{hash } 1, k'_{ms}; \\ & \texttt{send } \hat{S}'.\hat{M}'.mac1';]_{S'}\end{aligned}$$

$$\mathbf{SAF5} \quad \texttt{SafeMsg}(HASH[k'_{ms}](1), k_{ms}, \mathcal{K}) \tag{42}$$

$$42, \mathbf{S1, NET3} \quad \texttt{SafeNet}(k_{ms}, \mathcal{K})[\mathbf{Init}_2]_{S'} \ \texttt{SendsSafeMsg}(S', k_{ms}, \mathcal{K}) \tag{43}$$

The session key k_{ms} is computed from two nonces, r_s and r_m, where r_s is sent in the clear. Thus the secrecy of k_{ms} lies on the secrecy of r_m. For the basic sequences that send out nonces, we need to show that the nonces are not equal to r_m, or that r_m is encrypted under the public key of $\hat{S}$. These arguments are formulated as $\Phi := \Phi^1_{r_m} \wedge \Phi^2_{r_m}$.

$$\Phi^1_{r_m} : \forall M, \hat{Z}.\texttt{New}(M, r_m) \wedge \texttt{PkEnc}(M, r_m, k_{\hat{Z}}) \supset \hat{Z} = \hat{S}$$
$$\Phi^2_{r_m} : \forall M, \texttt{New}(M, r_m) \wedge \texttt{Send}(M, msg) \supset \neg\texttt{ContainsOpen}(msg, r_m)$$

The predicate $\texttt{ContainsOpen}(m, a)$ asserts that a can be obtained from m (directly or a series of unpairings only) without any decryption. $\Phi^1_{r_m}$ and $\Phi^2_{r_m}$ can be established from invariant Γ_{mp2}: from thread M's point of view, it knows that it has freshly generated the nonce r_m, and has only sent r_m out encrypted with only principal $\hat{S}$'s public key.

$$\text{Let } [\mathbf{Resp}_1]_{M'} : [\texttt{receive } \hat{S}'.\hat{M}'.ids'.r'_s;\ \texttt{new } r'_m;$$
$$encrm' := \texttt{pkcnc } r'_m,\ \hat{S}';$$
$$hrs' := \texttt{hash } r'_s;\ k'_{ms} := \texttt{hash } r'_s.r'_m;$$
$$symterm' := hrs'.idp';$$
$$encidp' := \texttt{symenc } symterm',\ k'_{ms};$$
$$\texttt{send } \hat{M}'.\hat{S}'.encrm'.encidp';]M'$$

$$\text{Case : } r'_m \neq r_m \tag{44}$$
$$\mathbf{S1, SAF3} \quad [\mathbf{Resp}_1]_{M'}\ \texttt{SafeMsg}(E_{pk}[\hat{S}'](r'_m), r_m, \mathcal{K}) \tag{45}$$
$$45, \mathbf{NET3} \quad \texttt{SafeNet}(r_m, \mathcal{K})[\mathbf{Resp}_1]_{M'}\ \texttt{SendsSafeMsg}(M', r_m, \mathcal{K}) \tag{46}$$
$$\text{Case : } r'_m = r_m \tag{47}$$
$$\mathbf{S1, P1} \quad [\mathbf{Resp}_1]_{M'}\ \texttt{PkEnc}(M', r_m, k_{\hat{S}'}) \tag{48}$$
$$48, \Phi^1_{r_m} \quad [\mathbf{Resp}_1]_{M'}\ \hat{S}' = \hat{S} \tag{49}$$
$$48, 49, \mathbf{SAF3} \quad [\mathbf{Resp}_1]_{M'}\ \texttt{SafeMsg}(E_{pk}[\hat{S}](r_m), r_m, \mathcal{K}) \tag{50}$$
$$50, \mathbf{NET3} \quad \texttt{SafeNet}(r_m, \mathcal{K})[\mathbf{Resp}_1]_{M'}\ \texttt{SendsSafeMsg}(M', r_m, \mathcal{K}) \tag{51}$$

$$\text{Let } [\mathbf{Init}_1]_{S'} : [\texttt{new } r'_s;$$
$$\texttt{send } \hat{S}'.\hat{M}'.ids'.r'_s;]_{S'}$$

$$\Phi^2_{r_m} \quad [\mathbf{Init}_1]_{S'}\ r'_s \neq r_m \tag{52}$$
$$52, \mathbf{SAF0} \quad [\mathbf{Init}_1]_{S'}\ \texttt{SafeMsg}(\hat{S}'.\hat{M}'.r'_s, r_m, \mathcal{K}) \tag{53}$$
$$53, \mathbf{NET3} \quad \texttt{SafeNet}(r_m, \mathcal{K})[\mathbf{Init}_1]_{S'}\ \texttt{SendsSafeMsg}(S', r_m, \mathcal{K}) \tag{54}$$

As k_{ms} is computed from r_m and r_s, we can say $\texttt{SafeNet}(r_m, \mathcal{K}) \supset \texttt{SafeNet}(k_{ms}, \mathcal{K})$. Thus, from 41, 43, 46, 51, 54 and the application of the **NET** rule and the **POS** axiom, we conclude that $\texttt{SafeNet}(k_{ms}, \mathcal{K})$ is always true, and $\Phi \wedge \texttt{Honest}(\hat{M}) \wedge \texttt{Honest}(\hat{S}) \wedge \texttt{Has}(\hat{Z}, k_{ms}) \supset (\hat{Z} = \hat{M} \vee \hat{Z} = \hat{S})$.

B.4 Client-side authentication

We use protocol invariants Γ_{mp4} and Γ_{mp5}, secrecy of the server's private key, and properties of nonces to argue that there must be a thread of server $\hat{S}$ which must have performed certain actions corresponding to the server role **Init** (see Fig. 9). The proof sketch is summarized by the following steps below.

$$\mathbf{AA1, P1, AA4} \quad [\mathbf{Resp}]_M\ \texttt{Receive}(M, msg1) < \texttt{Send}(M, msg2) < \texttt{Receive}(M, msg3) \tag{55}$$
$$\mathbf{AA1} \quad [\texttt{receive } \hat{S}.\hat{M}.mac1]_M\ \texttt{Receive}(M, \hat{S}.\hat{M}.mac1) \tag{56}$$
$$56, \mathbf{S1, P1} \quad [\mathbf{Resp}]_M\ \texttt{Receive}(M, \hat{S}.\hat{M}.mac1) \tag{57}$$
$$57, \mathbf{HASH3'} \quad [\mathbf{Resp}]_M \wedge \texttt{Honest}(\hat{M}) \supset \exists X.\texttt{Computes}(X, HASH[k_{ms}](1)) \wedge \texttt{Send}(X, msg) \wedge \texttt{Contains}(msg, HASH[k_{ms}](1)) \tag{58}$$
$$57, 58 \quad [\mathbf{Resp}]_M \wedge \texttt{Honest}(\hat{M}) \supset \exists X.(\texttt{Receive}(M, msg3) < \texttt{Send}(X, msg3)) \tag{59}$$
$$\mathbf{HASH2} \quad [\texttt{verifyhash } mac1, 1, k_{ms}]_M\ mac1 = HASH[k_{ms}](1) \tag{60}$$
$$60, \mathbf{S1, P1} \quad [\mathbf{Resp}]_M\ mac1 = HASH[k_{ms}](1) \tag{61}$$
$$61, \Gamma_{mp4}, \mathbf{SEC, DEC} \quad [\mathbf{Resp}]_M \wedge \texttt{Honest}(\hat{S}) \supset \exists X.((\texttt{Send}(X, msg3) < \texttt{Receive}(X, msg2)) \wedge \texttt{PkDec}(X, E[k_{\hat{S}}](r_m), \bar{k}_{\hat{S}})) \supset (\hat{X} = \hat{S} \wedge \texttt{Has}(S, r_m)) \tag{62}$$
$$\mathbf{S1, P1, AN3, FS1} \quad [\mathbf{Resp}]_M\ \texttt{FirstSend}(M, r_m, msg2) \tag{63}$$
$$63, \mathbf{FS2} \quad [\mathbf{Resp}]_M\ \texttt{Receive}(S, msg2) \wedge \hat{M} \neq \hat{S} \supset \texttt{Send}(M, msg2) < \texttt{Receive}(S, msg2) \tag{64}$$
$$\Gamma_{mp5}, \mathbf{AA1, FS2} \quad [\mathbf{Resp}]_M\ \texttt{Receive}(M, msg1) \wedge \hat{M} \neq \hat{S} \supset \texttt{Send}(S, msg1) < \texttt{Receive}(M, msg1) \tag{65}$$
$$55, 59, 62, 64, 65 \quad [\mathbf{Resp}]_M \wedge \texttt{Honest}(S) \wedge \hat{M} \neq \hat{S} \supset \phi_{auth,M} \tag{66}$$

B.5 Frequently-used PCL Axioms, Rules, and Definitions in MP-Auth

The PCL axioms and rules that we use here have been proposed previously [15, 28, 60, 35]. Some of these axioms are natural logical assumptions (also known as first order logical axioms, e.g., creation of a nonce implies possession of that nonce). Others are 'idealized' cryptographic axioms which provide formal logic equivalent of standard cryptography. (Note that, in reality, most cryptographic primitives do not achieve idealized cryptographic functionality.) In the axioms here, a denotes an action (e.g., `send`, `receive`, `new`, `pkenc`), and `a` denotes the corresponding predicate in PCL. Axiom **AA4** states that after thread X executes actions a, ..., b in a sequence, the action predicates `a`, ..., `b` are temporarily ordered in the corresponding sequence. Axiom **SEC** states that if a principal $\hat{X}$ is honest, and a thread Y of another principal $\hat{Y}$ can decrypt a term encrypted with the public key of $\hat{X}$ then principals $\hat{X}$ and $\hat{Y}$ must be the same ($\wedge$ is logical conjunction and $\supset$ can be read as 'implies'). We introduce a new axiom **HASH0'** which refers to the fact that if principal X computes the hash of a value then X also possesses the computed hash.

AA1 $\phi[a]_X$ `a`

AA4 $\phi[a;...;b]_X$ `a` $<$ ... $<$ `b`

AN3 $\phi[\texttt{new } x]_X$ $\texttt{Fresh}(X, x)$

REC $\texttt{Receive}(X, x) \supset \texttt{Has}(X, x)$

ENC $\texttt{Has}(X, x) \wedge \texttt{Has}(X, K) \supset \texttt{Has}(X, E[K](x)$

PROJ $\texttt{Has}(X, x.y) \supset \texttt{Has}(X, x) \wedge \texttt{Has}(X, y)$

DEC $\texttt{Has}(X, E[K](x)) \wedge \texttt{Has}(X, K) \supset \texttt{Has}(X, x)$

AR1 $\texttt{a}(x)[\texttt{match } q(x)/q(t)]_X$ $\texttt{a}(t)$

AR3 $\texttt{a}(x)[y := \texttt{dec } x, K]_X$ $\texttt{a}(E[K](y))$

SEC $\texttt{Honext}(\hat{X}) \wedge \texttt{Decrypt}(Y, E[k_{\hat{X}}](x)) \supset (\hat{Y} = \hat{X})$

G4 $\frac{\phi}{\theta[P]_X \phi}$

S1 $\frac{\phi_1[P]_X \phi_2 \quad \phi_2[P']_X \phi_3}{\phi_1[PP']_X \phi_3}$

P1 $\texttt{Persist}(X, t)[a]_X$ $\texttt{Persist}(X, t)$ for $\texttt{Persist} \in \{\texttt{Has, Send, Receive}\}$

FS1 $\texttt{Fresh}(X, t)[\texttt{send } t']_X$ $\texttt{FirstSend}(X, t, t')$, where $t \subseteq t'$

FS2 $\texttt{FirstSend}(X, t, t') \wedge \texttt{a}(Y, t'') \supset \texttt{Send}(X, t) < \texttt{a}(Y, t'')$, where $X \neq Y$ and $t \subseteq t''$

ENC3 $\texttt{Enc}(X, m, k) \supset \texttt{Has}(X, k) \wedge \texttt{Has}(X, m)$, where $\texttt{Enc} \in \{\texttt{SymEnc}, \texttt{PkEnc}\}$

PENC4 $\texttt{PkDec}(X, E[k](m), \bar{k}) \supset \exists Y.\texttt{PkEnc}(Y, m, k)$

ENC4 $\texttt{SymDec}(X, E_{sym}[k](m), k) \supset \exists Y.\texttt{SymEnc}(Y, m, k)$

HASH3' $\texttt{Receive}(X, HASH[k](x)) \supset$
$\exists Y.\texttt{Computes}(Y, HASH[k](x)) \wedge \texttt{Send}(Y, m) \wedge \texttt{Contains}(m, HASH[k](x))$

HASH2 $\phi[\texttt{verifyhash } m', m, k]_X$ $m' = HASH[k](m)$

HASH0' $\texttt{Computes}(X, HASH(m)) \supset \texttt{Has}(X, HASH(m))$

SAF0 $\neg\texttt{SafeMsg}(s, s, \mathcal{K}) \wedge \texttt{SafeMsg}(x, s, \mathcal{K})$, where x is an atomic term, and $x \neq s$

SAF2 $\texttt{SafeMsg}(E_{sym}[k](m), s, \mathcal{K}) \equiv \texttt{SafeMsg}(m, s, \mathcal{K}) \vee k \in \mathcal{K}$

SAF3 $\texttt{SafeMsg}(E_{pk}[k](m), s, \mathcal{K}) \equiv \texttt{SafeMsg}(m, s, \mathcal{K}) \vee \bar{k} \in \mathcal{K}$

SAF5 $\texttt{SafeMsg}(HASH[k](m), s, \mathcal{K})$

$\texttt{SendsSafeMsg}(X, s, \mathcal{K}) \equiv \forall m.(\texttt{Send}(X, m) \supset \texttt{SafeMsg}(m, s, \mathcal{K}))$

$\texttt{SafeNet}(s, \mathcal{K}) \equiv \forall X.\texttt{SendsSafeMsg}(X, s, \mathcal{K})$

NET1 $\texttt{SafeNet}(s, \mathcal{K})[\texttt{receive } m]_X$ $\texttt{SafeMsg}(m, s, \mathcal{K})$

NET2 $\texttt{SendsSafeMsg}(X, s, \mathcal{K})[\texttt{a}]_X$ $\texttt{SendsSafeMsg}(X, s, \mathcal{K})$, where `a` is not a `send`

NET3 $\texttt{SendsSafeMsg}(X, s, \mathcal{K})[\texttt{send } m]_X$ $\texttt{SafeMsg}(m, s, \mathcal{K}) \supset \texttt{SendsSafeMsg}(X, s, \mathcal{K})$

POS $\texttt{SafeNet}(s, \mathcal{K}) \wedge \texttt{Has}(X, m) \wedge \neg\texttt{SafeMsg}(m, s, \mathcal{K}) \supset \exists k \in \mathcal{K}.\texttt{Has}(X, k) \vee \texttt{New}(X, s)$