Security Models for Protocols with Protocol-Unaware Principals

Thomas Groß*
IBM Zurich Research Lab
Rüschlikon, Switzerland

Klaus Kursawe†
Katholieke Universiteit Leuven
Leuven, Belgium

Ahmad-Reza Sadeghi‡
Ruhr-University Bochum
Bochum, Germany

Abstract

In common security protocols, principals are assumed to execute precisely the security protocol under consideration unless they are corrupted. Also the security models, analysis methods and proofs, proposed in the literature so far, rely on this approach. However, a protocol principal that has predefined feature set and that is unaware of the protocol it is involved in (protocol unawareness) has not been considered in the literature so far. An important and widespread class of protocols with this property consists of web-based applications that require to only rely on a standard web-browser due to cost-effectiveness and usability reasons. Hence, corresponding security services also rely solely on a browser for interaction with the user and establishing a security context. The browser has its own behavior since it reacts on a set of predefined messages, adds information to responses automatically, and stores histories in potentially adversarial environment (e.g., Internet kiosk at airport). Further, standard browsers simply do not execute most of the known security protocols for which security proofs exist. Hence, the security of the surrounding protocol can be crucially affected by the behavior of the browser.

In this paper we introduce the paradigm of protocol-unaware parties, and propose how to handle these parties for formal security proofs. In this context, we focus on the browser-based protocols where our primary goal is to discuss how to model these protocols, and how to approach them when proving them secure in such a model. We also suggest how to close the gap between model and reality by means of a wrapper. Our main goal is to present methods, principles, and a systematic approach to practitioners to model and analyze security protocols deployed in the practice, e.g., those based on browsers.

1 Introduction

In the last decade the need for distributed applications over open environments have received much attention by the research community and industry. These applications are becoming very sophisticated and confront us with challenging problems regarding the security and usability requirements: on the one hand these applications have to fulfill requirements on confidentiality, integrity and reliability. On the other hand they should be easy to use, not require too much knowledge from the user, and provide the desired services to the user transparently.

To analyze the security of the corresponding protocols, security and cryptographic research provides us with a variety of useful methods, security models and definitions. However, in security protocols, principals are assumed to execute precisely the security protocol under consideration, unless they are corrupted. This means that these principals are *protocol-aware*. Consequently the security models, analysis methods and proofs, proposed in the literature so far, rely on this approach.

*tgr@zurich.ibm.com
†klaus.kursawe@esat.kuleuven.ac.be
‡sadeghi@crypto.rub.de

Our new paradigm: However, these methods do not consider those parties that are *protocol-unaware*, i.e., have no information about the *surrounding* protocol where by surrounding we mean the protocol in which the principal is involved. More concretely, these principals have own behavior, i.e., they may react only to a set of predefined messages, may process and store security-critical information in potentially insecure places, and may need the assistance of other principals to access certain information (e.g., login information). We call this paradigm *protocol unawareness*. Hence, a protocol-unaware principal with own behavior can lead to major security problems for the surrounding environment and protocols. In Section 2 we will consider a concrete real world problem.

Concrete applications/protocols: Let us consider an important class of applications which is affected by this new paradigm: web-based applications. They have become very attractive to industry and into the focus of research, such that we use them as our running example in this paper. The motivation behind web-based services is to allow an easy deployment of applications at low cost and without specific user education: here users are provided with a browser that has some basic capabilities. Using the browser, users can request certain services and get back a view of the results. The desired services may be offered by a server or by computer systems of several affiliated enterprises while the required computation for the users' requests should remain transparent to users. The browsers in these applications do not require the installation of any special client software. This requirement is called *zero-footprint*. Also the corresponding security services must be zero-footprint, i.e., only a browser is used for user authentication, and, when needed, for retaining a secure channel with the user. Additional security relevant attributes about the users are requested from a third party and transmitted through a secure channel.

A prominent example of browser-based security protocols is *identity federation*, that aims at linking a user's (otherwise) distinct identities at several locations. The idea behind federated identity management systems is to allow individuals to use the same identification information to *sign on* to the networks of several enterprises in order to conduct transactions, e.g., to access certain services. The partners in such system depend on each other to authenticate their respective users and vouch for their access to services. The advantage is that companies can share applications without having to adopt the same technologies for different services, security and authentication, and reduce user management costs (such as the cost of password helpdesks and user registration and deletion.) Some concrete (and complex) browser-based security protocols are, e.g., Microsoft's Passport [8], the Security Assertion Markup Language (SAML)[1] standardized by OASIS [43], the Shibboleth project for university identity federation [7], the Liberty Alliance project [40], and WS-Federation [19, 20].

Security models and proofs: In most of these applications it is important that only registered and authorized users can securely and reliably access the desired services or resources. In security protocols the typical approach is to establish a secure channel. This is performed by a key exchange, based on local master keys, master keys shared with a third party, or public-key certificates, and to subsequently use the established key to secure the communication. For this a variety of protocols have been proposed in the literature starting with the seminal paper [31].[2] Moreover, many researchers have investigated the systematic security analysis of these protocols using various models and tools such as [28, 21, 27][3], [26, 29, 9], and [12, 32]. Further, cryptographic proofs of key-exchange and authentication protocols were initiated in [2]. Modeling secure channels by a comparison to ideal secure channels was introduced in [42, 36, 6]. Analyses

[1] It defines authentication and attribute tokens usable for identity federation, as well as basic profiles (protocols in typical security terms) for using these tokens.

[2] a vulnerability in one of the original protocols was later found in [25].

[3] based on abstractions of cryptographic primitives introduced in [11]

specifically for SSL and TLS were made in [44, 30, 33, 24]. These considerations are close to underlying mechanism used in browsers.

Browsers as protocol-unaware parties: However, standard browsers simply do not execute most of these protocols. The only exception is 2-party authentication through establishing SSL or TLS channels where the browser has a client-certificate on its own. However, this technique has several drawbacks. It is not considered truly zero-footprint, as users have to obtain a client-certificate for their browser. Also, users may not easily use different browsers at different times (e.g., kiosk scenario). And finally the client-authentication is bound to the browser itself (due to the browser's certificate) and not to the user who is actually browsing. Hence browser-based protocols are different from all protocols for which prior security proofs exist.

In case of Federated Identity Management systems, several papers have considered their vulnerabilities, in particular for Passport [23], the Liberty enabled-client protocol [39], and a SAML profile [16]. Others discussed privacy design principles and details [38, 34, 35]. Also the analysis of web services security protocols [15, 4] is related as they partially use the same security frameworks and introduce techniques for the analysis of real standards. Basic browser-based authentication without federated identity management is discussed in [14]. Apart from [17], which took a first step towards proofs of browser-based protocols by explicitly making top-down assumptions about the browser, the common approach is to identify vulnerabilities. Then, one tries to remove them and revise the protocol, as far as these security weaknesses could be disposed.

Now past experience in protocol design has shown that incorporating countermeasures against known attacks does not guarantee to recover all vulnerabilities. Hence it is desirable to devise security proofs. However, existing security proof techniques, both cryptographic techniques and formal-methods techniques, are not trivially applicable to protocols with protocol-unaware principals such as browser-based protocols. The new paradigm arises by the reason that a protocol-unaware principal, such as a browser, sticks to its own, predefined behavior which can have impact on the security of the surrounding protocol as pointed out before. In common security protocols, principals are assumed to execute precisely the security protocol under consideration unless they are corrupted. A browser, in contrast, reacts on a set of predefined messages, adds information to responses *automatically*, and stores certain information such as histories in places which cannot always be assumed to be secure, e.g., in an Internet kiosk at airport. We will consider a concrete real world attack scenario on SAML, found in [16], in Section 2.

Our approach and goals: Therefore, a detailed and rigorous browser model is a prerequisite for convincing security proofs of browser-based protocols, and no such model exists so far. For the resulting model, one has to assume that a real browser does *not* perform additional actions, because it seems that for most security protocols arbitrary additional actions could destroy the security. Hence, it is not enough to make a minimal model covering the few messages and parameters explicitly used by security protocols, but one has to get as close as possible to real browsers. Due to the limited capabilities of a browser, the user at the browser is an active participant and certain assumptions must be made about the user as well.[4] Hence, we also model the security-relevant browsing behavior of a user, i.e., a machine that implements the explicit constraints on a user that are needed for protocol proofs, but still allows arbitrary behavior apart from that.

Our main goal in this paper is to discuss the paradigm of security protocols with protocol-unaware principals, and to present methods, principles, and a systematic approach to practitioners to model and analyze security protocols with unawareness property in the practice. For this we focus on solutions for a class of protocols with widespread applications, namely, browser-based protocols with zero footprint. We demonstrate our approach to the new paradigm with browser-based protocols as running example in three

[4] e.g., that the user verifies that a secure channel to a trusted server is used before entering an important password.

steps. Firstly, we discuss models for browsers as protocol-unaware principal. Due to lack of space, however, we restrict ourself to an introduction without going into details. Of course, we have already started with fundamental theoretical basis for research in this area [18], and modeled the major building blocks for browser-based protocols. Our framework presents a rigorous and abstract model for a standard web browser as a principal for these protocols. We believe that we have captured the major explicit and implicit browser features that play a role in typical browser-based protocols. We call the corresponding features *main feature set*. Secondly, we explain different proof techniques employed by research for security proofs and their suitability for protocols with protocol-unaware parties. Thirdly, we discuss how to close the gap between formal models of protocol-unaware parties and their counterpart in the real world. Real web browsers exemplify this problem very well, as most real browsers have many more features than the modeled main feature set. Hence, security proofs in the model would not necessarily carry over to the real world. We discuss the approach to tame real browsers and introduce a special browser component or proxy, called *browser wrapper*. The idea is to have a user-friendly tool that reduces the feature set of the browser to one for which the surrounding security protocols are proven secure (main feature set). Hence, we can prevent that the protocol-unaware party browser breaks the protocol with yet another undocumented feature.

2 A Real World Scenario

The SAML Single Sign-on Browser/Artifact protocol [43] is one prominent example of a security protocol that is executed with a protocol unaware client. It is a three-party entity authentication protocol with a protocol-unaware standard web browser as principal.

We have three parties involved in the authentication protocol: an identity supplier S, a browser B, and identity consumer C. The task of S is to authenticate the user of browser B by some means and to generate a security token for higher-level authentication. C accepts such security tokens for user authentication.

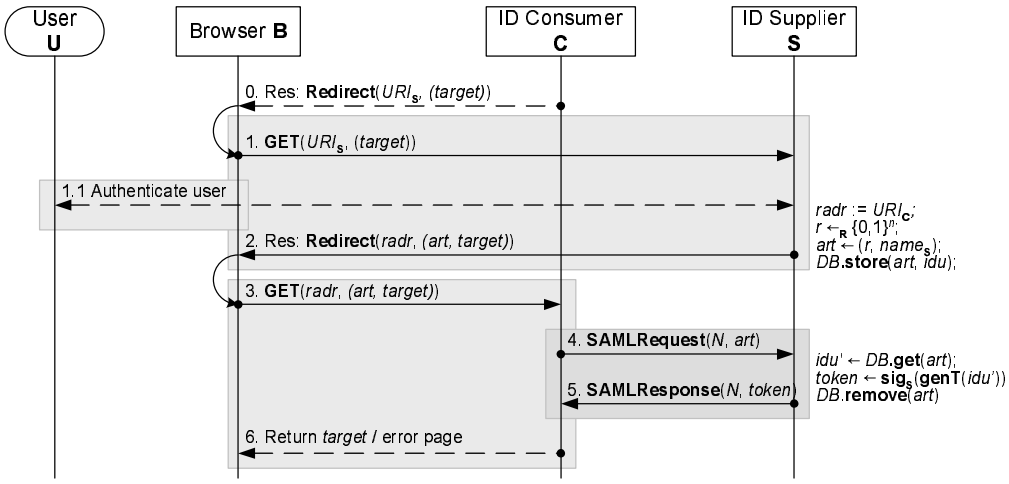

Figure 1: Protocol flow of the SAML Single Sign-on Browser/Artifact profile

The protocol itself is depicted in Figure 1 and works as follows: The protocol assumes that the user U has been once authenticated to S. U requests for a target page $target$ at C. In Step 1, it is then redirected[5] to URI_{S}[6] of identity supplier S requesting for a single sign-on page. The user of browser B performs a password-based user authentication in Step 1.1 authenticating for an identity idu. The identity supplier

[5] Redirect() and GET() are http protocol messages.

[6] Uniform Resource Identifier, an address for a resource available on the Internet.

generates a signed credential $token$ and responds with a redirect to C including a random number r into the redirect by means of the so called SAML artifact art. The browser B reacts to this redirect response by requesting the URL $radr$ specified by the identity supplier S. Hence the browser B establishes a secure channel to C and sends a GET request including the SAML artifact art. Upon such a request the consumer C establishes a secure and mutual authenticated channel with the identity supplier and transfers the artifact to S. Given such an artifact art, supplier S looks up the corresponding credential for idu and sends it to the identity consumer. After that the identity supplier deletes the artifact. This is important as this enforces the so called one-time property of the artifact. If S sees the artifact a second time, it will not recognize it anymore. With the artifact the identity consumer C identifies the user behind browser B. Consumer C sends the $target$ page accordingly or an error if something went wrong.

This protocol (i.e., SAML Single Sign-on Browser/Artifact) may be modeled in one of the known security models, with protocol machines, solely executing the protocol specified and analyze the protocol rigorously. However, we may not be able to find a vulnerability, and we may even be able to prove this protocol secure in the model with protocol machines. The main reasons for this conclusion are: (i) all parties involved are assumed honest, (ii) supplier S and consumer C have explicitly established a trust relationship in the setup and use mutual authentication, (iii) the insecure connections do neither carry confidential data nor has an adversary a benefit from manipulating their messages, (iv) all other data is transfered through secure channels, (v) the artifact is random (assumed to be hard to guess) and may only be used once, and finally (vi) the token is signed by S.

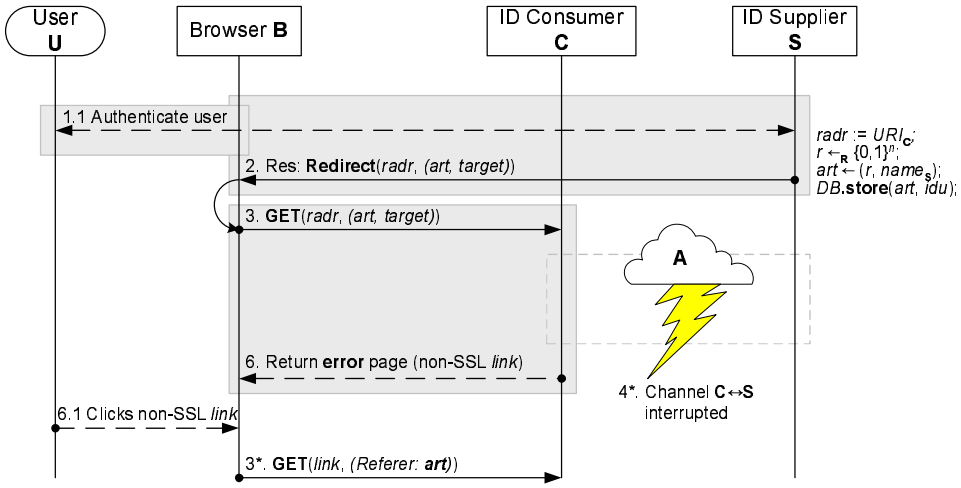

Figure 2: Referer Attack on the SAML Single Sign-on leveraging browser B being not protocol-aware

All this sounds reasonable. However, the security analysis [16] has introduced an attack based on special properties of B. We present this attack of [16] in Figure 2 starting with Step 1.1. Having a look at this attack with our paradigm in mind, it becomes clear that [16] used the protocol unawareness of B to break the SAML protocol. Loosely speaking, the adversary provokes an information flow of the SAML artifact art by interrupting the secure channel between machines C and S. Let us start with a Step 3 redirect in which the browser B arrives at C and hands over the artifact art at URI_{C}. Observing such request, the adversary interrupts the secure channel between S and C. Note that the artifact art then cannot be transfered to identity supplier S and therefore will not be invalidated. As a first result, we see that there exists a valid SAML artifact. Not being able to establish a channel to S, consumer C will respond to the browser with an error message. We assume that this error message contains a link to an insecure HTTP URL and that the

user clicks this link.[7] Now, the protocol unawareness of the browser comes into play. If a browser comes from a source which has its own URI, a browser may include the preceding URL into the so called Referer tag of the new request. In our concrete example, a browser would include the full URI_{C} into the Referer tag which includes the SAML artifact art. Therefore, the browser exposes a still valid artifact to an insecure channel and therefore breaks the protocol, since the adversary may eavesdrop the artifact and use it in a Step 3 message.

The protocol scenario discussed above demonstrates that a protocol may be perfectly secure if its modeled as protocol machines solely executing the protocol specified. Further, it demonstrates that the protocol may be completely broken nevertheless because of the protocol unawareness of a participating party. It shows that neglecting the behavior of a protocol unaware principal will put the protocol security at stake.

3 Security Models

Meanwhile there exists a large body of literature on security proofs in cryptography and security where we find a variety of security models and proof techniques. A typical technique in classical cryptography is to prove security properties of protocols by reducing their security to an underlying cryptographic assumption or primitive (e.g., if an adversary can break the protocol under consideration, then it will break the underlying signature scheme assumed to be secure.) Also, other types of proofs may be used such as static information flow analysis or backward chaining of a desired security property to underlying assumptions or security properties of primitives.

Another method for protocol proofs was introduced by the *ideal-world/real-world paradigm*, in which one compares an ideal system with a real system and proves that the real world is "as secure as" the ideal world. Canetti [5] and Pfitzmann/Waidner [37] both developed the ideal-world/real-world paradigm for reactive systems and formalized a generic notion of *simulatability* and *composability*. The advantage of this method is: once one has proven a protocol or primitive "as secure as" a corresponding ideal system one can exchange real protocol instantiations with the corresponding ideal functionality and analyze the security with the ideal component.

We concentrate on the Pfitzmann/Waidner model and use reactive interacting I/O automata connected by *ports* as basis. By means of such automata, one models protocol machines in the real world that jointly execute the protocol. One complements the machines with an honest user H, which models a higher protocol layer (e.g., an e-commerce application) or ultimately the user of the protocol. Also, one complements the protocol machines with an adversary A that usually schedules all network traffic and may eavesdrop as well as manipulate messages sent over insecure channels. Protocol machines may be corrupted by the adversary and then become part of the adversary machine A. Machines H and A form environments in which the protocol specified may run.

To prove the security of a protocol one designs an ideal world where everything that might possibly happen is good by design. In this ideal world, one designs a *trusted host* TH that executes the designed protocol always according to the specification. The ideal TH replaces the protocol machines that we have in the real world and, therefore, TH controls the same ports the protocol machines would in the real world. This ideal world normally does not bother for cryptography but achieves the security goals in a bullet proof and easy to understand way. Very loosely speaking, one then proves that everything that may happen in the real world can also happen in the ideal world. A bit more formally, we describe "as secure as" for a system with real machines $\hat{M}_1$, a protocol interface S and ideal machines $\hat{M}_2$, in our case consisting of a trusted host TH corresponding to $\hat{M}_1$: $\hat{M}_2 = \{\mathsf{TH}\}$ (also depicted in Figure 3 with simplified machines). $(\hat{M}_1, S)$ "as secure as" $(\hat{M}_2, S)$ means: for every configuration $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1)$, there exists a

[7] As all common e-commerce sites contain a non-HTTPS link to their main page, we can safely assume this.

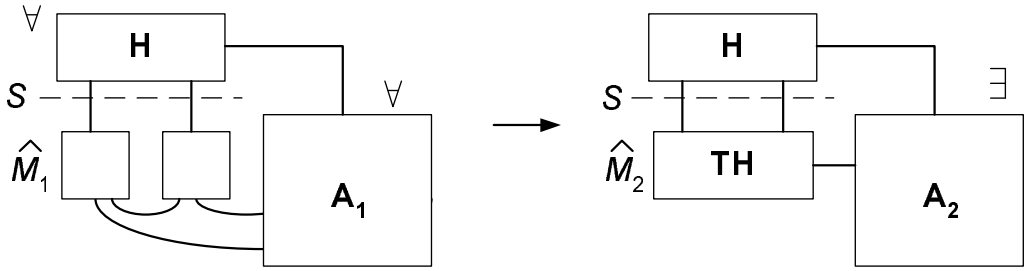

Figure 3: Reactive simulatability ("as secure as") of real machines $\hat{M}_1$ and ideal machines $\hat{M}_2$ (only consisting of trusted host TH)

configuration $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2)$ (with the same H) such that the view of H in both configurations is indistinguishable, i.e., $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$. Pfitzmann/Waidner name this property *reactive simulatability*. In the case of blackbox simulatability A_2 consists of a fixed submachine, a *simulator*, using the real world adversary A_1 as black box.

How does those approaches work when a protocol-unaware party is involved in a protocol run? One might be tempted to abstract from the protocol-unaware party and model it as a protocol machine as usual. But then this protocol machine may not reflect the behavior that the protocol-unaware party has. Thus, protocol proofs do not carry over into reality. A more promising approach is to model the protocol-unaware party with all its properties independently from a concrete protocol. Given such a generic formal model, one realizes the surrounding protocol including this party and tries to prove the security properties of the protocol.

One may still approach proofs with the ideal-world/real-world paradigm, yet for protocol-unaware parties with large state machines and much other behavior, this may be tedious as ideal systems need to reflect all imperfections of the protocol-unaware party involved. Also, proving indistinguishability between probability distributions of the views of ideal and real world may not capture the intuition of practitioners designing those protocols. More handy as a first step into the area of protocol-unaware parties comes the classical proof of specific security properties such as authenticity, confidentiality or integrity. One backs up the proof with invariants about the behavior of the protocol-unaware parties, protocol parties and interactions between them. A prominent example of such invariants are static information flow analysis that state to which variables and ports certain information may flow. Building upon the invariants one proves a security property of the protocol at a time. We sketch an example for such a proof for a single sign-on protocol involving a protocol-unaware browser in Section 4.4.

4 Browser Model

In this section we focus on the browser model and explain informally its main components, their roles, and their relations.

4.1 Overview of Main Components

We begin with the general structure of browser-based protocols, and proceed to an overview of our model for security analysis and proof of such protocols.

Browser-based protocols generally involve a browser and a number of servers. The protocols are client-server based with a browser as client.[8] Therefore, the initial authentication of the user behind a browser is an

[8]In general, browsers have the capability to engage in client-authentication, however the users usually do not install the certifi-

integral part of all browser-based protocols. In the standard case, this initial user authentication is performed by means of the user's knowledge of a username-password pair for another entity, such as a server trusted by the user or a local wallet. Browser-based protocols bootstrap the initial user authentication for various applications, such as stronger proofs of authentication to other parties, authorization, attribute or credential exchange, pseudonym establishment etc. Because of the limited cryptographic capabilities of a standard web-browser and the anonymity of the browser itself, browser-based protocols are very vulnerable to man-in-the-middle and replay attacks. Therefore, beginning with the initial user authentication, browser-based protocols must use an unbroken chain of secure channels to the final protocol application, e.g., accessing the desired resources or services. Browser-based protocols may transfer data during such protocols in general by means of two main messages: they may transfer data as part of the body of HTML *forms* and use HTTP POST requests to send them to servers.[9]

Browsers may also transfer data as part of the URL in HTTP GET requests. However, the amount of data transferable by this means is very limited. Further, the data from the URL flows into browsing history and into future requests (e.g., through *referer tags*). Most browser-based protocols only transfer a random number, called *artifact*, as reference to a piece of payload and transfer the payload itself over a secure back-channel in which the browser is not involved anymore (e.g., in so-called browser/artifact profiles).

Our goal is to model and analyze the whole class of browser-based protocols with all its variants. For this, we propose a framework based on the reactive machine model of Pfitzmann and Waidner [37] as briefly reviewed in Section 3. Figure 4 shows an overview of the model. The framework contains three generic automata that are indispensable when proving the security of browser-based protocols rigorously. The first machine B models the generic browser machine. This machine involves all the capabilities of web browsers used in browser-based security protocols. The machine models the characteristic behavior of browsers with their typical messages and their *imperfections* such as information flow to other parties. Especially the precise model of the browser's information flow is of vital importance for security proofs of browser-based protocols. Note that we only model the subset of messages and parameter types that is actively used in browser-based protocols or may have impact on their security. This part is the subject of Section 4.2.

The second machine U implements the *minimum assumptions on secure user behavior* (the knowledge of a user and his or her capabilities to handle initial user authentications.) It also enforces constraints about a user without which every browser-based protocol must fail. Those constraints are, for instance, that the user only accepts certificates compliant with its trust relationships, only sends its authentication data to the corresponding servers, and this only over a secure channel. Also, the user observes the channel status, especially step-downs from secure to insecure channels. This part is the subject of Section 4.3.

The third machine $\mathsf{secchan}$ models the channel types used in browser-based protocols and the dynamic establishment and closing of channels. It implements the behavior of secure channels as implemented within HTTPS (see [41]).

We also define the communication between the various machines by means of ports, e.g., $\mathsf{gui}_{\mathsf{U,B}}?$ and $\mathsf{gui}_{\mathsf{B,U}}!$ between the user and browser machines and by $\mathsf{channel_in}?$ and $\mathsf{channel_out}!$ between the channel machine and other machines (see Figure 4)

To analyze and prove browser-based security protocols one complements these general-purpose machines with one or more server machines, here denoted by S_i, that jointly execute the browser-based protocol. Furthermore, one configures the user machine U with a suitable initial trust relationship and knowledge about trusted parties.

---

cates required.

[9] Usually, this POST is issued by active scripts (e.g. JavaScript), which is unfavorable in certain scenarios due to security reasons (e.g., Trojan horses). Moreover, the payload transferred gets known to the user and the operating system of the user's platform.

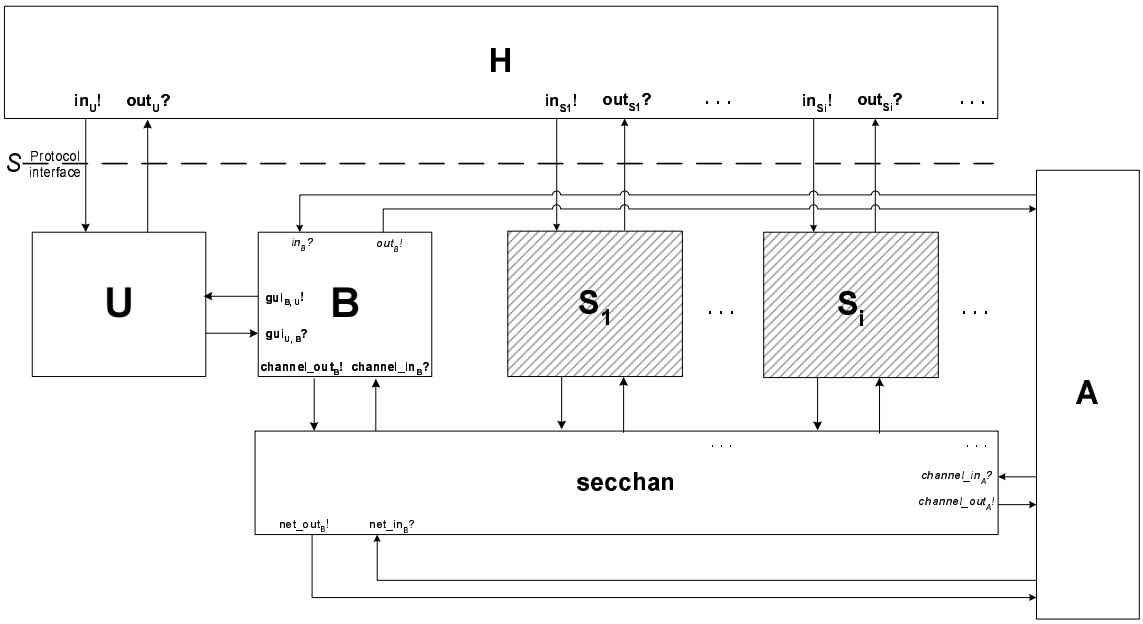

Figure 4: Overview over the system architecture for browser-based protocols with a browser B, a corresponding user U, multiple servers S_i, secchan and the corresponding interfaces and channels.

4.2 Modeling a Browser

In this section, we introduce the functionality of real web browsers as a prominent example of a protocol-unaware party. We describe the main feature set that we model and its benefit for the rigorous security analysis of browser-based security protocols.

A web browser acts as the client in transactions of the Hypertext Transfer Protocol (HTTP) [13] and renders protocol state and payloads to its user. A browser acts on behalf of one single user in a browsing session. A real browser accepts inputs from the user specifying addresses to retrieve and render the content associated with such addresses, as well as the status of the channel to the server and the identity of the server. As a browser may display a wide range of content retrieving arbitrary addresses, it fulfills two basic prerequisites for being used in surrounding protocols. Also, the HTTP protocol executed by the browser has versatile applications providing surrounding protocols with various options.

HTTP is a client-server protocol positioned in the application layer of the TCP/IP protocol stack with variable underlying transport protocols. In order to initiate an HTTP transaction, the browser establishes a connection to a server specified by the address to access and may leverage multiple types of transport protocols underlying the HTTP transaction, e.g., TCP/IP, SSL 3.0 or TLS1.0 [10]. Having established a channel, the browser issues an HTTP request to the server. Such a request specifies the resource that the browser intends to retrieve, but may also contain additional data and parameters. The server evaluates the request and issues a response using the same channel. We call such an interaction an HTTP transaction. In principle, browsers do not need to hold state beyond such a single transaction, however, a real web browser builds, e.g., local cache and browsing history and lets a transaction influence the subsequent one. HTTP transactions may implement various functions. Clients may use a transaction to retrieve data by a GET request, and send data by a POST request. Servers may not only deliver content but also direct the browser to a behavior change by issuing executable scripts and error messages. Most prominent examples are HTTP responses with scripted form POST and redirect messages, which direct the browser to another address of the server's choice. The server may also issue an HTTP response that requests a user's authentication by means

of a username-password pair. The versatility of HTTP renders a browser a relatively complex protocol-unaware party. As one would expect, a browser can send as well as retrieve data, where the accessed URL as well as the transmitted payload may contain data of surrounding protocols. Also the browser may be directed by server commands to access certain URLs. This property clearly drives surrounding protocols, as those protocols have the browser communicate with other parties. However, it also makes the browsers—being unaware of a surrounding protocol—a security risk. The browser may become cue ball of other protocol participants arbitrarily redirecting the browser around.

Supported by a rudimentary trust model, a web browser can establish secure channels to servers by leveraging the SSL3.0 or TLS1.0 protocol. This property is crucial for surrounding protocols, however, browsers normally only use server-authenticated secure channels and do not provide mutual authentication.[10] surrounding protocols using the protocol-unaware browser do not engage in a key-establishment protocol on their protocol layer using the key for establishing a secure channel (the usual technique in entity authentication), but have the browser establish such an semi-anonymous secure channel and gain mutual authentication by security tokens or password-based user authentication. We need to model this special behavior of a protocol-unaware browser, as usual models of mutual secure channels do not capture this functionality.

An important aspect of real web browsers is that they do more than surrounding protocols intend. This aspect clearly demonstrates the protocol unawareness of a browser, which we also used as an example in the SAML attack in Section 2. Most prominent is the problem of information flow. On the one hand, we have information flows through the HTTP requests to the server, e.g., by means of the HTTP referer tag. This means that in each message to another party a browser may leak information not intended by a surrounding protocol to the communication partner. We already used the information flow of preceding URLs in Section 2. Moreover, data about the user may flow to other parties putting privacy properties of surrounding protocols at stake. Also, features such as history, cache or password storage have data flow to the underlying operating system.

Browsers normally may execute active scripts, such as JavaScripts and Java Applets which, loosely speaking, renders them Turing-complete. In this paper, however, we focus on pure browser-based protocols that do not use generic active content, such as JavaScript or Java Applets, since most of these protocols fall into this class. The only exception we take into account is the *scripted form posts*.[11] Browsers may also store small pieces of information provided by a server on their machine, and therefore generate a persistent state. We do not consider cookies as many browser-based protocols do not use them directly.

4.3 Modeling User's Browsing Behavior

Using the browser as protocol-unaware in a surrounding protocol, the user has an important role. As we have discussed, the browser is a state-less device that only provides a rudimentary trust management. However, in higher protocols one needs to store information beyond a single transaction and have a stronger trust model. Also, the user controls most of the browser's behavior and has the final say about the browser's actions. Therefore, without a user fulfilling certain tasks and properties all browser-based protocols must fail. Thus, we consider a user as active protocol participant and model it by an, in general, transparent machine. This user machine, however, enforces the basic requirements for browser-based protocols. Thus, in principle, the user machine is also protocol-unaware. It is a generic machine for all kinds of browser-based protocols.

However, the user stores data of the protocols it is involved in. Such data may be addresses of trusted

[10] Usually a standard browser does not hold a private key for authentication and needs to authenticate its user by other means.

[11] Browser-based protocols use HTTP forms to transfer data from one server to another by a method called scripted form POST. The issuer of the data includes the data in hidden form fields of its HTTP response and has the browser submit the form by means of a JavaScript. Upon such a submit command the browser issues a HTTP POST containing the data to the receiving server.

servers or identity information. The user knows its trust relationship to other parties in the browser-based protocols. The machine U stores this data in its state. This data is protocol-specific, therefore we have a party with generic behavior, yet knowledge stemming from surrounding protocols. As the web browser is not aware of any surrounding protocol, the user acts as a supervisor of the protocol flow the browser is involved in. It checks certificates, observes the status of secure channels and logs off from the browser in error cases. Also, the user engages in the user authentication with a server and performs the crucial verification of the server's identity. The machine acts autonomically upon browser dialogs concerning these tasks.

4.4 How to Prove the Security of a Browser-based Protocol

In this section, we sketch how to approach protocol proofs with a protocol-unaware party with the browser model. From the proof techniques we discussed in Section 3, we choose the classical proof of security properties, as it is better suited for understanding the intuition of the proof. We focus on the authenticity property of a browser single sign-on protocol, which is one of the most commonly used browser-based identity federation protocols. We present a simplified version of the theorem. It means, loosely speaking, that if the identity consumer C receives a security token with user identity idu through a secure channel, then the user at the other end of the channel is indeed the user U corresponding to idu.

Theorem 4.1 (Authenticity of Browser Single Sign-on) *Let a correct user machine* U *with correct browser* B *be given. Let an identity consumer* C *and its identity supplier* S *be honest. Then the following holds: If the identity consumer* C *obtains an output* $(\mathsf{accepted}, cid_{bc}, idu)$ *from the single sign-on protocol, then the secure channel with channel identifier* cid_{bc} [12] *is a channel with the only user* U *that has the identity* idu *at* C*'s identity supplier* S. □

To prove such statements we approach as follows: we first show that C accepts a certain security token only if (i) all security tests prescribed in the protocol description were successful and (ii) the signature of the token could be verified as an identity supplier of C. Then we prove, using the security of the signature scheme against adaptive chosen-message attacks, that C's identity supplier must have signed the token at some time in the past. The absence of cross-protocol key usage[13]restricts this to a protocol-run of the single sign-on protocol. Let us name the corresponding identity supplier S. We prove under which circumstances S may have signed the token and to which party S may have sent it. We show that the token generation is the direct consequence of a successful user authentication over a secure channel with some browser and a user with identity idu.[14]

Until now, we have only considered *protocol aware* parties S and C. However, now we need to prove to which parties the token or the information about the token may flow. Then, the *protocol-unaware* web browser and the user are involved in the communication and need to be taken into account. Here we use the browser model, which we informally introduced in Section 4.2. It does two kinds of proofs: (i) correctness, i.e., to which parties the browser may have send the token actively, and (ii) information flow, i.e., to which variables (and consequently to which other parties) information flow about the token and other parts of the browser's state takes place. Usually, one proves that as invariants over the browser's information flow, i.e., always the same variables are the destination of the information flow. For the user machine one handles the proof similarly.

Finally, we prove that if honest party C accepted the token then it was indeed issued by S in the same protocol-run and was handed over by the browser B to the correct C. We leverage correct addressing and

[12]We use strings as identifiers for the channels a party maintains.

[13]i.e., the signing key is only used for this protocol

[14]The security and information flow of the initial user authentication are proven separately.

channel establishment as well as the correspondence of audience specification and POST address specified.

5 Taming Your Wild Browser

One of the major practical problems with our approach is that modern browsers are complex systems, and they may perform additional functionality that is not allowed in the model. JavaScript, for example, may allow to completely reconfigure the browser. Thus, we need some way to transform a real world browser into a secure browser as defined in our model.

Thus, we not only need to adapt our model to reflect the properties of the browser, but also the browser to stick to the restrictions required by the model to prove the security of browser based protocols. Such a browser may limit the amount of information it gives to the server – e.g., by not including a referer tag, and ignore critical code such as JavaScript (note that in the implementation, the browser should maintain a set of allowed actions rather than a set of forbidden ones).

This transformation must be temporary — one cannot expect a normal user to permanently disable features such as flash and JavaScript, as they are vital for numerous websites. Also, we must keep in mind that the guarantees we can give in practice are limited; if the browser is permanently modified either in its binary form or by a rouge plugin, it may not be possible anymore to give any security guarantees.

There are two ways to adapt the browser to the needs of a model that allows provably secure protocol executions: either by changing the browser itself, or by wrapping it in a security envelope that filters all critical code.

5.1 Changing the Browser

The first possible approach is to modify the browser itself to allow for a secure mode. This modification can be done relatively easily, as most modern browsers can be extended by means of browser plugins. An example for a plugin similar to the proposed one is the adblock extension, which filters certain advertisement from the webpages the user visits. This approach has the advantage that it is easy to operate for the end-user, and there are few limits in the way the browser plugin can adapt the browser to the needs of our model.

However, there are some practical issues. First of all, extending the browser itself requires different implementations of the software for different browsers, and potentially for different versions as well. This increases the management costs of such an approach, and bears the risk that less popular browsers are not supported. Also, the ease of expanding the browser with the new functionality may well work against our purpose. In most cases, the plugins themselves are written in a high-level script language such as JavaScript, and there may be numerous conflicting plugins adding functionality to the same browser. Suppose, for example, a plugin that adds debugging information to the HTTP header. The functionality of this plugin is directly contradicting our security plugin, and it is not clear which plugin can act last to cancel the other one.

5.2 Using a Proxy Wrapper

As an alternative to modifying the browser, it is possible to implement an html-wrapper which filters all content before it is delivered to the browser. This can be done by implementing a web proxy. Instead of connecting directly to the Internet, the browser connects to a proxy, which filters critical content and only allows tags that are allowed according the underlying security policy. There are several examples on implementing local proxies, such as the Java Anonymity Proxy for untraceable web browsing [22] and the PCA Prototype for secure browser authentication [1].

This approach solves all problems of a plugin based solution. The proxy can be executed independent of the browser used, and largely independent of the underlying operating system. Additionally, the security functionality is in an isolated, independent program that is not influenced by the browsers complexity.

However, isolating the program that adapts the functionality to our model from the main browser bears other problems. If a protocol uses a secure connection (i.e. an SSL connection via a https) — and we can expect the protocols we consider to do so — the communication between the browser and the server is encrypted, and the proxy has no means of accessing the content. However, the proxy must analyze and manipulate the communication, rather than just tunneling the SSL connection (which is, for example, done in JAP). The other problem is that a proxy can only access the traffic between the server and the browser, but has no access to the communication between the browser and the host operating system. If a browser violates the model by other means, for example, by writing critical information on disk and thus making it available to other processes, the proxy wrapper may never even notice. While it is possible to tweak the filesystem in a way that prevents the browser from doing so, this would destroy much of the elegance a wrapper-based protocol can offer.

5.3 Our Proposal

While both solutions presented above have successfully been deployed in related settings, and both could be used to achieve most of our goals, neither solution is really satisfying. The browser extension delivers performance and all functionality required, but requires browser and version specific implementations; furthermore, as it makes the browser more complex rather than less, there may be new security issues. The proxy-based solution is browser independent and does not need to increase complexity, but cannot access all critical information flow paths and does have problems with SSL encrypted communication.

In both cases, it is not impossible to resolve the problems. Browser extensions mostly result in more implementation effort and an increased danger of security critical bugs in the implementation; most problems with a proxy based solution eventually come down to a user interface issue.

Even though a browser extension based solution will faster result in working code, we prefer to base an actual solution on the proxy model. It still may be helpful to use browser extensions to assist the proxy; such a plugin could give the proxy some privileged display, and help cleaning up data traces stored by the browser. However, for interoperability and security reasons, we do not want to base any security on the plugins, but rather use them to improve user convenience.

To overcome the above mentioned problems, some of the browser functionality has to be moved to the proxy – essentially, the proxy has to carry some of the protocol logic, using the browser as a rendering engine:

- The proxy has some privileged display to provide status information, such as a secure connection, to the user. The most elegant approach would be a browser plugin that gives the proxy access to the browser status bar. As this may not be an option for all browsers, JavaScript could be used as an alternative (as the proxy filters all JavaScript when in secure mode, the problem of impersonation could be solved). Finally, the proxy could revert to displaying a status header using standard html frames.

- Any cookies, passwords and authentication information needed for the protocol are stored by the proxy and never forwarded to the browser. If information has to be stored on the platform, it is stored in encrypted form and only applicable to a user with the correct authorisation. In this functionality, our proxy works similar to the CookieCooker [3].

- SSL connections to the remote host are built by the proxy and the server; the browser only opens a secure connection to the proxy itself. Certificate validation and indication of a secure connection

should still be indicated in the browser, even though the browser is not involved in the actual secure communication with the server.

- While it is possible to withhold most critical information — all authentication information, or even the content of hyperlinks — from the browser, the proxy may not be able to hold back all critical information and prevent the browser from storing it on disk, e.g., in history or cache files. This has to be resolved by user interaction (some browsers, such as firefox, make it quite easy to delete all collected information), or again by a simple plugin.

For our purposes, it does not make a significant difference if the proxy is implemented locally or remotely; thus, such a proxy is applicable both for home use and in a kiosk system, where a user has no authority to perform major changes on the browser configuration. In any case, however, we must assume that the underlying operating system and the browser have not been maliciously modified — while we can prevent a remote attacker from using advanced browser features against the user, protecting the user from a malicious browser is not the scope of this work.

6 Conclusion

We introduced a new paradigm that we call protocol unawareness: in security/cryptographic protocols the involved principals are assumed to precisely behave according to the protocol specifications, unless they are corrupted. This holds also for all the security models, analysis methods and proofs, proposed in the literature so far. The existing literature, however, does not cover those security protocols where one or more principals may have no information about the protocol they are involved in. Hence these parties have their own behavior which may be a high security risk for the surrounding protocol. We then focused on a widespread class of applications, namely web-based applications, that is covered by this new paradigm. These applications require to use standard browsers which have own behavior and are unaware of the surrounding protocol. We discussed that the aspect of protocol unawareness of browsers leads to crucial security problems in the corresponding protocols and showed that the real life attack scenario that have been found on certain single sign-on protocols was indeed due to the browser's protocol unawareness.

We focused on solutions for browser-based protocols, and presented and discussed methods, principles, and a systematic approach for practitioners to model and analyze security protocols (here browser-based) with unawareness property in the practice.

Acknowledgements

We thank Birgit Pfitzmann for the valuable discussions on the modelling a web browser as protocol unaware party. Also, we thank Michael Waidner for the initial discussions on wrapping a real browser. This paper was written as cooperation of members of the European Network of Excellence in Cryptology (ECRYPT) within the Information Societies Technology (IST) Programme of the European Commission.

References

[1] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the Web. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, August 2002. `http://www.ece.cmu.edu/~lbauer/papers/webauth-sec02.pdf`.

[2] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.

[3] Oliver Berthold. Cookiecooker, 2002. http://www.cookiecooker.de/.

[4] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 198–209. ACM Press, 2004.

[5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Report 2000/067, Cryptology ePrint Archive, October 2001. Extended Abstract appeared in proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001.

[6] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels (extended abstract). In *Advances in Cryptology: EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002. Extended version in IACR Cryptology ePrint Archive 2002/059, http://eprint.iacr.org/.

[7] Scott Cantor and Marlena Erdos. Shibboleth-architecture draft v05, May 2002. http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v0%5.pdf.

[8] Microsoft Corporation. .NET Passport documentation, in particular Technical Overview, and SDK 2.1 Documentation (started 1999), September 2001.

[9] Zhe Dang and Richard Kemmerer. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. http://dimacs.rutgers.edu/Workshops/Security/.

[10] Tim Dierks and Christopher Allen. RFC 2246: The TLS protocol, January 1999. Status: Standards Track.

[11] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[12] B. Dutertre and S. Schneider. Using a PVS embedding of CSP to verify authentication protocols. In *Proc. International Conference on Theorem Proving in Higher Order Logics (TPHOL)*, volume 1275 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1997.

[13] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. RFC 2616: Hypertext transfer protocol – HTTP/1.1, June 1999. Status: Standards Track.

[14] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001. USENIX. An extended version is available as MIT-LCS-TR-818.

[15] Andrew D. Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *Proc. 2002 ACM Workshop on XML Security*, pages 18–29, Fairfax VA, USA, November 2002.

[16] Thomas Groß. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Proc. 19th Annual Computer Security Applications Conference*. IEEE, December 2003.

[17] Thomas Groß and Birgit Pfitzmann. Proving a WS-Federation Passive Requestor profile. In *2004 ACM Workshop on Secure Web Services (SWS)*, Washington, DC, USA, October 2004. ACM Press.

[18] Thomas Groß, Birgit Pfitzmann, and Ahmad-Reza Sadehgi. Browser model for security analysis of browser-based protocols. Research Report RZ3600, IBM Research, April 2005. http://www.research.ibm.com/resources/paper_search.shtml.

[19] Chris Kaler and Anthony Nadalin (ed.). Web Services Federation Language (WS-Federation), Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, http://www-106.ibm.com/developerworks/webservices/library/ws-fed/.

[20] Chris Kaler and Anthony Nadalin (ed.). WS-Federation: Passive Requestor Profile, Version 1.0, July 2003. BEA and IBM and Microsoft and RSA Security and VeriSign, http://www-106.ibm.com/developerworks/library/ws-fedpass/.

[21] Richard A. Kemmerer. Using formal verification techniques to analyze encryption protocols. In *Proc. 1987 IEEE Symp. on Security and Privacy*, pages 134–138, Oakland, California, April 1987. IEEE.

[22] Stefan Köpsell. Java anonymity proxy, 2005. `http://anon.inf.tu-dresden.de/index_en.html`.

[23] David P. Kormann and Aviel D. Rubin. Risks of the Passport single signon protocol. *Computer Networks*, 33(1–6):51–58, June 2000.

[24] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: how secure is SSL?). In *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

[25] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–135, 1995.

[26] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[27] Catherine Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[28] Jonathan K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.

[29] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proc. 18th IEEE Symposium on Security & Privacy*, pages 141–151, 1997.

[30] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0 and related protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. `http://dimacs.rutgers.edu/Workshops/Security/`.

[31] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[32] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[33] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.

[34] Birgit Pfitzmann. Privacy in enterprise identity federation - policies for Liberty single signon. In *3rd International Workshop on Privacy Enhancing Technologies (PET 2003)*, volume 2760 of *Lecture Notes in Computer Science*, pages 189–204, Berlin, March 2003. Springer-Verlag, Berlin Germany.

[35] Birgit Pfitzmann. Privacy in enterprise identity federation - policies for Liberty 2 single signon. *Elsevier Information Security Technical Report (ISTR)*, 9(1):45–58, 2004. `http://www.sciencedirect.com/science/journal/13634127`.

[36] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.

[37] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 184–200, Oakland, CA, May 2001. IEEE Computer Society Press.

[38] Birgit Pfitzmann and Michael Waidner. Privacy in browser-based attribute exchange. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 52–62, Washington, USA, November 2002.

[39] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty single-signon with enabled clients. *IEEE Internet Computing*, 7(6):38–44, 2003.

[40] Liberty Alliance Project. Liberty Phase 2 final specifications, November 2003. http://www.projectliberty.org/.

[41] E. Rescorla. Internet RFC 2818: HTTP over TLS, May 2000.

[42] Victor Shoup. On formal models for secure key exchange. Research Report RZ 3120 (#93166), IBM Research, April 1999. Version 4, November 1999, available from http://www.shoup.net/papers/.

[43] OASIS Standard. Security assertion markup language (SAML), November 2002.

[44] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.