This is the full version of the extended abstract which appeared in:
Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS'08), March 18-20, 2008
M. Abe, V. Gligor (Eds.), ACM, pp. 300--311.

Provably Secure Browser-Based User-Aware Mutual Authentication over TLS

(Full version)

Sebastian Gajek[1] and Mark Manulis[2] and Ahmad-Reza Sadeghi[1] and Jörg Schwenk[1]

[1] Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany
{sebastian.gajek|joerg.schwenk}@nds.rub.de
ahmad.sadeghi@trust.rub.de
[2] UCL Crypto Group, Belgium
mark.manulis@uclouvain.be

Abstract. The standard solution for user authentication on the Web is to establish a TLS-based secure channel in server authenticated mode and run a protocol on top of TLS where the user enters a password in an HTML form. However, as many studies point out, the average Internet user is unable to identify the server based on a X.509 certificate so that impersonation attacks (e.g., phishing) are feasible. We tackle this problem by proposing a protocol that allows the user to identify the server based on *human perceptible authenticators* (e.g., picture, voice). We prove the security of this protocol by refining the game-based security model of Bellare and Rogaway and present a proof of concept implementation.

Key words: Browser-based security protocols, human-perceptible authentication, TLS, phishing

1 Introduction

1.1 Motivation

The World Wide Web has grown and become a fundamental technology of the information society whereby the browser plays an indispensable function both as the user's interface to access the rich world of Web-based services and provider of security tools in order for safeguarding a honest perception of the Web. This constrains the design of applications and security protocols to TLS, HTTP and higher order protocols, such as AJAX, AFLEX or SOAP. We call protocols realizable within the constraints of commodity Web browsers *browser-based* protocols.

An immediate need on the Web is to authenticate users and permit access to services or private information. A widely adopted approach is to use TLS in server authenticated mode and execute a protocol on top of TLS whereby the user enters a password in a Web form. A folklore belief is that the user is able to authenticate the server through a X.509 certificate. Recent studies point out however that average-skilled Internet users understand neither server certificates nor browsers' security indicators [17, 25, 37]. Users are overstrained by the verification procedure and tend to ignore browser's warnings; in fact, they evaluate Web sites on the basis of non-technical indicators (e.g., brands, logos). This ceremony[3] provides a wrong sense of security. An adversary may fake the site and disclose the user's password ("phishing attacks").

Apart from the user, an alternative problem is the X.509 public key infrastructure for server authentication. Various Certification Authorities (CAs) have stored their root certificates in browsers and are thus trusted *per se*. Although CAs may considerably differ in their issuing policies browsers equally treat the certificates. Adversaries may exploit weak issuing policies, retrieve certificates for rogue servers, and do a lot of harm (e.g., [21]).

In addition to that, the browser remains a potential point of attack. The browser is partly controllable by a remote party, using the facets of Web languages. An adversary is capable of sending queries that alter the internal state of the browser or reveal temporary connection information, including access to the browser's user interface (chrome), cache, cookies, history and has led to several vulnerabilities in browser-based protocols (e.g., [22, 29, 35]).

[3] Carl Ellison coined at Crypto 2005 Rump Session the term ceremony to denote the paradigm that a provably secure cryptosystem becomes insecure when it is interfaced to a user. See [19].

1.2 Our Contribution

We solve the above problems by presenting a protocol that explicitly considers the user and the browser as protocol participants, and ties the user's authentication to the TLS secure channel. We relieve the user from the responsibility of authenticating the server by mediating the roles of authentication between the user, browser and server.

We first make the server identify the browser on the basis of client certificates. This step ensures that the server establishes a secure channel to the browser, however does not authenticate the user. In order to prove its identity to the user, the server sends a *human perceptible authenticator (HPA)* (e.g., personal picture, voice recording). The user has to recognize the authenticator; verification of server certificates (and the underlying public key infrastructure) and any security indicator (e.g., URL, padlock) is irrelevant. Finally, the user authenticates to the server, using a password. We call the proposed protocol BBMA and the corresponding security goal *browser-based mutual authentication (BBMA)*. Using BBMA has the following advantages over previous browser-based protocols:

1. BBMA provides *user-aware* authentication which takes into account human skills. BBMA reduces the security of browser-based protocols to the assumption that users are capable of identifying human perceptible authenticators—as they are used to do in the physical world where identities are provided in an easily recognizable fashion. Based on results of human authentication protocols [15, 16, 26, 38], visual authenticators are appealing; however, BBMA is flexible in the choice of authenticators. TLS in client authentication mode, by contrast, ensures browser-aware server authentication. In this context browser-aware authentication means that the browser authenticates whenever the server certificate is valid. This is especially true for rogue servers using valid certificates. Client certificates prevent the adversary from the impersonation of the browser; they do not prevent that a user discloses sensitive information requested by such rogue servers, as the browser indicates an authenticated communication.
2. We prove BBMA secure in the refined model for authentication due to Bellare and Rogaway [7]. The refinements concern partitioning of the client entity into user and browser entities, since both entities are active participants of browser-based protocols and target of different attacks. This requires to formulate the user and browser behavior. To the best of our knowledge, it is the first browser-based protocol that has been formally analyzed in the model.
3. BBMA is flexible. The protocol fits into the standard TLS specification since the human perceptible authenticator and the password are exchanged as HTML payloads over the secure TLS channel. We have implemented the protocol [32] in a proof of concept. The implementation comprises an online registration protocol where the user may setup client certificates for different browsers in a user-convenient way. That is, we do not need client certificates to be issued by a trusted CA. They may be self-signed. The server has to store the hash value of the public key as a unique cryptographic identifier of the browser.

The use of both client certificates and passwords in BBMA may seem redundant at first glance, since the methods are commonly regarded as orthogonal mechanisms to authenticate the user. We stress the fact that a client certificate only authenticates the browser. It rules out the ability to distinguish between multiple users, having access to the browser. Moreover, some business models require a fine-grained access control where the user is permitted to sign on different accounts (e.g., online banking). Using passwords in addition to client certificates (instead of multiple client certificates) is a design solution for ease of use that complies with requirements of browser-based protocols and is provably secure in our model.

1.3 Related Work

Browser-based authentication protocols originate from HTTP Basic Authentication [20] which has been a standard tool to authenticate users for static Web pages. With the advent of dynamic Web page generation, it proved more useful to deliver the password to the scripting language as the value of an HTML form input field.

Another, independent line of development are password-based authenticated key exchange protocols (PAKE). Bellovin and Merritt [10], Bellare et al. [6, 9], Boyko et al. [11], and McKenzie [11, 30] proposed various PAKE structures that have been augmented to provide resiliency against offline dictionary attacks [27] and to provide

forward secrecy under various assumptions [2, 12, 13, 28]. Later Abdalla and Pointcheval [3] proposed SPAKE a simplified PAKE structure that provides more flexibility in the choice of groups. Steiner et al. [36] and Abdalla et al. [1] have shown that PAKE structures may be incorporated into the TLS protocol framework and proposed provably secure ciphersuites for password-based authenticated key exchange.

The analysis of these PAKE protocols has in common that protocol participants are assumed to be machines that are able to strictly obey the protocol specification. From this follows a model for authentication that idealizes the user and browser. Such a model does not thoroughly reflect the Web where ordinary Internet users are involved in the protocol and browsers naively trust certificates that origin from any pre-installed authority. The model presented in this paper relaxes these assumptions in a way that neither a trusted CA exists nor the user understands the meaning of public key infrastructures. Under these assumptions PAKE protocols are vulnerable to attacks where the adversary simply mimics the password dialog. In order to thwart such attacks, BBMA makes use of passwords and client secret keys. Then, impersonation of the user without compromising the client secret is infeasible, even if the adversary has revealed the password.

So far, few browser-based protocols have been subject to rigorous security analysis: Kormann and Rubin [29] show that Microsoft's .NET passport, a Web-based realization of the Kerberos protocol for single sign on, is susceptible to attacks where the adversary steals the ticket granting ticket cookie. Groß [22] analyzes SAML, an alternative single sign on protocol, and shows that the protocol is vulnerable to adaptive attacks where the adversary intercepts the authentication token contained in the URL. Soghoian and Jakobsson [35] investigate the SiteKey-protocol (a.k.a. PassMark Security Inc.'s Two-Factor-Two-Way Authentication™) that displays a previously negotiated image in addition to password forms in order to signal that the user is connected to the benign server. The authors show the feasibility of stealing the shared secret that is stored in a cookie.

By contrast, BBMA has formal security arguments and is provably secure in a revised version of the standard model for authentication due to Bellare and Rogaway. The model takes into account that the adversary controls the network and may circumvent the browser's same origin policy (see Section 2.1) in order to corrupt weak identifiers, such as cookies. BBMA uses cryptographically strong bindings, i.e., client certificates that are not affected in case of possible violation of the same origin policy.

Groß et al. prove in [24] the security of WS-Federation passive Requestor Profile—a browser-based protocol for federated identity management. The proof is carried out in the browser model [23] that is build on the Reactive Simulatability framework due to Pfitzmann and Waidner [31]. The model abstracts away the TLS-protected channel through an ideal functionality that captures the same cryptographic task and presupposes ideal users who are able to identify servers based on certificates. There exists no soundness proof that TLS is simulatable and realizes such functionality, especially with respect to the user behavior. BBMA takes explicitly into account the TLS protocol and is shown to be provably secure in the Random Oracle Model when instantiated with the key transport ciphersuite in server authentication mode.

1.4 Organization

The remaining sections are structured as follows. In Section 2, we present the formal security model for browser-based authentication protocols. We describe BBMA in a high level description of TLS and prove that it is secure in Section 3. In Section 4, we discuss variants of realizing BBMA. Finally, we conclude the paper in Section 5.

2 Modeling Browser-Based Mutual Authentication (BBMA)

In this section we refine the original security model for mutual authentication from [7] according to our idea on partitioning the client party into the entities representing the participating user and browser. Similar to [7] we consider an active probabilistic polynomial time (PPT) adversary who interacts with involved parties through queries and controls all the communication.

2.1 Communication Model

Protocol Participants and Long-Lived Keys We consider the *server* $\mathcal{S}$, the *browser* $\mathcal{B}$, and the *human user* $\mathcal{U}$ as participants of a BBMA protocol Π. Furthermore, by $\mathcal{C}$ we denote the *client* given by a pair $(\mathcal{U}, \mathcal{B})$.

According to our refinement and contrary to the current cryptographic literature, we explicitly model the user as a stand-alone entity. We start from scratch and make the weakest security assumptions to represent a practical setting namely that $\mathcal{U}$ is security-unaware and $\mathcal{B}$ supports standard techniques of commodity Web browsers.

The human user $\mathcal{U}$ is modeled as a probabilistic machine with "very limited" computational capabilities. Intuitively, it is clear that human users are not able to perform the same computations as browser and server. More specifically, we assume that human users are unable to perform cryptographic operations, but may recognize high-entropy data if it is in a human-recognizable form and may output low-entropy passwords. More technically, we assume that $\mathcal{U}$ holds a long-lived key $LL_{\mathcal{U}} \in \{0,1\}^{p_1(\kappa)}$ which is typically some set of pairs that consists of a human-memorable password $pw \in \mathcal{D}$ and a *human perceptible authenticator (HPA)* $w \in \mathcal{W}$. Here and in the following $p_i : \mathbb{N} \to \mathbb{N}, i \in [1,5]$ is a polynomial, $\kappa \in \mathbb{N}$ is a security parameter and $\mathcal{D}, \mathcal{W}$ are dictionaries.

The browser $\mathcal{B}$ is modeled as a PPT machine that exchanges protocol messages with $\mathcal{S}$ through physical communication links and interacts with $\mathcal{U}$, using various visualization techniques and user input (e.g., via HTML forms). We assume that $\mathcal{B}$ holds a high-entropy long-lived secret $LL_{\mathcal{B}} \in \{0,1\}^{p_2(\kappa)}$. Typically, $LL_{\mathcal{B}}$ consists of a private key $sk_{\mathcal{B}}$ for which $\mathcal{B}$ maintains the corresponding public key certificate. The server $\mathcal{S}$ is modeled in a classical way namely as a PPT machine with the long-lived key $LL_{\mathcal{S}} \in \{0,1\}^{p_2(\kappa)}$ which usually consists of the own private key and further secrets shared with $\mathcal{U}$ and $\mathcal{B}$. We model the communication flow from $\mathcal{B}$ to $\mathcal{U}$ through an abstract channel whose input is interfaced to the *visualization function* `render` and output is interfaced to the *human perception function* `recognize`. We also assume that there is a channel allowing $\mathcal{U}$ to provide inputs to $\mathcal{B}$ running on the user's device, i.e., through common interfaces such as keyboards, mice, touch-screens, etc. In the following we specify the relationship between the functions `render` and `recognize`. In its simplest form, we could model the user as a deterministic Turing machine that waits for a certain, high-entropy input (the authenticator), and then outputs a low-entropy secret (the password). By using the `render` and `recognize` functions, we are able to express a more complex behavior of the user.

Modeling Browser Message Processing via `render`-Function The browser plays the role of a messenger and translates the messages from user and server, however, is unaware of the semantic meaning: It takes a message $m \in \mathcal{M}$ from the message space $\mathcal{M} \in \{0,1\}^{\lambda_1(\kappa)}$ (the space of all HTML documents) that the server $\mathcal{S}$ wishes to send to user $\mathcal{U}$ and presents the information according to the browser's state $\Psi \in \{0,1\}^{\lambda_2(\kappa)}$ to $\mathcal{U}$ as a Web page. Here and in the following, $\lambda_i : \mathbb{N} \to \mathbb{N}, i \in [1,2]$ is a polynomial. State Ψ denotes the browser's configuration for processing the retrieved message that may be altered by querying the browser's DOM[4] model.

Loosely speaking, the DOM model describes the browser's tree-based view of a Web page and defines access to document nodes, browser and connection information through Web scripting languages (e.g., Javascript). The *same origin* security policy which is universally supported in browsers controls access to the nodes and ensures that there is no information retrieval between pages that have different domains, including access to the browser's chrome, cache, cookies, and history. Access to any ephemeral and long-term secrets which are stored in separated containers, or the ability to open, create or delete a file from the operating system, are subject to different security policies which normally involve user interaction.

$\mathcal{B}$ communicates to $\mathcal{U}$ calling the visualization function $\texttt{render} : \mathcal{M} \times \Psi \to \mathcal{M}^*$. More precisely, whenever $\mathcal{B}$ wishes to communicate a protocol message m to $\mathcal{U}$ it calls `render`, which takes m and current browser's state Ψ as input and outputs the appropriate visualization m^* to $\mathcal{U}$.

Modeling User Behavior via `recognize`-Function The overall goal of browser-based authentication protocols is to provide mutual authentication between the entity user $\mathcal{U}$ and the entity server $\mathcal{S}$. Since the user is involved in the protocol execution, we need to formulate the behavior. The necessity for modeling the user behavior is due to the fact that the adversary may mount attacks that target the user.

Devising a rigorous model that captures user behavior clearly raises various issues, both technical and philosophical. For instance, which human abilities can we model? Should we restrict the user behavior to certain skills, say perceive certain objects. If so, how can we model the quality of skills? Do users behave "correctly"

[4] Document Object Model, see [39] for details.

in the sense that they always behave in the same way? The problem becomes even more intricate when one wishes to quantify the behavior. This work takes a rather simplistic approach. We assume that the user is able to recognize the high-entropy HPA w and remembers a low-entropy password pw.

The communication between $\mathcal{B}$ and $\mathcal{U}$ is established in a consistent way across through the boolean *human perception function* $\texttt{recognize} : \mathcal{M}^* \times \mathcal{W} \rightarrow \{0,1\}$ which on input a visualized message $m^* \in \mathcal{M}^*$ and w outputs 1 if $\mathcal{U}$ recognizes w among the content of m^*; otherwise the output is 0. In this paper we assume that if m^* contains w (denoted as $m^*|w$) then `recognize` outputs 1, i.e., the ability of $\mathcal{U}$ to recognize w is *perfect*. On the other hand, we do *not* assume that w is the only HPA for which `recognize` outputs 1, i.e., we do *not* idealize $\mathcal{U}$ as there can be some set $\mathcal{W}^* \subseteq \mathcal{W}$ which contains HPAs that are *perfectly human-indistinguishable* from $\mathcal{U}$ according to the following definition.

Definition 1 (Perfect Human-Indistinguishability of HPAs[5]). *Let $w \in \mathcal{W}$ be some given HPA. For any $m^* \in \mathcal{M}^*$ and any $w^* \in \mathcal{W}$, we say that w and w^* are perfectly human-indistinguishable, if for any human user $\mathcal{U}$*

$$\left| \Pr[\mathcal{U}.\texttt{recognize}(m^*|w, w) = 1] - \Pr[\mathcal{U}.\texttt{recognize}(m^*|w^*, w) = 1] \right| = 0$$

where the probabilities are computed over the choices of w^*. By $\mathcal{W}^* \subseteq \mathcal{W}$ we denote the set of all perfectly human-indistinguishable HPAs for some given $w \in \mathcal{W}$ assuming that $w \in \mathcal{W}^*$.

Our main idea in designing user-aware security protocols based on HPAs is to opt for authenticators for which $\mathcal{W}^*$ is *sufficiently small* for most of the users. In this case the probability that an adversary chooses or guesses some HPA that cannot be distinguished from w by $\mathcal{U}$ can be kept low. The ideal case would be if $\mathcal{W}^*$ would consist only of w. We call w a *good* HPA if the size of the set $\mathcal{W}^*$ is sufficiently small such that the term $|\mathcal{W}^*|/|\mathcal{W}|$[6] which addresses user-awareness in our proof and is used (beside further cryptography-related terms) to compute the overall probability of a successful attack is negligible.

For our protocol we assume that the HPA used by $\mathcal{U}$ in the execution of our protocol is good.

Remark 1. An interesting topic of future work would be to consider the similarity of authenticators in order to model users' fuzziness. Consider the following example. Let w be an authenticator that consists of an image, and let w^* be the same image, but marginally compressed. Some users would be able to distinguish w^* and w whereas some would fail. Many metrics exist that allow for scaling the meaning of "some" and quantify the similarity for different types of HPAs. Using such metrics is an interesting challenge, since it allows to refine the assumptions on user behavior.

Protocol Sessions and Instances In order to model participation of $\mathcal{C} = (\mathcal{U}, \mathcal{B})$ and $\mathcal{S}$ in distinct executions of the same BBMA protocol Π we consider *instances* $[\mathcal{C}, sid_{\mathcal{C}}]$ and $[\mathcal{S}, sid_{\mathcal{S}}]$ where $sid_{\mathcal{C}}, sid_{\mathcal{S}} \in \mathbb{N}$ are respective *session identifiers*. If $sid_{\mathcal{C}} = sid_{\mathcal{S}}$ then we assume that both instances belong to the same session, and say the instances are *partnered*. Note that in our protocol $sid_{\mathcal{C}}$ and $sid_{\mathcal{S}}$ will be given by the concatenation of random nonces exchanged between $\mathcal{B}$ and $\mathcal{S}$ in the beginning of the protocol. For simplicity, we sometimes omit the indication of the instance and write $\mathcal{C}$ and $\mathcal{S}$ instead. Whether, $\mathcal{C}$ or $\mathcal{S}$ denote the actual party or its instance is usually visible from the context.

Execution States Each instance $[\mathcal{C}, sid_{\mathcal{C}}]$ and $[\mathcal{S}, sid_{\mathcal{S}}]$ may be either *used* or *unused*. The instance is considered as unused if it has never been initialized. Each unused instance can be initialized with the corresponding long-lived key. The instance is initialized upon being created. After the initialization the instance is marked as used, and turns into the *stand-by* state where it waits for an invocation to execute the protocol. Upon receiving such invocation the instance turns into a *processing* state where it proceeds according to the protocol specification. The instance remains in the processing state until it collects enough information to decide whether

[5] The definition we use in this full version differs from the one given in the published conference version of this paper as it no longer uses a security parameter for the estimation of the ability of human users to recognize HPAs.

[6] In the published conference version of this paper this term was mistakenly denoted as $2^{|\mathcal{W}^*|-|\mathcal{W}|}$ though the text description explicitly mentions the relation between the sizes of $\mathcal{W}^*$ and $\mathcal{W}$.

the execution was successful or not, and to *terminate* then. If the execution is successful then we say that the instance *accepts* before it terminates. The acceptance in case of the client instance $[\mathcal{C}, sid_{\mathcal{C}}]$ with $\mathcal{C} = (\mathcal{U}, \mathcal{B})$ is implied by the acceptance of the user $\mathcal{U}$ regardless of the state of the browser $\mathcal{B}$, as $\mathcal{U}$ is the ultimate endpoint of the communication and controls the browser. In case that the execution was not successful (due to failures) instances terminate without accepting, i.e., they *abort*. Note that the client instance aborts whenever the user or the browser aborts.

2.2 Security Model

Assumptions The adversary $\mathcal{A}$ controls all communication between the protocol parties. This implies:

- The adversary controls the domain name resolution. Upon sending forged domain resolution responses, the adversary is capable of foiling browsers' same origin policy, thus accessing through the DOM model the browser's chrome, cache, cookies, and history. However, the adversary is prevented from opening, creating or deleting a file from the operating system. Since the long-term secrets are prevented from caching and part of different security policies, this implies that the adversary is incapable of accessing passwords and HPAs.
- There exists no trusted third party in the sense of a trusted CA. A certified public key in a X.509 server certificate is treated as a public key that can be identified by a unique identifier (i.e., hash value of the public key). Disburden the model from certified public keys captures the fact that (i) the adversary has retrieved from a trusted CA certified public keys for rogue servers such that $\mathcal{B}$ accepts the keys without any user notification and (ii) the behavior of users who ignore the certificate validation warnings.
- The adversary is unable to corrupt $\mathcal{B}$. Note that in this model we do not deal with malware[7] attacks against the browser and server, therefore, do not consider the case where $\mathcal{A}$ reveals the ephemeral and longterm secrets stored inside $\mathcal{B}$.
- The adversary is unable to corrupt $\mathcal{S}$. Note also that in this model we do not deal with malware attacks against the server. This means that the adversary is excluded from revealing the ephemeral and longterm secrets stored inside $\mathcal{S}$.
- The communication within the client $\mathcal{C}$ that is the communication between $\mathcal{U}$ and $\mathcal{B}$ is not authenticated. Note that in this model we do not deal with physical attacks against the user; otherwise over-the-shoulder surfers are enabled to see the human perceptible authenticator and the user password.

Adversarial Queries The adversary $\mathcal{A}$ participates in the actual protocol execution via the following queries:

- $\mathsf{Execute}(\mathcal{C}, \mathcal{S})$: This query models passive attacks. The adversary $\mathcal{A}$ eavesdrops the execution of the new protocol session between $\mathcal{C}$ and $\mathcal{S}$. $\mathcal{A}$ is given the corresponding transcript.
- $\mathsf{Invoke}(\mathcal{C}, \mathcal{S})$: This is a special query used by an active adversary $\mathcal{A}$ to invoke the actual protocol execution. This query is described in detail in the next paragraph.
- $\mathsf{Send}(P, m)$: This query models active attacks where $\mathcal{A}$ sends a message to some instance of $P \in \{\mathcal{U}, \mathcal{B}, \mathcal{S}\}$. That is, messages addressed to $\mathcal{U}$ are implicitly handled as messages addressed to the associated browser $\mathcal{B}$ with the subsequent execution of the $\texttt{render}(m, \Psi)$ function while messages to the server $\mathcal{S}$ from $\mathcal{U}$ are processed by browser $\mathcal{B}$ and then send in a machine-readable form. The information flow in the $\mathsf{Send}(P, m)$-query is implicitly denoted by including the sender and receiver identities in m.
- $\mathsf{RevealState}(\mathcal{B})$: This query models attacks which reveal information stored with the browser's state Ψ.

It should be mentioned that corruptions of the browser and server cannot be allowed because of the weakness of the underlying TLS protocols (see Section 3). Such corruptions would immediately compromise all previously executed sessions.

[7] Consideration of malware attacks and augmentation of the proposed model with Trusted Computing functionalities to model resistance against malware attacks is surely an interesting aspect for the future work on security of browser-based protocols.

Protocol Execution in the Presence of $\mathcal{A}$ By asking the $\mathsf{Execute}(\mathcal{C}, \mathcal{S})$ query $\mathcal{A}$ obtains the transcript of the complete protocol execution between new instances of $\mathcal{C}$ and $\mathcal{S}$ without being able to perform any further actions during this execution.

On the other hand, if $\mathcal{A}$ wishes to actively participate in the execution of Π then it can ask a special invocation query $\mathsf{Invoke}(\mathcal{C}, \mathcal{S})$ implying that a new instance of $\mathcal{U}$ starts the protocol execution with the new instance of $\mathcal{S}$ using the associated instance of browser $\mathcal{B}$. $\mathcal{A}$ obtains then the first protocol message returned by $\mathcal{B}$ (which is usually generated on some input received from $\mathcal{U}$, e.g., the entered URL). Active participation of $\mathcal{A}$ is defined further through the Send queries.

Correctness and Mutual Authentication in BBMA Protocols The following definition of correctness specifies the purpose of BBMA protocols.

Definition 2 (Correctness). *A BBMA protocol Π is* correct *if each* $\mathsf{Execute}(\mathcal{C}, \mathcal{S})$ *query results in two instances* $[\mathcal{C}, sid_{\mathcal{C}}]$ *and* $[\mathcal{S}, sid_{\mathcal{S}}]$ *which are partnered (*$sid_{\mathcal{C}} = sid_{\mathcal{S}}$*) and accept prior to termination.*

If two instances $[\mathcal{C}, sid_{\mathcal{C}}]$ and $[\mathcal{S}, sid_{\mathcal{S}}]$ would accept and terminate without being partnered ($sid_{\mathcal{C}} \neq sid_{\mathcal{S}}$), then interleaving attacks are feasible.

In the following we define the main security goal of BBMA protocols, namely the requirement of mutual authentication between participating $\mathcal{U}$ and $\mathcal{S}$.

Definition 3 (Browser-Based Mutual Authentication). *Let Π be a correct BBMA protocol and* $\mathsf{Game}_{\Pi}^{bbma}(\mathcal{A}, \kappa)$ *the interaction between the instances of $\mathcal{C} = (\mathcal{U}, \mathcal{B})$ and $\mathcal{S}$ with a PPT adversary $\mathcal{A}$ who is allowed to query* Execute, Invoke, Send, *and* RevealState. *We say that $\mathcal{A}$* wins *if at some point during the interaction:*

1. *An instance $[\mathcal{C}, sid_{\mathcal{C}}]$ accepts but there is **no** partnered instance $[\mathcal{S}, sid_{\mathcal{S}}]$, **or***
2. *An instance $[\mathcal{S}, sid_{\mathcal{S}}]$ accepts but there is **no** partnered instance $[\mathcal{C}, sid_{\mathcal{C}}]$.*

The maximum probability of this event over all adversaries is denoted $\mathsf{Succ}_{\Pi}^{bbma}(\mathcal{A}, \kappa) = \max_{\mathcal{A}} |\Pr[\mathcal{A}$ *wins in* $\mathsf{Game}_{\Pi}^{bbma}(\mathcal{A}, \kappa)]|$. *We say that a BBMA protocol Π provides* mutual authentication *if this probability is a negligible function of the security parameter κ.*

Recall, the acceptance in case of the client instance $[\mathcal{C}, sid_{\mathcal{C}}]$ with $\mathcal{C} = (\mathcal{U}, \mathcal{B})$ is implied by the acceptance of the user $\mathcal{U}$ (regardless of the state of $\mathcal{B}$) since $\mathcal{U}$ is the ultimate endpoint of the communication and controls the browser. Then, the first requirement ensures that client $\mathcal{C}$ authenticates to the matching server $\mathcal{S}$. The second requirement ensures that the server $\mathcal{S}$ authenticates to the matching client $\mathcal{C}$.

3 Browser-Based Mutual Authentication Protocol based on TLS

In this section we describe our protocol for browser-based authentication based on the standard TLS protocol. We call the proposed protocol `BBMA`.

3.1 Building Blocks

TLS Protocol A main pillar of `BBMA` is the mutually authenticated *key transport*. This complies with RSA-based ciphersuites as specified in [4]. These suites are preferentially negotiated between standard browsers and servers. (It is notable that `BBMA` can be likewise designed for DH-based key exchange.) We write in parenthesis the corresponding TLS messages.

Cryptographic Primitives `BBMA` uses the (well-known) cryptographic primitives from the cryptographic suites of the TLS protocol, namely:

- A *pseudo-random function* PRF : $\{0,1\}^{p_3(\kappa)} \times \{0,1\}^* \rightarrow \{0,1\}^*$. Note that TLS defines PRF with data expansion s.t. it can be used to obtain outputs of a variable length which becomes useful for the key derivation phase. By $\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa)$ we denote the maximum advantage over all PPT adversaries in distinguishing the outputs of PRF from those of a random function better than by a random guess. Let l_1, l_2, l_3 and l_4 denote the publicly known *labels* specified in TLS for the instantiation of PRF (see [18, Sect. 5]).
- A *symmetric encryption scheme* which provides indistinguishability under chosen plaintext attacks (IND-CPA). The symmetric encryption operation is denoted Enc and the corresponding decryption operation Dec. By $\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(\kappa)$ we denote the maximum advantage over all PPT adversaries in breaking the IND-CPA property of (Enc, Dec) better than by a random guess.
- An IND-CPA secure *asymmetric encryption scheme* whose encryption operation is denoted $\mathcal{E}$ and the corresponding decryption operation $\mathcal{D}$. By $\mathsf{Adv}^{ind-cpa}_{(\mathcal{E},\mathcal{D})}(\kappa)$ we denote the maximum advantage over all PPT adversaries in breaking the IND-CPA property of $(\mathcal{E}, \mathcal{D})$ better than by a random guess.
- A cryptographic *collision-resistant hash function* Hash: $\{0,1\}^* \rightarrow \{0,1\}^{p_4(\kappa)}$. By $\mathsf{Succ}^{coll}_{\mathtt{Hash}}(\kappa)$ we denote the maximum success probability over all PPT adversaries in finding a collision, i.e., a pair $(m, m') \in \{0,1\}^* \times \{0,1\}^*$ such that $\mathtt{Hash}(m) = \mathtt{Hash}(m')$.
- A *digital signature scheme* which provides existential unforgeability under chosen message attacks (EUF-CMA). The signing operation is denoted Sig and the corresponding verification operation Ver. We denote the maximum success probability over all PPT adversaries given access to the signing oracle in finding a forgery by $\mathsf{Succ}^{euf-cma}_{(Sig,Ver)}(\kappa)$.
- A *message authentication code* function HMAC which is believed to satisfy *weak unforgeability under chosen message attacks* (WUF-CMA) [5]. By $\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa)$ we denote the maximum success probability over all PPT adversaries given access to the tagging/verification oracle in finding a forgery.

3.2 Protocol Description

Before BBMA can be executed, a registration phase is necessary where every party obtains its own long-lived key. Then, the long-lived key $LL_{\mathcal{U}}$ consists of the low-entropy password pw and the HPA w which are shared with $\mathcal{S}$. The long-lived key $LL_{\mathcal{B}}$ stored in the credential store of the browser $\mathcal{B}$ consists of the certified private/public key pair $(sk_{\mathcal{B}}, cert_{\mathcal{B}})$; the corresponding public key $pk_{\mathcal{B}}$ is part of the certificate. Finally, the long-lived key $LL_{\mathcal{S}}$ consists of the certified private/public key pair $(sk_{\mathcal{S}}, cert_{\mathcal{S}})$, the password pw and authenticator w. The server maps (pw, w) to $cert_{\mathcal{B}}$ in order to select the matching authenticator during the protocol execution. In the following we briefly describe the execution of our BBMA protocol illustrated in Figure 1.

1. **Protocol Invocation.** The user $\mathcal{U}$ initiates the protocol by communicating the server's URL to the browser $\mathcal{B}$. Upon resolving the corresponding address $\mathcal{B}$ chooses a *nonce* $r_{\mathcal{C}}$ of length $p_5(\kappa)$ at random and forwards it to $\mathcal{S}$ (ClientHello). In response $\mathcal{S}$ chooses a random *nonce* $r_{\mathcal{S}}$ and a *TLS session identifier* sid of length $p_5(\kappa)$ and appends it to the own certificate $cert_{\mathcal{S}}$ (ServerHello). We stress that sid chosen by $\mathcal{S}$ is not the session identifier $sid_{\mathcal{S}}$ used in our security model but a value specified in TLS.

2. **Key Material Negotiation.** $\mathcal{B}$ chooses a *pre-master secret* k_p of length $p_5(\kappa)$ at random and sends it to $\mathcal{S}$ encrypted with the received public key $pk_{\mathcal{S}}$ (ClientKeyExchange). The pre-master secret k_p is used to derive the *master secret* k_m through a pseudo-random function PRF on input $(l_1, r_{\mathcal{C}}|r_{\mathcal{S}})$ with k_p as the secret seed. This key derivation is performed based on the standard TLS pseudo-random function PRF (see [4, Sect. 5]). The master secret is then used as secret seed for the instantiation of the pseudo-random function PRF on input $(l_2, r_{\mathcal{C}}|r_{\mathcal{S}})$ to derive the *session keys* (k_1, k_2) used to encrypt and authenticate session messages exchanged between $\mathcal{B}$ and $\mathcal{S}$. TLS specifies the generation of six session keys: A symmetric encryption key, a MAC key, and an IV for block ciphers only (either for client and server). For simplicity, we denote k_1 as the encryption key and k_2 as the authentication key which are the same for $\mathcal{B}$ and $\mathcal{S}$. The browser $\mathcal{B}$ also proves possession of the private key $sk_{\mathcal{B}}$ by signing the hash $h_{\sigma_{\mathcal{C}}}$ over all previously negotiated messages, i.e., signature $\sigma_{\mathcal{C}}$ (ClientVerify).

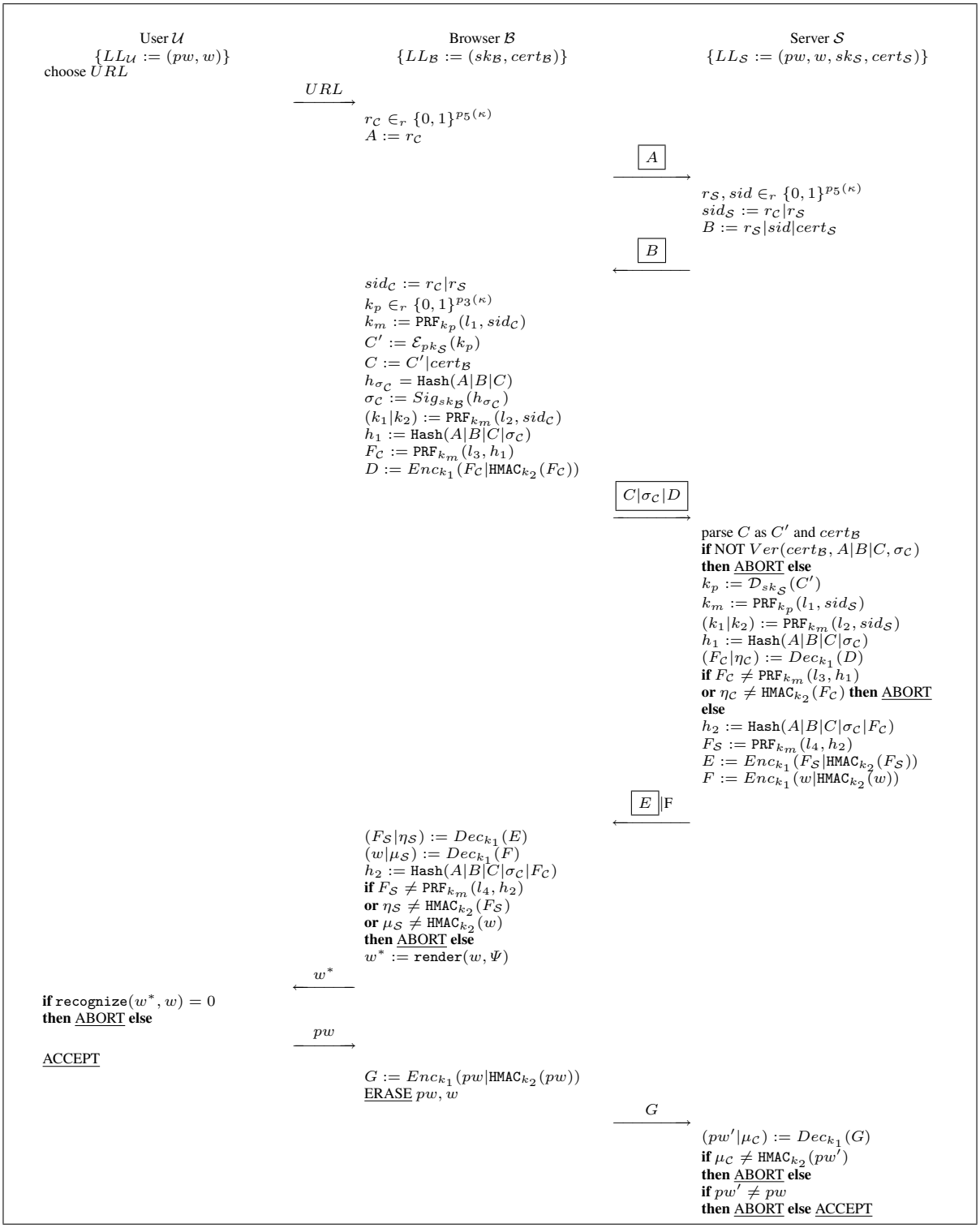

Fig. 1. BBMA Protocol over TLS with Mutual Authentication between User $\mathcal{U}$ and Server $\mathcal{S}$. Boxed messages denote the standard TLS handshake.

3. **Session Key Confirmation.** $\mathcal{B}$ confirms the session key generation, i.e., $F_{\mathcal{C}}$ is the first message that is authenticated via HMAC computed with k_2 and encrypted via the symmetric encryption scheme computed with k_1. $F_{\mathcal{C}}$ is computed as output of PRF on input (l_3, h_1) with k_m as the secret seed; whereby h_1 denotes the hash value computed over all messages previously processed by $\mathcal{B}$ (Finished). $\mathcal{S}$ verifies $\sigma_{\mathcal{C}}$, using the public key $pk_{\mathcal{B}}$. Further, $\mathcal{S}$ generates k_m and derives the session keys (k_1, k_2) in a similar way. $\mathcal{S}$ uses the own session keys (k_1, k_2) to ensure that it communicates with $\mathcal{B}$ through the verification of $F_{\mathcal{C}}$. If the verification fails, $\mathcal{S}$ aborts the protocol. Otherwise, it confirms the negotiated session parameters, using PRF on input (l_4, h_2) with k_m as secret seed; whereby h_2 denotes the hash value over the received messages. The output of PRF is first authenticated via HMAC computed with k_2 and then encrypted via the symmetric encryption scheme computed with k_1.

4. **Authenticate to the User.** The actual authentication between $\mathcal{U}$ and $\mathcal{S}$ proceeds in the last communication rounds. $\mathcal{S}$ sends authenticator w encrypted with k_1 with the attached message authentication code computed using k_2. We call the message in a high-level description the HumanAuth message. $\mathcal{B}$ communicates the decrypted authenticator to $\mathcal{U}$ through execution of the render function which takes as input the authenticator w and state Ψ and outputs the visualization of w named w^*. The abstract human perception function recognize is used to model the ability of $\mathcal{U}$ to decide whether the authenticator w^* matches the original authenticator w which is shared with $\mathcal{S}$ after the initialization stage. Upon the successful recognition $\mathcal{U}$ communicates the password pw to $\mathcal{B}$ and accepts. The browser in turn forwards pw to the server over the established secure channel, i.e., authenticated via HMAC computed with k_2 and encrypted with k_1. We call the response message in a high-level description the HumanResponse message. $\mathcal{S}$ accepts upon the successful verification of the received password.

Before we continue with the security analysis we reemphasize the triangular model of authentication deployed in BBMA. When verifying $F_{\mathcal{C}}$, $\mathcal{S}$ is sure to deal with $\mathcal{B}$. Then, $\mathcal{S}$ resolves $cert_{\mathcal{B}}$ to look up for the corresponding authenticator w. If no matching pair $(cert_{\mathcal{B}}, w)$ exists, $\mathcal{S}$ aborts the protocol; otherwise $\mathcal{S}$ proceeds by sending $F_{\mathcal{S}}$. Upon the stage, $\mathcal{S}$ knows that it can establish a secure channel with $\mathcal{B}$. When verifying $F_{\mathcal{S}}$, $\mathcal{B}$ knows that it can establish a secure channel to $\mathcal{S}$. Upon the stage, the protocol ensures that $\mathcal{S}$ and $\mathcal{B}$ are able exchange confidential messages. As with TLS in client authentication mode, the channel does not prevent $\mathcal{U}$ from contacting a rogue server and disclosing sensitive information. However, when verifying w through the execution of recognize, $\mathcal{U}$ is sure to communicate to $\mathcal{S}$ through $\mathcal{B}$, since $\mathcal{S}$ is the only owner of w apart from U. Upon this stage, the protocol ensures that $\mathcal{S}$ is authenticated to $\mathcal{U}$. Finally, when verifying pw, $\mathcal{S}$ is sure to deal with the matching client $\mathcal{C} := (\mathcal{U}, \mathcal{B})$.

3.3 Security Analysis

In the following we analyze the security of BBMA. We recall that the goal of the protocol is to provide browser-based mutual authentication between $\mathcal{U}$ and $\mathcal{S}$ according to Definition 3. Although not stated in Theorem 1 explicitly, the security proof of proposed BBMA based on the current TLS standard is valid in the Random Oracle Model (ROM) [8]. The reason is that the specification of TLS prescribes the use of the RSA encryption according to PKCS#1 (a.k.a. RSA-OAEP) which in turn is known to provide IND-CPA security in ROM (see [33] for the proof). However, Theorem 1 assumes $(\mathcal{E}, \mathcal{D})$ to be IND-CPA secure (independent of ROM). Thus, using an encryption scheme whose security holds under standard assumptions would also disburden the current security of BBMA from the strong assumptions of ROM.

Theorem 1 (BBMA-Security). *Let q denote the total number of executed protocol sessions. If* PRF *is pseudo random, (Enc, Dec) and $(\mathcal{E}, \mathcal{D})$ are IND-CPA secure,* Hash *is collision-resistant, (Sig, Ver) is EUF-CMA secure, and* HMAC *is WUF-CMA secure, then* BBMA *provides mutual authentication in the sense of Definition 3 and*

$$\mathsf{Succ}^{bbma}_{\mathtt{BBMA}}(\kappa) \leq \frac{3q^2}{2^{p_5(\kappa)}} + \frac{q^2}{2^{p_3(\kappa)}} + 4q\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(\kappa) + 4q\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa) + 3q\mathsf{Succ}^{coll}_{\mathtt{Hash}}(\kappa) + 4q\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa) + q\mathsf{Adv}^{ind-cpa}_{(\mathcal{E},\mathcal{D})}(\kappa) + q\mathsf{Succ}^{euf-cma}_{(Sig,Ver)}(\kappa) + \frac{q|\mathcal{W}^*|}{|\mathcal{W}|}.$$

Proof. (Sketch) In this proof we apply the meanwhile classical proving technique from [34]. We construct a *sequence of games* $\mathbf{G}_i$, $i = 0, \ldots, 21$ and denote by Win_i the event that adversary $\mathcal{A}$ breaks the mutual authentication of the protocol in game $\mathbf{G}_i$, i.e., wins in the corresponding interaction as described in Definition 3.

Game $\mathbf{G}_0$. [*Real protocol*] This is the real $\mathsf{Game}^{bbma}_{\mathtt{BBMA}}(\kappa)$ played between a simulator Δ and a PPT adversary $\mathcal{A}$. Δ simulates protocol participants according to the natural protocol specification and answers all queries of the adversary.

Game $\mathbf{G}_1$. [*Same TLS Session Id*] In this game the simulation aborts if during the interaction the simulator on behalf of the browser $\mathcal{B}$ chooses the same TLS session id sid in two different protocol sessions. Considering the probability for the collision of two random choices we obtain

$$|\Pr[\mathsf{Win}_1] - \Pr[\mathsf{Win}_0]| \leq \frac{q^2}{2^{p_5(\kappa)}}.$$

Game $\mathbf{G}_2$. [*Same Nonces*] In this game the simulation aborts if during the interaction the simulator on behalf of the browser $\mathcal{B}$ or server $\mathcal{S}$ chooses the same random nonce $r_{\mathcal{C}}$ or $r_{\mathcal{S}}$ in two different protocol sessions. Similar to Game $\mathbf{G}_1$ we obtain

$$|\Pr[\mathsf{Win}_2] - \Pr[\mathsf{Win}_1]| \leq \frac{2q^2}{2^{p_5(\kappa)}}.$$

Note that since in our protocol both session ids – $sid_{\mathcal{C}}$ and $sid_{\mathcal{S}}$ – are computed as concatenation $r_{\mathcal{C}}|r_{\mathcal{S}}$ this game rules out the occurrence of different (uncorrupted) client or server instances having the same session id, i.e., for the honest party each new session is associated with a different session id.

Game $\mathbf{G}_3$. [*Hash collision of $h_{\sigma_{\mathcal{C}}}$*] This game proceeds exactly as Game $\mathbf{G}_2$ except that the simulation aborts if during the interaction Δ computes the same hash value $h_{\sigma_{\mathcal{C}}} := \mathtt{Hash}(A|B|C)$ in two different session. Note that games $\mathbf{G}_1$ and $\mathbf{G}_2$ ensure that values A and B are fresh in different protocol sessions. Hence, the simulation aborts in the current game if Δ computes a hash collision. Due to the collision-resistance of $\mathtt{Hash}$ we obtain

$$|\Pr[\mathsf{Win}_3] - \Pr[\mathsf{Win}_2]| \leq q\mathsf{Succ}^{coll}_{\mathtt{Hash}}(\kappa).$$

This game implies that $h_{\sigma_{\mathcal{C}}}$ is fresh for each new session.

Game $\mathbf{G}_4$. [*Signature Forgery of $\sigma_{\mathcal{C}}$*] In this game the simulation aborts if $\mathcal{A}$ asks a Send query for the message $C'|cert_{\mathcal{B}}|\sigma_{\mathcal{C}}|F_{\mathcal{C}}$ such that $\sigma_{\mathcal{C}}$ is a valid signature on some string $r_{\mathcal{C}}|sid|r_{\mathcal{S}}|cert_{\mathcal{S}}|C'|cert_{\mathcal{B}}$ which has never been signed by $\mathcal{B}$ before, i.e., if a signature forgery occurs. It is possible to construct a forger algorithm $\mathcal{F}$ which will simulate all protocol participants and answer all queries of $\mathcal{A}$ such that if $\mathcal{A}$ wins in this game then $\mathcal{F}$ breaks the EUF-CMA security of the applied digital signature scheme (Sig, Ver). Thus,

$$|\Pr[\mathsf{Win}_4] - \Pr[\mathsf{Win}_3]| \leq q\mathsf{Succ}^{euf-cma}_{(Sig,Ver)}(\kappa).$$

Note that with this game we have also excluded replay attacks on the protocol messages containing $A = r_{\mathcal{C}}|sid$, $B = A|r_{\mathcal{S}}|cert_{\mathcal{S}}$ and the part $C'|cert_{\mathcal{B}}$ from the third protocol message in Figure 1.

Game $\mathbf{G}_5$. [*Same Pre-master Secret*] In this game the simulation aborts if during the interaction the simulator on behalf of the browser $\mathcal{B}$ chooses the same pre-master secret k_p in two different protocol sessions. Thus,

$$|\Pr[\mathsf{Win}_5] - \Pr[\mathsf{Win}_4]| \leq \frac{q^2}{2^{p_3(\kappa)}}.$$

Game $\mathbf{G}_6$. [*Indistinguishability of C'*] This game proceeds exactly as Game $\mathbf{G}_5$ except for the following actions of Δ: if Δ receives a message $B = r_{\mathcal{S}}|sid|cert_{\mathcal{S}}$ as part of the adversarial Send query then Δ computes $C' := \mathcal{E}_{pk_{\mathcal{S}}}(\alpha)$ for some additional randomly chosen $\alpha \neq k_p$. Otherwise, if $B = A|r_{\mathcal{S}}|cert'_{\mathcal{S}}$ with some $cert'_{\mathcal{S}} \neq cert_{\mathcal{S}}$ then Δ computes $C' := \mathcal{E}_{pk_{\mathcal{S}}}(k_p)$, i.e., exactly as specified in the protocol. We denote this certificate injection event by $\mathsf{InjCert}$. With the above modification we consider in our proof attacks against users that do not properly verify the validity of server's certificate. That is, the protocol proceeds in a natural way even if the user accepts some forged or invalid certificate. On the other hand, if the real server's certificate is received (no $\mathsf{InjCert}$ occurred) then the purpose of this game is to show that the security of the used asymmetric

encryption scheme has an impact on the secrecy of the transmitted pre-master secret k_p. Due to the IND-CPA property of $(\mathcal{E}, \mathcal{D})$ and since InjCert can occur only in a session invoked via the Invoke query we obtain

$$|\Pr[\mathsf{Win}_6] - \Pr[\mathsf{Win}_5]| \leq q\mathsf{Adv}^{ind-cpa}_{(\mathcal{E},\mathcal{D})}(\kappa).$$

Note that the specification of TLS prescribes the use of the RSA encryption according to PKCS#1 (a.k.a. RSA-OAEP) which in turn is known to provide IND-CPA security in ROM (see [33] for the proof).

Game $\mathbf{G}_7$. [*Pseudo-randomness of k_m*] This game proceeds exactly as Game $\mathbf{G}_6$ except that if no InjCert occurred then the simulator chooses the master secret k_m at random instead of computing it using the pseudo-random function PRF. This can be done since the secret seed (given by the pre-master secret k_p) used in the computation is uniformly distributed. Note that if InjCert has occurred then these modifications are not applied. Due to the pseudo-randomness of PRF we obtain

$$|\Pr[\mathsf{Win}_7] - \Pr[\mathsf{Win}_6]| \leq q\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa).$$

Game $\mathbf{G}_8$. [*Pseudo-randomness of k_1 and k_2*] This game proceeds exactly as Game $\mathbf{G}_7$ except that if no InjCert occurred then the simulator chooses $k_1|k_2$ at random instead of computing it using PRF. Note that the master secret k_m is already uniform if InjCert has not occurred (according to Game $\mathbf{G}_7$). On the other hand, if InjCert has occurred then $k_1|k_2$ are computed as specified in the protocol. Due to the pseudo-randomness of PRF we obtain

$$|\Pr[\mathsf{Win}_8] - \Pr[\mathsf{Win}_7]| \leq q\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa).$$

Game $\mathbf{G}_9$. [*Hash collision of h_1*] This game proceeds exactly as Game $\mathbf{G}_8$ except that the simulation aborts if during the interaction Δ computes the same hash value $h_1 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}})$ in two different sessions. Note that games $\mathbf{G}_1$ to $\mathbf{G}_3$ ensure that values A, B, and $\sigma_{\mathcal{C}}$ are fresh in different protocol sessions. Hence, the simulation aborts in the current game if Δ computes a hash collision. Due to the collision-resistance of Hash we obtain

$$|\Pr[\mathsf{Win}_9] - \Pr[\mathsf{Win}_8]| \leq q\mathsf{Succ}^{coll}_{\mathtt{Hash}}(\kappa).$$

Note, this game implies that h_1 is fresh for each new session.

Game $\mathbf{G}_{10}$. [*Pseudo-randomness of $F_{\mathcal{C}}$*] This game proceeds exactly as Game $\mathbf{G}_9$ except that if no InjCert occurred then the simulator chooses $F_{\mathcal{C}}$ at random instead as computing it using PRF. Due to the pseudo-randomness of PRF we obtain

$$|\Pr[\mathsf{Win}_{10}] - \Pr[\mathsf{Win}_9]| \leq q\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa).$$

Note, this game implies that $F_{\mathcal{C}}$ does not leak any information about k_m. Note also, this and the previous games exclude replay and forgery attacks on A, B and $C|\sigma_{\mathcal{C}}$, since the pseudo-random function is instantiated with fresh values, i.e., h_1, and different labels.

Game $\mathbf{G}_{11}$. [*Forgery of $\eta_{\mathcal{C}}$*] This game proceeds exactly as Game $\mathbf{G}_{10}$ except that the simulation aborts if no InjCert occurred *and* there has been a Send query containing valid $\eta_{\mathcal{C}}$ which has not been previously returned by Δ on behalf of the browser (i.e., if $\mathcal{A}$ outputs a successful forgery for $\eta_{\mathcal{C}}$). Thus,

$$|\Pr[\mathsf{Win}_{11}] - \Pr[\mathsf{Win}_{10}]| \leq q\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa).$$

Game $\mathbf{G}_{12}$. [*Indistinguishability of D*] The only difference to Game $\mathbf{G}_{11}$ is that if no InjCert occurred then the simulator computes $D := Enc_{k_1}(\beta)$ using some additional randomly chosen $\beta \neq F_{\mathcal{C}}|\eta_{\mathcal{C}}$; otherwise D is computed as specified in the protocol. The purpose of this game is to show that the symmetric encryption protects secrecy of the transmitted encrypted and authenticated client finished message $F_{\mathcal{C}}$. Due to the IND-CPA property of (Enc, Dec) we obtain

$$|\Pr[\mathsf{Win}_{12}] - \Pr[\mathsf{Win}_{11}]| \leq q\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(\kappa).$$

Note that this and the previous games exclude replay and forgery attacks on D resulting in the acceptance by $\mathcal{S}$.

Game $\mathbf{G}_{13}$. [*Hash collision of h_2*] This game proceeds exactly as Game $\mathbf{G}_{12}$ except that the simulation aborts if during the interaction Δ computes the same hash value $h_2 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}}|F_{\mathcal{C}})$ in two different

sessions. Since as observed in Game $\mathbf{G}_1$ to Game $\mathbf{G}_3$ and Game $\mathbf{G}_{10}$ the input of h_2 is fresh for each new session the probability of such collision in this game is given by the collision-resistance of Hash, i.e.,

$$|\Pr[\mathsf{Win}_{13}] - \Pr[\mathsf{Win}_{12}]| \leq q\mathsf{Succ}^{coll}_{\mathtt{Hash}}(\kappa).$$

Game $\mathbf{G}_{14}$. [*Pseudo-randomness of $F_{\mathcal{S}}$*] This game proceeds exactly as Game $\mathbf{G}_{13}$ except that if no InjCert occurred then the simulator chooses $F_{\mathcal{S}}$ at random instead as computing it using PRF. Due to the pseudo-randomness of PRF we obtain

$$|\Pr[\mathsf{Win}_{14}] - \Pr[\mathsf{Win}_{13}]| \leq q\mathsf{Adv}^{prf}_{\mathtt{PRF}}(\kappa).$$

Note, this game implies that $F_{\mathcal{S}}$ does not leak any information about k_m. Note also, this and the previous games exclude replay and forgery attacks on $A, B, C|\sigma_{\mathcal{C}}$, and $F_{\mathcal{S}}$, since the pseudo-random function is instantiated with fresh values, i.e., h_2, and different labels.

Game $\mathbf{G}_{15}$. [*Forgery of $\eta_{\mathcal{S}}$*] This game proceeds exactly as Game $\mathbf{G}_{14}$ except that the simulation aborts if no InjCert occurred *and* there has been a Send query containing valid $\eta_{\mathcal{S}}$ which has not been previously returned by Δ on behalf of the server (i.e., $\mathcal{A}$ outputs a successful forgery for $\eta_{\mathcal{C}}$), thus

$$|\Pr[\mathsf{Win}_{15}] - \Pr[\mathsf{Win}_{14}]| \leq q\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa).$$

Game $\mathbf{G}_{16}$. [*Indistinguishability of E*] The only difference to Game $\mathbf{G}_{15}$ is that if no InjCert occurred then the simulator computes $D := Enc_{k_1}(\gamma)$ using some additional randomly chosen $\gamma \neq F_{\mathcal{S}}|\eta_{\mathcal{S}}$; otherwise E is computed as specified in the protocol. The purpose of this game is to show that the symmetric encryption protects secrecy of the transmitted encrypted and authenticated server finished message $F_{\mathcal{S}}$. Due to the IND-CPA property of (Enc, Dec) we obtain

$$|\Pr[\mathsf{Win}_{16}] - \Pr[\mathsf{Win}_{15}]| \leq q\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(\kappa).$$

Note, this and the previous games exclude replay and forgery attacks on E resulting in the acceptance by $\mathcal{B}$.

Game $\mathbf{G}_{17}$. [*Forgery of $\mu_{\mathcal{S}}$*] This game proceeds exactly as Game $\mathbf{G}_{16}$ except that the simulation aborts if no InjCert occurred *and* there has been a Send query containing valid $\mu_{\mathcal{S}}$ which has not been previously returned by Δ on behalf of the server (i.e., if $\mathcal{A}$ outputs a successful forgery for $\mu_{\mathcal{S}}$). Thus,

$$|\Pr[\mathsf{Win}_{17}] - \Pr[\mathsf{Win}_{16}]| \leq q\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa).$$

Game $\mathbf{G}_{18}$. [*Indistinguishability of F*] The only difference to Game $\mathbf{G}_{17}$ is that if no InjCert occurred then the simulator computes $F := Enc_{k_1}(\delta)$ using some additional randomly chosen $\delta \neq (w|\mu_{\mathcal{S}})$; otherwise F is computed as specified in the protocol. The purpose of this game is to show that the symmetric encryption protects secrecy of the transmitted encrypted authenticator w. Due to the IND-CPA property of (Enc, Dec) we obtain

$$|\Pr[\mathsf{Win}_{18}] - \Pr[\mathsf{Win}_{17}]| \leq q\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(\kappa).$$

Note, this and the previous games exclude replay and forgery attacks on F resulting in the acceptance by $\mathcal{B}$.

Game $\mathbf{G}_{19}$. [*Forgery of $\mu_{\mathcal{C}}$*] This game proceeds exactly as Game $\mathbf{G}_{18}$ except that the simulation aborts if no InjCert occurred *and* there has been a Send query containing valid $\mu_{\mathcal{C}}$ which has not been previously returned by Δ on behalf of the client (i.e., if $\mathcal{A}$ was able to output a successful forgery for $\mu_{\mathcal{C}}$). Thus,

$$|\Pr[\mathsf{Win}_{19}] - \Pr[\mathsf{Win}_{18}]| \leq q\mathsf{Succ}^{wuf-cma}_{\mathtt{HMAC}}(\kappa).$$

Game $\mathbf{G}_{20}$. [*Indistinguishability of G*] The only difference to Game $\mathbf{G}_{19}$ is that if no InjCert occurred then the simulator computes $G := Enc_{k_1}(\epsilon)$ using some additional randomly chosen $\epsilon \neq pw|\mu_{\mathcal{C}}$; otherwise G is computed as specified in the protocol. The purpose of this game is to show that if no certificate injection took place then $\mathcal{A}$ does not obtain any information about the encrypted password pw. Due to the IND-CPA property of (Enc, Dec) we get

$$|\Pr[\mathsf{Win}_{20}] - \Pr[\mathsf{Win}_{19}]| \leq q\mathsf{Adv}^{ind-cpa}_{(Enc,Dec)}(p_3(\kappa)).$$

Note, this and the previous games exclude replay and forgery attacks on G resulting in the acceptance by $\mathcal{S}$.

Game $\mathbf{G}_{21}$. [*Random Guess of the Authenticator w*] This game proceeds exactly as Game $\mathbf{G}_{20}$ except that the simulation aborts if during the interaction there is a client instance which accepts but there exists no partnered server instance, i.e., the first condition for the adversarial success from Definition 3 is satisfied. Note that according to the protocol specification $\mathcal{U}$ accepts after having recognized the authenticator w. Obviously the adversary succeeds only if $\mathcal{B}$ has received a Send query containing a message of the form $E|F$ which has not been previously returned by $\mathcal{S}$ such that decryption of F to $(w'|\mu_{\mathcal{S}})$ and rendering of w' results in the acceptance by $\mathcal{U}$. Having excluded forgeries of $\mu_{\mathcal{S}}$ in Game $\mathbf{G}_{17}$ we follow that the success probability of $\mathcal{A}$ is conditioned by the occurrence of InjCert (by which $\mathcal{A}$ learns k_1 and k_2) and is given by the probability of $\mathcal{A}$ to find some authenticator $w \in \mathcal{W}$ excluding the subset of human-indistinguishable authenticators $\mathcal{W}^*$ which will then be visualized to $\mathcal{U}$. Obviously, the success probability of $\mathcal{A}$ in finding such HPA depends on the size of $\mathcal{W}^*$. Thus, this is precisely the point in our security analysis of BBMA where human skills to distinguish the authenticators become important. We get

$$|\Pr[\mathsf{Win}_{21}] - \Pr[\mathsf{Win}_{20}]| \leq \frac{q|\mathcal{W}|}{|\mathcal{W}^*|}.$$

Note, this excludes attacks resulting in the impersonation of the server towards the user. Hence, $\mathcal{A}$ wins in this game if there is a server instance which accepts but no partnered client instance, i.e., if the second condition from Definition 3 is satisfied. Note, $\mathcal{S}$ accepts only after having verified the authenticity of $\mu_{\mathcal{C}}$ and that the decrypted password pw' matches the one shared with $\mathcal{U}$. Having excluded replays and forgeries of $\mu_{\mathcal{C}}$ in Game $\mathbf{G}_{19}$ and Game $\mathbf{G}_{20}$ we follow that there must be a client instance which sends this message. This also implies that the mentioned client instance should be using the same session id as the server instance, thus is partnered; otherwise, the server instance would have aborted upon the verification of the client's signature $\sigma_{\mathcal{C}}$. Hence, $\Pr[\mathsf{Win}_{21}] = 0$.

Considering Games $\mathbf{G}_1$ to $\mathbf{G}_{21}$ we obtain the desired inequality. □

4 Realization

In our realization of BBMA the TLS protocol is first executed in the bilateral authentication mode to establish a secure channel; then, HumanAuth and HumanResponse are exchanged. Recall, HumanAuth contains the authenticated and encrypted human authenticator whereas HumanResponse contains the authenticated and encrypted user password. Thus, the authenticator and password are protected by the message authentication and encryption mechanisms, using the keys provided by the TLS record layer. The flexibility of choosing the higher order protocol makes this realization especially interesting. Processing and displaying of the authenticator is modular and outsourced to the Web application. This allows to use authenticators that are personally tailored for a user. By personally, we mean the way in which the authenticators are presented to the user. Since technically sophisticated users might opt for classical Web authentication, our realization makes it feasible to downgrade to TLS in server authentication mode only.

We used HTTP over TLS in client authentication mode, executing a stand-alone Web application (in order to avoid cross-site-scripting attacks) that is written in the widely adopted script language PHP. In order to avoid a performance penalty for servers that have a large-scale penetration a TLS accelerator may be used. The protocol consists of two stages. In the register stage the user opts for an image and generates a client certificate as defined in PKCS#10. In the login stage, the image is displayed while the user is requested to enter a password into a Web form. A demonstration is available at [32].

As with any password-based key exchange protocol, we assume that there is no (active) adversary who intercepts the setup of credentials. (In the setup phase, the protocol is vulnerable to man-in-the-middle attacks.). When the user requests the Web site (Fig. 2), she may opt for registration. (Here the server may already test if a client certificate is present in the browser by sending a CertificateRequest message.) Then, she has to enter an email address, a username and password, and upload an image.

If no client certificate is stored in the browser, the server may trigger the generation of a public key pair and a PKCS#10 certificate signing request. (For the protocol to work, it would be sufficient that the browser generates a self-signed certificate. However, this is not implemented in modern browsers.) Upon receiving the

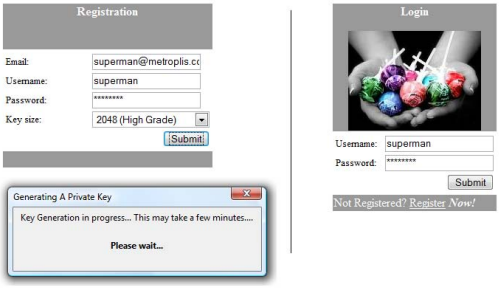

Fig. 2. Registration (left), Login (right)

PKCS#10 request, the server issues a certificate. Since the server is the issuer, no third party is required. The certificate is provided with attributes chosen by the server in order to relieve the user from filling out the (security) parameters. Finally, the server displays a link where the user downloads the client certificate. The client certificate's fingerprint in addition to the credentials are stored in the server's database and used to identify the user in the login stage. The overhead to install the client certificate is minimal. Today's browsers provide an easy-to-use interface. For instance, in the current release of Firefox the user has to click a link in order to store the certificate; no more interaction is required.

Upon requesting the login page (Fig. 2), the server checks that the browser is in possession of a valid client certificate as specified by TLS (`CertificateRequest`). Then, it displays the corresponding image above the login form (`HumanAuth`); otherwise, the protocol halts and the user is prevented from seeing the login page (and the authenticator). In order to get access to the site, the user has to enter username and password (`HumanResponse`). The messages `HumanAuth` and `HumanResponse` are authenticated and then encrypted by the TLS record layer. However, TLS-protected messages are secured while in transit on the network; after reception, the message plaintext as recovered by the record layer is forwarded to the browser where it can persist in the cache. In order for preventing the adversary from revealing cache information and disclosing the HPA we set up the server using the HTTP cache-response-directive [20, Sect. 14.9.2] so that the browser avoids caching the `HumanAuth` message.

5 Conclusion

Authenticating the user on the Web is an essential primitive and target to various attacks. We have introduced and analyzed a browser-based authentication protocol that makes weakest assumptions on users' skills. They have to detect a human perceptible authenticator. The protocol is specifically designed for those users who are security-unaware and used to evaluate Web sites through easy-to-recognize indicators. It remains an open question, if indeed all users detect the human perceptible authenticator. This requires to conduct usability experiments that specify the choice of "good" authenticators and keep the human fuzziness low. The presented protocol makes first steps towards bridging the gap between protocols that are provably secure, interfaced to users who are prone to errors, and implementable within the design constraints of standard browsers.

Interesting approaches to make `BBMA` more lightweight include the automatic generation of self-signed X.509 certificates in the browser or modifications of the mature browser security model that allows for securely handling cookies. Another interesting challenge is a cryptographic compiler that composes browser-based protocols in a general and provably secure sense. A last issue is to model alternative human skills such as the ability to solve human-hard puzzles (e.g., CAPTCHAs). First steps towards a formal treatment of solving such puzzles have been recently made by Canetti, Halevi, and Steiner [14].

Acknowledgement

This work has been partially supported by the European Commission under contract IST-2002-507932 through the ECRYPT Network of Excellence.

References

1. M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in tls. In *ASIACCS'06*, pages 35–45. ACM, 2006.
2. M. Abdalla, O. Chevassut, and D. Pointcheval. One-time verifier-based encrypted key exchange. In *PKC'05*, volume 3386 of *LNCS*, pages 47–64. Springer, 2005.
3. M. Abdalla and D. Pointcheval. Simple Password-Based Encrypted Key Exchange Protocols. In *CT-RSA'05*, volume 3376 of *LNCS*, pages 191–208. Springer, 2005.
4. C. Allen and T. Dierks. The TLS Protocol — Version 1.1. Internet proposed standard RFC 4346, 2006.
5. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *ASIACRYPT'00*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.
7. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.
8. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS'93*, pages 62–73. ACM, 1993.
9. M. Bellare and P. Rogaway. The AuthA Protocol for Password-Based Authenticated Key Exchange. *Contributions to IEEE P1363*, 2000. `http://grouper.ieee.org/groups/1363/passwdPK/contributions.html`.
10. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary atttacks and password file compromise. In *ACM CCS'93*, pages 244–250. ACM, 1993.
11. V. Boyko, P. D. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange using Diffie-Hellman. In *EUROCRYPT'00*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Security Proofs for an Efficient Password-Based Key Exchange. In *ACM CCS'03*, pages 241–250. ACM, 2003.
13. E. Bresson, O. Chevassut, and D. Pointcheval. New Security Results on Encrypted Key Exchange. In *PKC'04*, volume 2947 of *LNCS*, pages 145–158. Springer, 2004.
14. R. Canetti, S. Halevi, and M. Steiner. Mitigating Dictionary Attacks on Password-Protected Local Storage. In *CRYPTO'06*, volume 4117 of *LNCS*, pages 160–179. Springer, 2006.
15. S. Chiasson, P. C. van Oorschot, and R. Biddle. Graphical Password Authentication using Cued Click Points. In *ESORICS'07*, volume 4734 of *LNCS*, pages 359–374. Springer, 2007.
16. R. Dhamija and J. D. Tygar. The Battle against Phishing: Dynamic Security Skins. In *SOUPS'05*, pages 77–88. ACM, 2005.
17. R. Dhamija, J. D. Tygar, and M. A. Hearst. Why Phishing Works. In *CHI'06*, pages 581–590. ACM, 2006.
18. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.1. RFC 4346, IETF, 2006. Proposed Standard.
19. C. Ellison. Ceremony Design and Analysis. Cryptology ePrint Archive, Report 2007/399, 2007.
20. R. T. Fielding, J. Gettys, J. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Technical Report 2616, Internet Engineering Task Force, June 1999.
21. I. Giang. SSL Phishing, Microsoft Moves to Brand, and Nyms. In *FC'06*, 14 February 2006. `https://www.financialcryptography.com/mt/archives/000654.html`.
22. T. Groß. Security Analysis of the SAML Single Sign-On Browser/Artifact Profile. In *ACSAC'03*. IEEE Computer Society, 2003.
23. T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser Model for Security Analysis of Browser-Based Protocols. In *ESORICS'05*, volume 3679 of *LNCS*, pages 489–508. Springer, 2005.
24. T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Proving a WS-Federation Passive Requestor Profile with a Browser Model. In *ACM SWS'05*, pages 54–64. ACM, 2005.
25. A. Herzberg. Why Johnny Can't Surf (Safely)?, 2007. (Personal communication, work in progress).
26. M. Jakobsson and S. Myers. Delayed Password Disclosure. In *ACM DIM'07*, pages 17–26. ACM, 2007.
27. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange using Human-Memorable Passwords. In *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 475–494. Springer, 2001.
28. J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-Only Key Exchange Protocols. In *SCN'02*, volume 2576 of *LNCS*, pages 29–44. Springer, 2002.
29. D. Kormann and A. Rubin. Risks of the Passport Single Sign-On Protocol. *Computer Networks*, 33(1–6):51–58, 2000.
30. P. MacKenzie. The PAK Suite: Protocols for Password-Authenticated Key Exchange. Technical Report 2002-46, DIMACS, 2002.

31. B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE S&P'01*, pages 184-201, 2001.
32. Proof of Concept Implementation of BBMA, 2007. http://www.demo.nds.rub.de/bbma.
33. V. Shoup. OAEP Reconsidered. *Journal of Cryptology*, 15(4):223–249, 2002.
34. V. Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. Cryptology ePrint Archive, Report 2004/332, 2006.
35. C. Soghoian and M. Jakobsson. A Deceit-Augmented Man in the Middle Attack against Bank of America's SiteKey Service, 2007. http://paranoia.dubfire.net/2007/04/deceit-augmented-man-in-middle-attack.html.
36. M. Steiner, P. Buhler, T. Eirich, and M. Waidner. Secure Password-Based Cipher Suite for TLS. *ACM Trans. on Inf. and Sys. Sec*, 4(2):134–157, 2001.
37. A. O. Stuart Schechter, Rachna Dhamija and I. Fischer. The Emperor's New Security Indicators. In *IEEE S&P'07*, pages 51–65. IEEE CS, 2007.
38. X. Suo, Y. Zhu, and G. S. Owen. Graphical Passwords: A Survey. In *ACSAC'05*, pages 463–472. IEEE CS, 2005.
39. W3C. Document Object Model (DOM), 2005. http://www.w3.org/DOM.