# A Browser-Based Kerberos Authentication Scheme

Sebastian Gajek[1], Tibor Jager[1], Mark Manulis[2], and Jörg Schwenk[1]

[1] Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
[2] UCL Crypto Group
Louvain-la-Neuve, Belgium
{sebastian.gajek,tibor.jager,joerg.schwenk}@nds.rub.de,
manulis@crypto.ucl.be

**Abstract.** When two players wish to share a security token (e.g., for the purpose of authentication and accounting), they call a trusted third party. This idea is the essence of Kerberos protocols, which are widely deployed in a large scale of computer networks. Browser-based Kerberos protocols are the derivates with the exception that the Kerberos client application is a commodity Web browser. Whereas the native Kerberos protocol has been repeatedly peer-reviewed without finding flaws, the history of browser-based Kerberos protocols is tarnished with negative results due to the fact that subtleties of browsers have been disregarded. We propose a browser-based Kerberos protocol based on client certificates and prove its security in the extended formal model for browser-based mutual authentication introduced at ACM ASIACCS'08.

## 1 Introduction

**Motivation.** An immediate goal browser-based protocols strive for to meet is user authentication and access control to services or private information. A widely adopted approach is to use TLS in server authenticated mode and execute a protocol on top, where the user enters a password in a Web form. To this end, the user has to memorize a plethora of passwords. The problem with passwords is that the user frequently forgets about them. Otherwise, it would be unnecessary to include a "'Forgot your password"' link in a Web application). Furthermore, the user tends to recurrently choose the same low-entropy password, thus making offline dictionary attacks feasible. In order to alleviate the problem, 3-party authentication protocols have been introduced where a trusted third party is asked to issue a token that is valid for a fixed time period and permits access to some service. A pioneer and quite successful protocol for closed networks that emulates the task is the widely adapted Kerberos protocol [1]. Here, the Kerberos server issues an token in form of a ticket that the client may redeem to authenticate to a server. Related protocols that adapt the idea are Microsoft's Passport and its successor Cardspace, the Security Assertion Markup Language (SAML), the Liberty Alliance project, the Shibboleth project for university identity federation, or WS-Federation, whereby SAML is an open protocol standard and basis for Liberty and Shibboleth.

The migration of the Kerberos idea to open networks, in particular, the Internet is peppered with problems (see Section 2). In particular, the problem with the browser-based realization is that some assumptions have been made which are unfounded today. Most notably, the user is assumed to determine the authenticity of a Web server on the basis of server certificate and the Domain Name System (DNS) is assumed to be an authentic host name resolution protocol. The first clashes with usability studies, showing that the average Internet user neither understands server certificates nor perceives the security indicators in commodity browsers [8,23]. The latter is a crucial factor for the enforcement of the *Same Origin Policy (SOP)*. This security policy, which is universally supported by browsers, loosely states that Web objects are accessible by other Web objects under the condition that they are from the same domain. However, many attacks against the domain name resolution exist, ranging from Javascript code that alters a router's configuration [22] to large scale DNS attacks [15]. A related attack vector arises from cross Site scripting (XSS) attacks [16] where the adversary injects some malicious code into the response of the application server. Since the code is in the same security context, the SOP does not apply. Consequently, malicious code can break free and invoke arbitrary browser scripting functionalities.

**Our Contribution.** We solve the above problems by presenting a Browser-based Kerberos-like protocol, in the following denoted by `BBKerberos`, that is close to the native Kerberos scheme. Our `BBKerberos` protocol.

– combines authentication with key agreement: The user authenticates to the Kerberos server through a TLS client certificate in addition to (optional) passwords. The Kerberos server issues an authentication ticket for the application server which is concealed within an HTTP cookie. The cookie is transferred in another TLS session whereby the browser authenticates to the server using the same client certificate. Thus in both TLS connections key agreement is linked to authentication through the client certificate.
– binds the Kerberos ticket to a specific browser. The ticket is linked to the client certificate. Thus, attacks that enable adversaries to extract the cookie carrying the Kerberos ticket (e.g. XSS, Pharming) work. However, the attacker is now unable to use the cookie. The reason is that the application server learns from the underlying TLS protocol session that the client is a legitimate owner of the client certificate (note that in the TLS protocol the client authenticates to the Kerberos server by signing a protocol transcript and proving ownership of the corresponding private key). Here, we make use of the fact that any feasible adversary does not have access to the long-term secrets for the TLS layer. It has only access to secrets on application layer. Conversely, the application server may extract the public key from the TLS layer and verify the cookie.
– provides secure single sign-on. The sign-on ticket may also be reused in a federation of application servers. Application servers need to establish a

TLS-protected channel where the client conveys a certificate matching the cookie binding and where the client demonstrates that it knows the corresponding secret key.

**Organization.** The remaining sections are structured as follows. We review related work in Section 2. In Section 3, we present the formal security model for browser-based authentication protocols, and use it to analyze `BBKerberos` in Section 4. Finally, we conclude in Section 5.

## 2 Related Work

The native Kerberos protocol has been studied without finding severe flaws [6,3]. By contrast, few browser-based Kerberos protocols have been subject to rigorous security analysis. The first attempt to disburden from a client application and employ a browser's capabilities has led to Microsoft's Passport protocol. Unfortunately, it turned out that the protocol had some vulnerabilities [17]. Korman and Rubin show that the adversary is able to steal the ticket granting ticket cookie by mounting a DNS attack. In addition to the Passport analysis due to [17], Groß [12] analyzes SAML, an alternative identity management protocol, and shows that the protocol is vulnerable to adaptive attacks where the adversary intercepts the authentication token contained in the URL. Groß makes use of the fact that browsers add the URL in a referrer tag into a HTTP response when they are redirected. Hence, a man-in-the-middle adversary signaling the browser to redirect the request to a rogue server retrieves the authentication token from the referrer tag. The previously described flaw in the SAML protocol has led to a revised version of SAML. Groß and Pfitzmann analyzed this version, again finding the need for improvements [13]. Similar flaws have been found in the analysis of the Liberty single sign on protocol [20] due to Pfitzmann and Waidner. The authors point out some weaknesses in presence of man-in-the-middle attacks. Gajek, Schwenk and Xuan show that Microsoft's identity metasystem CardSpace is vulnerable to dynamic pharming attacks and enables the adversary to steal a security token [11].

## 3 Modeling `BBKerberos` with Client Certificates

In this section we refine the original security model for mutual authentication from [10] to capture adversarial queries where the adversary access the DOM. Similar to [10] we consider an active probabilistic polynomial time (PPT) adversary who interacts with involved parties through queries and controls all the communication.

### 3.1 Protocol Participants and Communication Model

**Server, Browser, Authentication Server.** We consider the *server* $\mathcal{S}$, the *browser* $\mathcal{C}$, and the *authentication server* (e.g. Kerberos server) $\mathcal{K}$ as participants of a `BBKerberos` protocol $\Pi$. We make the weak security assumptions on $\mathcal{C}$ to represent a practical setting, namely that web objects stored in $\mathcal{C}$ (see below) can be accessed by the adversary via the DOM (by mounting DNS or XSS attacks). However, the adversary does not have the privilege to access the private key of the Web browser. Since we do not consider attacks on either $\mathcal{S}$ or $\mathcal{K}$, we model both servers as being secure.

The browser $\mathcal{C}$ is modeled as a PPT machine that exchanges protocol messages with $\mathcal{S}$ and $\mathcal{K}$ through physical communication links.We further assume that the authentication server $\mathcal{K}$ is able to identify the browser $\mathcal{C}$. This is achieved through client certificates. In high security applications trust in the client certificate must once be established, using some out-of-band mechanisms.

*Remark 1.* We cannot expect an average Internet user acting "behind" $\mathcal{C}$ to properly identify $\mathcal{K}$. In order to relax the assumption, the construction based on human perceivable authenticator, proposed in [10], may be employed in our protocol.

**Modeling Browser DOM Access.** The browser plays the role of a messenger and transports the messages from the authentication server $\mathcal{K}$ to the application server $\mathcal{S}$ (and vice versa), 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 web objects) that $\mathcal{K}$ wishes to send to $\mathcal{S}$ and stores the information according to the browser's state $\Psi \in \{0,1\}^{\lambda_2(\kappa)}$ to $\mathcal{U}$ as a Web object. (Here and in the following, $\lambda_i : \mathbb{N} \rightarrow \mathbb{N}, i \in [1,2]$ is a polynomial and $\kappa$ is the security parameter). State $\Psi$ denotes the browser's configuration for processing the retrieved message that may be altered by querying the browser's DOM[1] 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 the use of Web scripting languages (e.g., Javascript). One important security policy is the *same origin* policy. This policy which is universally supported in browsers, ensures that there is no communication between pages that have different domains. This includes 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.

*Remark 2.* There are different methods to send messages from one server to another through the browser:

– The most popular mechanism are HTTP cookies. Cookies are short text messages which contain (name, value) pairs. Cookies can be persistent, i.e. they can be stored in the browser for a certain time, and they can be set for a target address that consists of a domain and a path within that domain. Cookies can directly be sent to the destination server by combining the

[1] Document Object Model, see [24] for details.

Set-Cookie directive in the HTTP header with a redirect status code from the sending web server. However, for security reasons this mechanism is restricted to servers from the same domain. Thus in the general case we must use other mechanisms to transport data from $\mathcal{K}$ to $\mathcal{S}$ across Domain boundaries (e.g. HTTP POST and GET of hidden form fields).

– Dynamically generated URLs are hyperlinks within a HTML document, where part of the URL is used to encode the data to be transmitted. They are not persistent and will be deleted when the HTML document is closed. Since hyperlinks are treated differently according to their position within the HTML document, they can be used to send data directly (e.g. when used within an image tag), or only after some user action.
– Hidden HTML forms are normal HTML forms that a hidden from the user, i.e. the form will not be displayed. This only makes sense if the form is already filled with data, and this data can be transported to another server triggered by events. Data can be sent as POST data, or as a GET query string, and may later be persistently stored by the receiving server in form of a HTTP cookie.

All the above methods support the immediate transmission of data between two servers through the browser as an intermediary. They further have in common that the data is stored for some time within the DOM of the browser, and is thus vulnerable to DNS and XSS attacks. In the following, we will use HTTP cookies to transport data from the authentication server $\mathcal{K}$ to the application server $\mathcal{S}$ and vice versa. Our proof applies to other transport mechanisms as well.

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

**Execution States.** Each instance $[\mathcal{C}, sid_{\mathcal{C}}]$, $[\mathcal{K}, sid_{\mathcal{K}}]$ 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 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. In case that the execution was not successful (due to failures) instances terminate without accepting, i.e., they *abort*.

## 3.2 Security Model

In the following we specify attacks and security goals for `BBKerberos` protocols from the perspective of fixed identities $\mathcal{C}$, $\mathcal{S}$ and $\mathcal{K}$.

**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 foils browsers' same origin policy. Then, the adversary has access to the DOM model. That is, the adversary has access to the browser's chrome, cache, cookies, and history; specifically, we assume that the cookie containing the Kerberos ticket is known to the adversary. However, the adversary is prevented from opening, creating or deleting a file from the operating system; thus he can not read the browser's private key from disk or from memory.
- The adversary issues public keys which $\mathcal{C}$ accepts. There is no trusted third party in the sense of a trusted CA. Hence, 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).
- The adversary is unable to corrupt $\mathcal{C}$. Note that in this model we do not deal with malware attacks against the browser and server, therefore, do not consider the case where $\mathcal{A}$ reveals the ephemeral and longterm secrets stored inside $\mathcal{C}$.
- The adversary is unable to corrupt $\mathcal{S}$ or $\mathcal{K}$. Note also that in this model we do not deal with malware attacks against the servers. This means that the adversary is excluded from revealing the ephemeral and longterm secrets stored inside $\mathcal{S}$ or $\mathcal{K}$.

**Adversarial Queries.** The adversary $\mathcal{A}$ can participate in the actual protocol execution via the following queries:

- $\mathsf{Execute}(\mathcal{C}, P)$ $(P \in \{\mathcal{K}, \mathcal{S}\})$: This query models passive attacks where the adversary $\mathcal{A}$ eavesdrops the execution of the new protocol session between $\mathcal{C}$ and $P$. $\mathcal{A}$ is given the corresponding transcript.
- $\mathsf{Invoke}(\mathcal{C}, P)$ $(P \in \{\mathcal{K}, \mathcal{S}\})$: $\mathcal{C}$ starts the protocol execution with the new instance of $P$ and $\mathcal{A}$ obtains the first protocol message returned by $\mathcal{B}$ (which is usually generated on some input received from $\mathcal{C}$, e.g., the entered URL).
- $\mathsf{Send}(P, m)$: This query models active attacks where $\mathcal{A}$ sends a message to some instance of $P \in \{\mathcal{C}, \mathcal{K}, \mathcal{S}\}$. The adversary $\mathcal{A}$ receives the response which $P$ would generate after having processed the message $m$ according to the protocol specification (note that the response may be an empty string if $m$ is unexpected).

- $\mathsf{RevealDOM}(\mathcal{C})$: This query (which is refined in comparison to [10]) models attacks which reveal information stored with the browser's DOM, i.e. $\Psi$. Technically, the adversary queries for the authentication token (as stored in a cookie). Note that $\mathsf{RevealDOM}(\mathcal{C})$ enables the adversary to *read* the DOM. However, the adversary is prevented from writing the DOM, i.e. subverting the protocol execution by injecting malicious script code.

**Protocol Execution in the Presence of $\mathcal{A}$.** By asking the $\mathsf{Execute}(\mathcal{C}, P)$ ($P \in \{\mathcal{K}, \mathcal{S}\}$) query $\mathcal{A}$ obtains the transcript of the complete protocol execution between new instances of $\mathcal{C}$ and $P$ without being able to perform any further actions during this execution. We assume that at most $q_{ex}$ such queries can be asked during the attack. On the other hand, if $\mathcal{A}$ wishes to actively participate in the execution of $\Pi$ then it can ask a special invocation query $\mathsf{Invoke}(\mathcal{C}, P)$ implying that a new instance of $\mathcal{C}$ starts the protocol execution with the new instance of $P$ using the associated instance of browser $\mathcal{C}$. $\mathcal{A}$ then obtains the first protocol message returned by $\mathcal{C}$. Active participation of $\mathcal{A}$ is defined further through at most $q_s$ $\mathsf{Send}$ queries. We also assume that $\mathcal{A}$ can ask at most $q_{in}$ $\mathsf{Invoke}$ and $q_r$ $\mathsf{RevealDOM}$ queries. Thus, the total number of queries which can be asked by the PPT adversary during the duration of the attack is upper-bounded by $q := q_{ex} + q_{in} + q_s + q_r$.

**Correctness and Authentication.** The following definition specifies the correctness requirement for BBKerberos protocols.

**Definition 1 (Correctness).** *A browser-based Kerberos protocol $\Pi$ is* correct *if each* $\mathsf{Execute}(\mathcal{C}, P)$ *query where $P \in \{\mathcal{K}, \mathcal{S}\}$ results in two instances, $[\mathcal{C}, sid_{\mathcal{C}}]$ and $[P, sid_P]$ which are partnered ($sid_{\mathcal{C}} = sid_P$) and accept prior to termination.*

In the following we define the main security goal of browser-based Kerberos protocols, namely the requirement of authentication of $\mathcal{C}$ to $\mathcal{S}$ which is implied by an authentic communication between $\mathcal{C}$ and $\mathcal{K}$. The adversary $\mathcal{A}$ wins when he succeeds in authenticating to either $\mathcal{K}$ or $\mathcal{S}$ as a legitimate $\mathcal{C}$.

**Definition 2 (Authentication of $\mathcal{C}$).** *Let $\Pi$ be a correct browser-based Kerberos protocol and* $\mathsf{Game}_{\Pi}^{bb-auth}(\mathcal{A}, \kappa)$ *the interaction between the instances of $\mathcal{C}$, $\mathcal{K}$ and $\mathcal{S}$ with a PPT adversary $\mathcal{A}$ who is allowed to query* $\mathsf{Execute}$, $\mathsf{Invoke}$, $\mathsf{Send}$, *and* $\mathsf{RevealDOM}$. *We say that $\mathcal{A}$* wins *if at some point during the execution for $sid'_{\mathcal{C}} \neq sid_{\mathcal{C}}$:*

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

*The maximum probability of this event (over all adversaries running within the security parameter $\kappa$, and all public keys $pk_{\mathcal{C}}$ registered with $\mathcal{K}$) is denoted* $\mathsf{Succ}_{\Pi}(\mathcal{A}, \kappa) = \max_{\mathcal{A}} |\Pr[\mathcal{A}$ *wins in* $\mathsf{Game}_{\Pi}^{bb-auth}(\mathcal{A}, \kappa)|$. *We say that a browser-based Kerberos protocol $\Pi$ provides* authentication *if this probability is a negligible function of $\kappa$.*

The first requirement ensures that $\mathcal{C}$ authenticates to the matching server $\mathcal{S}$; the second requirement ensures a matching conversation with $\mathcal{K}$. (It is important to note that the second requirement is an prerequisite in our protocol to achieve the first. Details follow.)

## 4 The BBKerberos Protocol

### 4.1 Building Blocks

**TLS Protocol.** A main pillar of BBKerberos is the mutually authenticated *key transport/key agreement* [2]. The security of the TLS protocol has already been analyzed with respect to certain cryptographic primitives or in an abstract term-algebra (see [9]). However, since we combine two TLS sessions with higher-layer protocols, the security analyses cited above are insufficient. We thus use a model similar to [5] to model the security requirements and the proof. We describe the protocol using the most common, RSA based variant. All other ciphersuites are of course possible as well.

**Cryptographic Primitives.** In order to be able to use the term "negligible" in a mathematically correct way, here and in the following let $p_i : \mathbb{N} \rightarrow \mathbb{N}, i \in [1, 5]$ be polynomials, and let $\kappa \in \mathbb{N}$ be a security parameter. However, note that in practice many parameters in TLS are fixed in their length. As usual, we formalize the notion of an algorithm trying to solve a certain computational problem in order to compromise the security goals of our protocol and its building blocks by a probabilistic Turing machine running in time polynomial in $\kappa$ (PPT adversary).

In our BBKerberos protocol we make use of the (well-known) cryptographic primitives used by the cryptographic suites of the TLS protocol, namely:

– A *pseudo-random function* $\texttt{PRF} : \{0,1\}^{p_3(\kappa)} \times \{0,1\}^* \rightarrow \{0,1\}^*$. 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 $\mathsf{Adv}^{prf}_{\texttt{PRF}}(\kappa)$ we denote the maximum advantage over all PPT adversaries (running within security parameter $\kappa$) in distinguishing the outputs of PRF from those of a random function better than by a random guess.
– A *symmetric encryption schemes* 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 (running within security parameter $\kappa$) 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 (running within security parameter $\kappa$) in breaking the IND-CPA property of $(\mathcal{E}, \mathcal{D})$ better than by a random guess.

- A *collision-resistant hash function* $\texttt{Hash} : \{0,1\}^* \rightarrow \{0,1\}^{p_4(\kappa)}$. We denote by $\mathsf{Succ}^{coll}_{\texttt{Hash}}(\kappa)$ the maximum success probability over all PPT adversaries (running within security parameter $\kappa$) in finding a collision, i.e., a pair $(m, m') \in \{0,1\}^* \times \{0,1\}^*$ s.t. $\texttt{Hash}(m) = \texttt{Hash}(m')$.
- 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 $Ver$. By $\mathsf{Succ}^{euf-cma}_{(Sig,Ver)}(\kappa)$ we denote the maximum success probability over all PPT adversaries (running within security parameter $\kappa$) given access to the signing oracle in finding a forgery;
- The well-known *message authentication code* function HMAC which is believed to satisfy *weak unforgeability under chosen message attacks* (WUF-CMA) [4]. By $\mathsf{Succ}^{wuf-cma}_{\texttt{HMAC}}(\kappa)$ we denote the maximum success probability over all PPT adversaries (running within security parameter $\kappa$) given access to the tagging/verification oracle in finding a forgery.

## 4.2 Protocol Description

**Initialization Phase.** Before BBKerberos can be executed, a registration phase is necessary. During this phase, the following keys are exchanged/registered:

- The long-lived key $LL_\mathcal{C}$ stored in the credential store of the browser $\mathcal{C}$ consists of the private/public signature key pair $(sk_\mathcal{C}, cert_\mathcal{C})$; we assume that the corresponding public key $pk_\mathcal{C}$ is part of the certificate. This public key $pk_\mathcal{C}$ is registered with the Kerberos server $\mathcal{K}$ after some initial out-of-band authentication.
- The long-lived key $LL_\mathcal{S}$ consists of the private/public encryption key pair $(sk_\mathcal{S}, cert_\mathcal{S})$, and a symmetric encryption key $k_{\mathcal{KS}}$. The key $k_{\mathcal{KS}}$ has to be exchanged out-of-band between $\mathcal{K}$ and $\mathcal{S}$.
- Finally, the long-lived key $LL_\mathcal{K}$ consists of the private/public encryption key pair $(sk_\mathcal{K}, cert_\mathcal{K})$.

**Execution Phase.** In the following we briefly describe the execution of our BBKerberos protocol specified in Figures 1 and 2. We first give an overview of the protocol and then describe the TLS handshake,which is performed twice with different random values in detail.

1. **Initiate the Protocol.** The browser $\mathcal{C}$ initiates the protocol by requesting an $URL$ from the server $\mathcal{S}$ which requires authentication. The server $\mathcal{S}$ tells the browser $\mathcal{C}$ to connect to the Kerberos server $\mathcal{K}$ using TLS through a redirect status code.
2. **First TLS Handshake.** A first TLS handshake with client authentication is performed between $\mathcal{C}$ and $\mathcal{K}$. Both parties exchange certificates, so they know each other's public key. The identificator for the negotiated cryptographic key material is $sid_\mathcal{C}{=}r_\mathcal{C}|r_\mathcal{K}$. Symmetric keys are derived from the master secret $k_m$, which in turn is derived from the premaster secret $k_p$. This value

$k_p$ has been chosen randomly by $\mathcal{C}$, and has been sent to $\mathcal{K}$ encrypted with $pk_{\mathcal{K}}$. The browser $\mathcal{C}$ authenticates himself by signing a hash of previous messages. The Kerberos server $\mathcal{K}$ authenticates by computing a HMAC over all previous messages, using a key derived from $k_m$.

3. **Retrieving the Authentication Cookie.** After the TLS handshake, $\mathcal{K}$ knows that he is talking to $\mathcal{C}$ through a confidential and authentic channel. $\mathcal{K}$ now issues the ticket $t_{\mathcal{CS}}$, which is authenticated and encrypted by $\mathcal{K}$ using the shared symmetric key $k_{\mathcal{KC}}$ whereby the ticket is cryptographically linked to the client's public key $pk_{\mathcal{C}}$. The result is encoded as a text string $c$, sent to $\mathcal{C}$ using HTTP GET or POST, and stored persistently in the browser. The browser $\mathcal{C}$ sends $c$ whenever it thinks it is connected to $\mathcal{S}$.
4. **Second TLS Handshake.** A second TLS handshake with client authentication is performed between $\mathcal{C}$ and $\mathcal{S}$. Again both parties exchange certificates, so they know each other's public key. The new TLS session uses different session identifier $sid'_{\mathcal{C}}{=}r'_{\mathcal{C}}|r_{\mathcal{K}}$, and secret keys $k'_p$ and $k'_m$. Again the browser $\mathcal{C}$ authenticates by signing a hash of previous messages. The Kerberos server $\mathcal{K}$ authenticates by computing a HMAC over all previous messages, using a key derived from $k'_m$.
5. **Authenticating via Kerberos Cookie.** After a successful TLS handshake, browser $\mathcal{C}$ sends the value $c$ as GET or POST data (and later as a HTTP cookie) to $\mathcal{S}$. We only require the TLS tunnel to authenticate data sent from the browser $\mathcal{C}$; confidentiality is not needed. The server $\mathcal{S}$ validates the value $c$ using the key $k_{\mathcal{KS}}$, and if this validation is successful, it compares the public key contained in $c$ to the public key used to authenticate the browser $\mathcal{C}$. If this comparison is positive, he accepts the Kerberos ticket contained in $c$ and grants $\mathcal{C}$ access to the requested resource.

*TLS sessions in detail.* The TLS protocol with client authentication in order to establish a secure transport channel is the main component in our security analysis. Let $l_1$, $l_2$,$l_3$ and $l_4$ denote the publicly known *labels* specified in TLS for the instantiation of `PRF`. The TLS protocol proceeds as follows:

1. **ClientHello and ServerHello.** The browser $\mathcal{C}$ chooses his own *nonce* $r_{\mathcal{C}}$ of length $p_5(\kappa)$ at random and forwards it to $\mathcal{S}$ (`ClientHello`). In response $\mathcal{S}$ chooses his own random *nonce* $r_{\mathcal{S}}$ and a *TLS session identifier sid* of length $p_5(\kappa)$ and appends it to his 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. **Negotiate Key Material.** $\mathcal{C}$ 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 [2, Sect. 5]). The master secret is then used as secret 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

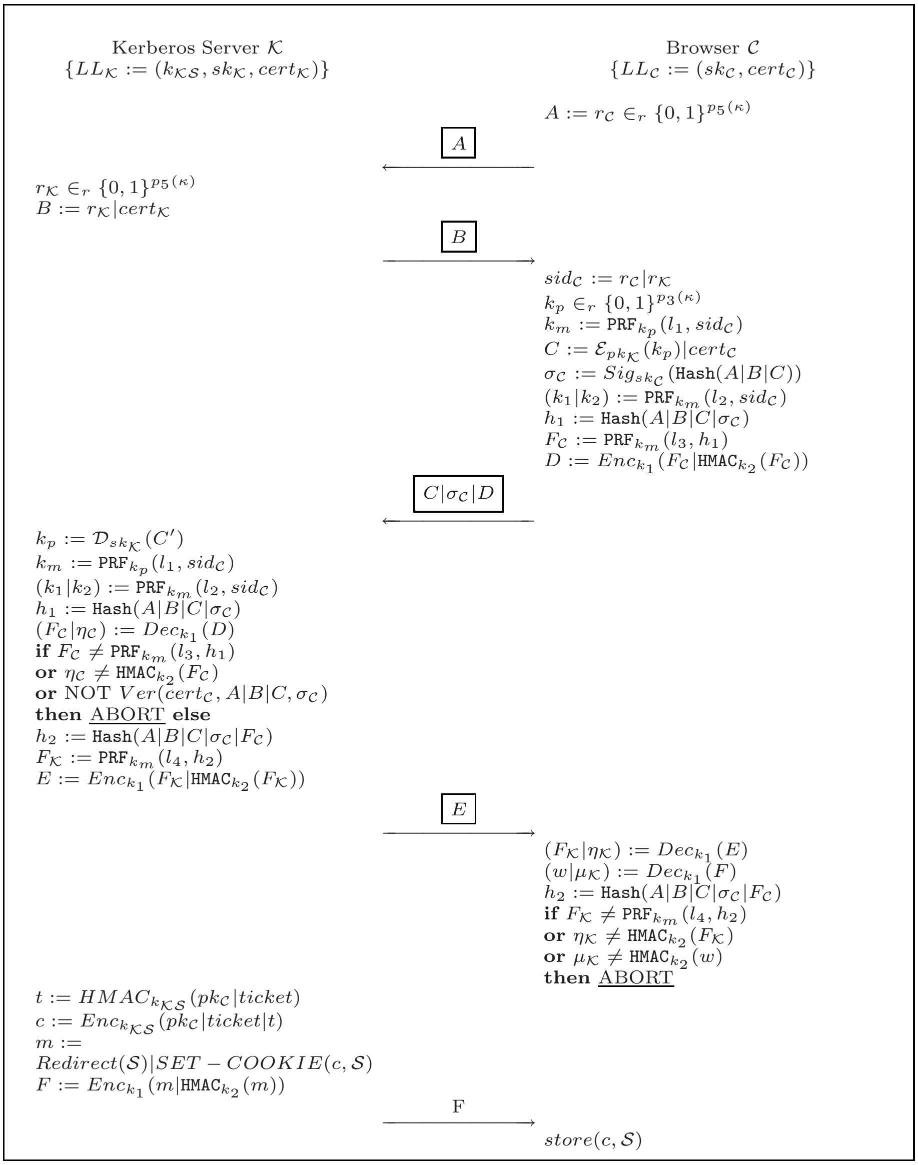
Kerberos Server $\mathcal{K}$
$\{LL_{\mathcal{K}} := (k_{\mathcal{KS}}, sk_{\mathcal{K}}, cert_{\mathcal{K}})\}$
Browser $\mathcal{C}$
$\{LL_{\mathcal{C}} := (sk_{\mathcal{C}}, cert_{\mathcal{C}})\}$
$A := r_{\mathcal{C}} \in_r \{0,1\}^{p5(\kappa)}$
$\boxed{A}$
$r_{\mathcal{K}} \in_r \{0,1\}^{p5(\kappa)}$
$B := r_{\mathcal{K}} | cert_{\mathcal{K}}$
$\boxed{B}$
$sid_{\mathcal{C}} := r_{\mathcal{C}} | r_{\mathcal{K}}$
$k_p \in_r \{0,1\}^{p3(\kappa)}$
$k_m := \mathtt{PRF}_{k_p}(l_1, sid_{\mathcal{C}})$
$C := \mathcal{E}_{pk_{\mathcal{K}}}(k_p) | cert_{\mathcal{C}}$
$\sigma_{\mathcal{C}} := Sig_{sk_{\mathcal{C}}}(\mathtt{Hash}(A|B|C))$
$(k_1|k_2) := \mathtt{PRF}_{k_m}(l_2, sid_{\mathcal{C}})$
$h_1 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}})$
$F_{\mathcal{C}} := \mathtt{PRF}_{k_m}(l_3, h_1)$
$D := Enc_{k_1}(F_{\mathcal{C}} | \mathtt{HMAC}_{k_2}(F_{\mathcal{C}}))$
$\boxed{C|\sigma_{\mathcal{C}}|D}$
$k_p := \mathcal{D}_{sk_{\mathcal{K}}}(C')$
$k_m := \mathtt{PRF}_{k_p}(l_1, sid_{\mathcal{C}})$
$(k_1|k_2) := \mathtt{PRF}_{k_m}(l_2, sid_{\mathcal{C}})$
$h_1 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}})$
$(F_{\mathcal{C}}|\eta_{\mathcal{C}}) := Dec_{k_1}(D)$
**if** $F_{\mathcal{C}} \neq \mathtt{PRF}_{k_m}(l_3, h_1)$
**or** $\eta_{\mathcal{C}} \neq \mathtt{HMAC}_{k_2}(F_{\mathcal{C}})$
**or** NOT $Ver(cert_{\mathcal{C}}, A|B|C, \sigma_{\mathcal{C}})$
**then** ABORT **else**
$h_2 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}}|F_{\mathcal{C}})$
$F_{\mathcal{K}} := \mathtt{PRF}_{k_m}(l_4, h_2)$
$E := Enc_{k_1}(F_{\mathcal{K}} | \mathtt{HMAC}_{k_2}(F_{\mathcal{K}}))$
$\boxed{E}$
$(F_{\mathcal{K}}|\eta_{\mathcal{K}}) := Dec_{k_1}(E)$
$(w|\mu_{\mathcal{K}}) := Dec_{k_1}(F)$
$h_2 := \mathtt{Hash}(A|B|C|\sigma_{\mathcal{C}}|F_{\mathcal{C}})$
**if** $F_{\mathcal{K}} \neq \mathtt{PRF}_{k_m}(l_4, h_2)$
**or** $\eta_{\mathcal{K}} \neq \mathtt{HMAC}_{k_2}(F_{\mathcal{K}})$
**or** $\mu_{\mathcal{K}} \neq \mathtt{HMAC}_{k_2}(w)$
**then** ABORT
$t := HMAC_{k_{\mathcal{KS}}}(pk_{\mathcal{C}} | ticket)$
$c := Enc_{k_{\mathcal{KS}}}(pk_{\mathcal{C}} | ticket | t)$
$m :=$
$Redirect(\mathcal{S}) | SET-COOKIE(c, \mathcal{S})$
$F := Enc_{k_1}(m | \mathtt{HMAC}_{k_2}(m))$
F
$store(c, \mathcal{S})$

**Fig. 1.** `BBKerberos` Protocol with TLS Client Authentication, Part 1. Boxed messages denote the standard TLS handshake.

and authenticate session messages exchanged between $\mathcal{C}$ and $\mathcal{S}$. [Remark: TLS specifies the generation of six session keys: A symmetric encryption key, a MAC key, and an IV for block ciphers only (both for client-to-server and for server-to-client communication). For simplicity, we denote $k_1$ as the encryption key and $k_2$ as the authentication key and assume that they are

Browser $\mathcal{C}$ $\{LL_{\mathcal{C}} := (sk_{\mathcal{C}}, cert_{\mathcal{C}})\}$

Server $\mathcal{S}$ $\{LL_{\mathcal{S}} := (k_{\mathcal{KS}}, sk_{\mathcal{S}}, cert_{\mathcal{S}})\}$

$A' := r'_{\mathcal{C}} \in_r \{0,1\}^{p_5(\kappa)}$

$$\xrightarrow{\boxed{A'}}$$

$r_{\mathcal{S}} \in_r \{0,1\}^{p_5(\kappa)}$
$B' := r_{\mathcal{S}}|cert_{\mathcal{S}}$

$$\xleftarrow{\boxed{B'}}$$

$sid'_{\mathcal{C}} := r'_{\mathcal{C}}|r_{\mathcal{S}}$
$k'_p \in_r \{0,1\}^{p_3(\kappa)}$
$k'_m := \mathtt{PRF}_{k'_p}(l_1, sid'_{\mathcal{C}})$
$C' := \mathcal{E}_{pk_{\mathcal{S}}}(k'_p)|cert_{\mathcal{C}}$
$\sigma'_{\mathcal{C}} := Sig_{sk_{\mathcal{C}}}(\mathtt{Hash}(A'|B'|C'))$
$(k'_1|k'_2) := \mathtt{PRF}_{k'_m}(l_2, sid'_{\mathcal{C}})$
$h'_1 := \mathtt{Hash}(A'|B'|C'|\sigma'_{\mathcal{C}})$
$F'_{\mathcal{C}} := \mathtt{PRF}_{k'_m}(l_3, h'_1)$
$D' := Enc_{k'_1}(F'_{\mathcal{C}}|\mathtt{HMAC}_{k'_2}(F'_{\mathcal{C}}))$

$$\xrightarrow{\boxed{C'|\sigma'_{\mathcal{C}}|D'}}$$

$k'_p := \mathcal{D}_{sk_{\mathcal{S}}}(C')$
$k'_m := \mathtt{PRF}_{k'_p}(l_1, sid'_{\mathcal{C}})$
$(k'_1|k'_2) := \mathtt{PRF}_{k'_m}(l_2, sid'_{\mathcal{C}})$
$h'_1 := \mathtt{Hash}(A'|B'|C'|\sigma'_{\mathcal{C}})$
$(F'_{\mathcal{C}}|\eta'_{\mathcal{C}}) := Dec_{k'_1}(D)$
**if** $F'_{\mathcal{C}} \neq \mathtt{PRF}_{k'_m}(l_3, h_1)$
**or** $\eta'_{\mathcal{C}} \neq \mathtt{HMAC}_{k'_2}(F'_{\mathcal{C}})$
**or** NOT $Ver(cert_{\mathcal{C}}, A'|B'|C', \sigma'_{\mathcal{C}})$
**then** $\underline{\text{ABORT}}$ **else**
$h'_2 := \mathtt{Hash}(A'|B'|C'|\sigma'_{\mathcal{C}}|F'_{\mathcal{C}})$
$F'_{\mathcal{S}} := \mathtt{PRF}_{k'_m}(l_4, h'_2)$
$E' := Enc_{k'_1}(F'_{\mathcal{S}}|\mathtt{HMAC}_{k'_2}(F'_{\mathcal{S}}))$

$$\xleftarrow{\boxed{E'}}$$

$(F'_{\mathcal{S}}|\eta'_{\mathcal{S}}) := Dec_{k'_1}(E')$
$h'_2 := \mathtt{Hash}(A'|B'|C'|\sigma'_{\mathcal{C}}|F'_{\mathcal{C}})$
**if** $F'_{\mathcal{S}} \neq \mathtt{PRF}_{k'_m}(l_4, h'_2)$
**or** $\eta'_{\mathcal{S}} \neq \mathtt{HMAC}_{k'_2}(F'_{\mathcal{S}})$
**then** $\underline{\text{ABORT}}$
$m' := COOKIE(c)$
$F' := Enc_{k'_1}(m'|\mathtt{HMAC}_{k'_2}(m'))$

$$\xrightarrow{\text{F'}}$$

$m' = Dec_{k'_1}(F')$
$c' = VALUE(m')$
$(pk'_{\mathcal{C}}|ticket'|t') := DEC_{k_{\mathcal{KS}}}(c')$
**if not** $pk'_{\mathcal{C}} = pk_{\mathcal{C}}$
**and** $t' := HMAC_{k_{\mathcal{KS}}}(pk_{\mathcal{C}}|ticket)$
**then** $\underline{\text{ABORT}}$

**Fig. 2.** `BBKerberos` Protocol with TLS Client Authentication, Part 2. Boxed messages denote the standard TLS handshake.

the same for both directions.] The browser $\mathcal{C}$ also proves possession of the private key $sk_{\mathcal{C}}$ by signing the hash $h_{\sigma_{\mathcal{C}}}$ over all previously negotiated messages, i.e., signature $\sigma_{\mathcal{C}}$ (`ClientVerify`).

3. **Session Key Confirmation.** $\mathcal{C}$ 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 $\mathcal{C}$ (`Finished`). $\mathcal{S}$ verifies $\sigma_{\mathcal{C}}$, using the public key $pk_{\mathcal{C}}$. 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 $\mathcal{C}$ 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 $k_1$.

## 4.3 Security Analysis

In the following we analyze security of the `BBKerberos` protocol. We recall that the goal of the protocol is to provide secure authentication of $\mathcal{C}$ to $\mathcal{S}$, brokered by $\mathcal{K}$.

**Theorem 1.** *Let $\pi$ be a* `BBKerberos` *protocol as specified in Section 4. If* `PRF` *is pseudo random, $(Enc, Dec)$ are IND-CPA secure, $(\mathcal{E}, \mathcal{D})$ are IND-CCA2 secure,* `Hash` *is collision-resistant, $(Sig, Ver)$ is EUF-CMA secure, and* `HMAC` *is WUF-CMA secure, then $\pi$ provides authentication in the sense of Definition 2.*

*Proof (Sketch).* Due to space limitation, the full proof appears in the extended version of the paper. The main idea is to simulate an execution of the protocol based on the event $\mathsf{RevealDOM}(\mathcal{C})$ event. Then, the security can be reduced to the MAC-and-encrypt construction which conceals the authentication ticket and protects the ticket from forgeries, i.e. the adversary wins if it issues a valid ticket. However, if the $\mathsf{RevealDOM}(\mathcal{C})$ event does not occur, then the security can be reduced to the TLS handshake, which ensures that the owner of the ticket which is linked to some public key is in fact a legitimate owner by proving possession of the corresponding private key.

*Remark 3.* Although not stated in Theorem 1 explicitly, the security proof of `BBKerberos` based on the current TLS standard is valid in the Random Oracle Model (ROM) [5]. 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 [21] 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 `BBKerberos` from the strong assumptions of ROM. Similarly, the proof holds when the signature scheme is instantiated with RSA signature according to PKCS#1 (a.k.a. RSA-PSS) which in turn is known to provide EUF-CMA security in ROM (see [14] for the proof).

*Remark 4.* The HMAC construction used in the standard specification of the TLS protocol, formally, does not play any role for the security of the protocol. This is not surprisingly since every output of HMAC is encrypted using session key $k_1$ before being sent over the network. Since $k_1|k_2$ is treated as a single output of PRF the separation into $k_1$ and $k_2$ can be seen as redundant from the theoretical point of view. Note also that Krawczyk has proved the MAC-then-encrypt construction as secure in [18]. Though he mentions some problems in the general construction he shows that they do not apply to TLS.

## 5 Conclusion

We have introduced and analyzed a browser-based Kerberos protocol that made weak assumptions on the browser's security: The browser guarantees that private keys and sessions keys are confidential. However, our model allows the adversary to take control of the browser's DOM model, thus taking into account known browser attacks, such as XSS, Pharming. We did not consider malware attacks on the operating system the browser is running on. Since malware attacks may subvert the security of *any* cryptographic protocol (including classical Kerberos) without additional assumptions (e.g. the existence of a Trusted Platform Module), this exception seems justified.

We proved security in a game-based style by refining the model, proposed in [10], towards the consideration of DOM attacks. An interesting challenge for future work is to design Browser-based Kerberos which are provably secure under the stronger notion of Universal Composition[7,19]. Thus, the protocols could be composed with higher-layer protocols, but the analysis of the composition would be considerably simplified.

## References

1. Kerberos: The network authentication protocol, `http://web.mit.edu/Kerberos/`
2. Allen, C., Dierks, T.: The TLS protocol — version 1.1. Internet proposed standard RFC 4346 (2006)
3. Backes, M., Cervesato, I., Jaggard, A.D., Scedrov, A., Tsay, J.-K.: Cryptographically sound security proofs for basic and public-key kerberos (2006)
4. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
5. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Conference on Computer and Communications Security, pp. 62–73. ACM Press, New York (1993)
6. Boldyreva, A., Kumar, V.: Provable-security analysis of authenticated encryption in kerberos (2007)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145. IEEE Computer Society, Los Alamitos (2001)

8. Dhamija, R., Tygar, J.D., Hearst, M.A.: Why phishing works. In: CHI, pp. 581–590. ACM Press, New York (2006)
9. Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.-R., Schwenk, J.: Universally composable security analysis of tls—secure sessions with handshake and record layer protocols. Cryptology ePrint Archive, Report 2008/251 (2008)
10. Gajek, S., Manulis, M., Sadeghi, A.-R., Schwenk, J.: Provably secure browser-based user-aware mutual authentication over tls. In: ASIACCS, pp. 300–311. ACM Press, New York (2008)
11. Gajek, S., Schwenk, J., Xuan, C.: On the insecurity of microsoft's identity metasystem cardspace (HGI TR-2008-004) (2008)
12. Groß, T.: Security analysis of the SAML single sign-on browser/artifact profile. In: Annual Computer Security Applications Conference. IEEE Computer Society, Los Alamitos (2003)
13. Groß, T., Pfitzmann, B.: Saml artifact information flow revisited. Research Report RZ 3643 (99653), IBM Research (2006)
14. Jonsson, J.: Security proofs for the RSA-PSS signature scheme and its variants. Cryptology ePrint Archive, Report 2001/053 (2001)
15. Karlof, C., Shankar, U., Tygar, J.D., Wagner, D.: Dynamic pharming attacks and locked same-origin policies for web browsers. In: CCS 2007, pp. 58–71. ACM, New York (2007)
16. Kirda, E., Krügel, C., Vigna, G., Jovanovic, N.: Noxes: a client-side solution for mitigating cross-site scripting attacks, pp. 330–337 (2006)
17. Kormann, D., Rubin, A.: Risks of the Passport single sign-on protocol. Computer Networks 33(1–6), 51–58 (2000)
18. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001)
19. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: IEEE Symposium on Security and Privacy, pp. 184–200 (2001)
20. Pfitzmann, B., Waidner, M.: Analysis of liberty single-signon with enabled clients. IEEE Internet Computing 7(6), 38–44 (2003)
21. Shoup, V.: OAEP reconsidered. J. Cryptology 15(4), 223–249 (2002)
22. Stamm, S., Ramzan, Z., Jakobsson, M.: Drive-by pharming, pp. 495–506 (2007)
23. Stuart Schechter, A.O., Dhamija, R., Fischer, I.: The emperor's new security indicators. In: Symposium on Security and Privacy. IEEE Computer Society, Los Alamitos (2007)
24. W3C. Document object model (DOM) (2005), `http://www.w3.org/DOM`