Analysis of Liberty Single-Sign-on with Enabled Clients

Channel-based enabled-client protocols, such as the Liberty-enabled client and proxy profile, offer Web single-sign-on service; however, several security concerns remain.

Birgit Pfitzmann and Michael Waidner
IBM Zurich Research Lab

Many users simplify Web sign-on by choosing the same username and password for all services. This approach presents several problems, however. Each service, as well as any successful attacker against one service, can impersonate the user to all the other services. In addition, services that want to verify that the user owns a certain preexisting name (first and last name, email address, or credit card number, for example) must still verify the name during registration. Finally, using the same information for all services makes regular password changes, which are a good security practice, awkward.

Web single-sign-on protocols — such as Microsoft Passport (http://msdn.microsoft.com), Oasis's Security Assertion Markup Language (SAML),[1,2] and the Internet2 project Shibboleth[3] — aim to solve these problems by letting individuals log in to many Internet services while authenticating only once, or at least always in the same way. Enterprises hope that single-sign-on protocols will significantly decrease customer-care costs due to forgotten passwords and increase e-commerce transactions by enhancing the user experience. Commercial interest centers on distributed enterprises and on small federations of enterprises with existing business relationships, such as supply chains.

In July 2002, the Liberty Alliance published a six-part specification that extends SAML message formats, classifies authentication classes, and provides four message-exchange protocols, starting with an architecture overview.[4] In this article, we concentrate on the fourth protocol, the Liberty-enabled client and proxy (LECP) profile.[5] The LECP protocol assumes a special protocol-aware client (the enabled client), whereas the first three Liberty protocols — like Passport, SAML, and Shibboleth — assume an

 Published by the IEEE Computer Society 1089-7801/03/$17.00©2003 IEEE

unmodified Web or wireless access protocol (WAP) browser as the client.

Given an enabled client, the LECP protocol is essentially a three-party authentication and channel-establishment in the standard setting of protocols such as Needham-Schroeder or Kerberos, in which all three parties run specific protocol engines. A large body of research exists for this area, from the design of individual private- and public-key protocols over robust design principles to security proofs, both cryptographic and tool-supported.[6] Nevertheless, to the best of our knowledge, the technique used in the LECP protocol, which we call *channel-based*, is new. Although Liberty published no rationale for avoiding a preexisting protocol, we believe this technique's main benefit is that it doesn't require special cryptography at the client.

In our analysis of the LECP protocol, we found a vulnerability to man-in-the-middle attacks. In response to our notification on 4 September 2003, Liberty corrected the flaw in v1.1, using a countermeasure we proposed. We use this experience to discuss generally the design of secure channel-based protocols, including the countermeasure used in the Liberty errata[7] and subsequent Liberty versions.

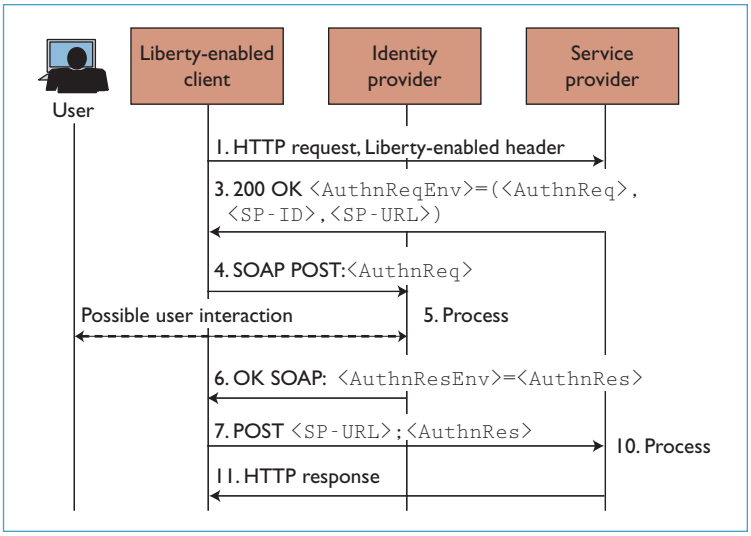

Figure 1. Liberty-enabled client and proxy (LECP) protocol. A user registers with the identity provider, which directly authenticates the user and confirms the user's identity to service providers.

LECP Protocol

To use a single-sign-on protocol, an individual registers with a so-called *identity provider*. The identity provider will later be the only party to directly authenticate the user, either globally or within a federation of enterprises. This confirms the user's identity to the other parties, or *service providers*.

Figure 1 gives an overview of the LECP protocol. It corresponds to the illustration in the Liberty Bindings and Profiles Specification,[5] except that we abbreviated some elements and added some detail for reference. The process includes the following steps (gaps in the step numbers are for compatibility with other Liberty protocols, some of which have more steps overall):

1. A client interacting with a service provider via HTTP (that is, browsing in a normal way) indicates in its header that it is Liberty-enabled. All further messages have this fixed LE header.
3. If the service provider wants to authenticate the client, and if it understands the LE header, it sends an authentication request `<AuthnReq>` to the client in an envelope `<AuthnReqEnv>`. The envelope also includes the service provider's identity `<SP-ID>` and the address `<SP-URL>` at which the service provider wants to receive the response.
4. The client takes the request from the envelope and forwards it in a SOAP[8] message to its identity provider.
5. The identity provider ascertains the user's identity (this can involve user interaction) and prepares a response `<AuthnRes>` for the service provider, which it sends to the client in a response envelope `<AuthnResEnv>`.
7. The client takes the response from the envelope and posts it to the service provider's address `<SP-URL>`.
10. The service provider processes the response.
11. Based on the response, the service provider continues the interaction.

We can generalize the LECP protocol in several ways. For instance, the protocol can be based entirely on Web services if the client and service provider interact via SOAP. Or it can be based entirely on classical Internet standards if the client and identity provider interact via HTTP. Authentication request and response formats and envelopes can differ slightly from the Liberty formats[9] — they might use SAML,[1] for example.

What then distinguishes the LECP protocol from other authentication and key-exchange protocols?

Channel-Based Approach

Classical three-party authentication protocols like Kerberos and Needham-Schroeder start with key exchange or key confirmation.[6] The client application then uses the new or confirmed key for encryption and authentication. The identity

provider is thus a key-distribution center or an online certification authority. No such three-party key exchange exists in the LECP protocol. Instead, the authentication response `<AuthnRes>`, which serves as an authentication token, travels over an independently established secure channel in step 7. In other words, the enabled client establishes a secure channel to the service provider without an authenticated client key, just as browser clients typically use secure sockets layer (SSL) or transport-layer security (TLS), and then send an authentication token through the channel without conveying a key.

The main advantage of channel-based protocols is that they work with SSL/TLS, the only current ubiquitous cryptographic infrastructure. In particular,

- Most service providers already have SSL server certificates.
- Because browsers already contain channel-based SSL/TLS implementations, enabled clients can easily be built as slight enhancements of browsers.
- Many servers have specific front ends, sometimes even hardware accelerators, for efficiently dealing with SSL connections.

In addition, an individual can use several unrelated authentication tokens, even from different identity providers, to provide information over a secure channel with the service provider.

A disadvantage of the SSL-channel-based protocol is that in closed scenarios such as intraenterprise single-sign-on, symmetric-key-only solutions can be computationally faster — recall that SSL has an asymmetric key-establishment phase. In a wider environment, however, the client's Internet round-trip to the identity provider is typically far more time-consuming than the SSL computations.

The channel-based protocols share this disadvantage with all three-party authentication protocols. Establishing a client certificate into the client (permanently for single-user clients, and per session from the identity provider for multiuser clients) so that the client can then identify directly to service providers is then more efficient. The client must be trusted in any case because it learns the user's single-sign-on password. Moreover, browsers already have key-loading capabilities. An enabled client can simplify the key-loading process further and provide an interface for managing multiple keys for different user roles or pseudonyms.

Enabled Clients

As mentioned earlier, other Liberty protocols and Web single-sign-on protocols often work with browsers as clients, whereas the LECP protocol requires specifically enabled clients. Enabled clients offer several benefits:

- They avoid an additional Internet round-trip between steps 1 and 3 when the service provider doesn't know the user's identity provider. The enabled client will either know the identity provider or will ask the user locally. With a browser, the service provider must ask the user in an extra message.
- They can transfer arbitrarily long authentication requests and responses directly. With a browser, steps 3, 4, 6, and 7 occur through a redirect action. The client transfers information primarily in the URL's search string, which should not be longer than 255 bytes. Because signed messages, especially those with certificates, are longer than 255 bytes, back-channels are typically needed. (POSTs need user interaction or scripting. User interaction is cumbersome, and scripting is already a form of enabled client and reduces security. Cookies are only possible if the identity and service providers are in the same domain.)
- They can alleviate some security problems. In particular, an enabled client can store authentication tokens securely and out of reach of potential scripts, and it can improve the user interface — for example, by notifying the user of the probability that he or she is connected to the right identity provider before the user inputs a password.

The disadvantage of enabled clients is clear: They must be installed on users' machines, and users are often reluctant to install anything, particularly software to make Web browsing simpler or more secure. This is demonstrated by the moderate success of providers of end-user security software and browser certificates. This was the original motivation behind browser-based (*zero-footprint*) Web single-sign-on. Nevertheless, new functionality makes its way to large user groups:

- *Browser evolution.* All browsers with a significant user base offer more than standard HTTP and HTML. Security and identity-management additions, such as SSL support, personality settings, and password management, are common. Adding related protocols is thus conceivable.

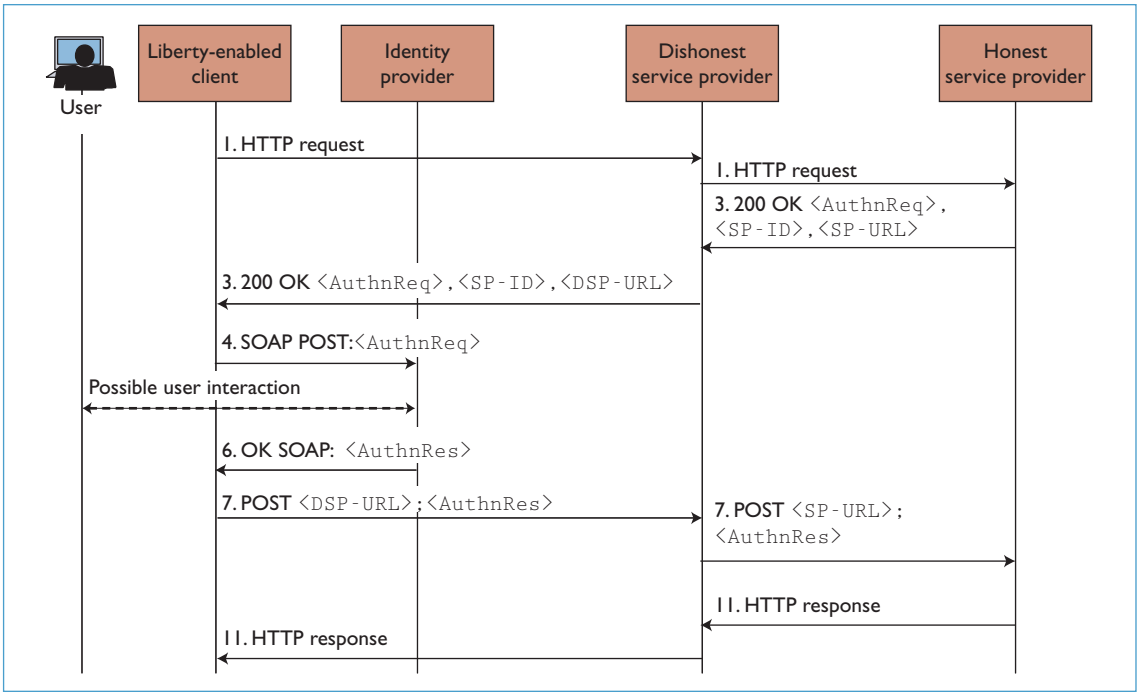

Figure 2. Overview of our man-in-the-middle attack. A dishonest service provider impersonates a user to an honest service provider by combining the honest service provider's request and ID with its own URL. It thus receives an authentication token that is in the user's name and suitable for the honest service provider.

- *Web services*. Browsers might soon be Web service-enabled, engendering additional security features, such as SOAP Message Security.[10]

Some other client applications might be Web service-enabled even earlier and run single-sign-on protocols other than browsers.

Man-in-the-Middle Attacks

Receiving a security token from an identity provider and sending the token to a service provider over a secure channel seems a natural and safe way to authenticate. Unfortunately, in this simple form the protocol is vulnerable to a *man-in-the-middle attack*. This was the vulnerability of Liberty v1.0.

In a man-in-the-middle attack, a dishonest service provider (DSP) impersonates a user to another service provider. An attacker can often guess or learn from other sources the providers at which a user is known, making the user worth impersonating. For instance, a dishonest Web shop might impersonate a user to the user's bank or the user's employer's intranet to access accounts or records.

DSPs were a small concern with Liberty v1.0 and v1.1, which focused on small, closed federations, but they become important in future versions aiming at a worldwide setting. Further, Liberty's prompt repair of its vulnerability to man-in-the-middle attacks proves its concern for the issue.

The DSP need not even be a federation member, as long as the user doesn't notice. This can easily happen even in small federations, either because the user doesn't think about it, doesn't know the federation members, or isn't involved in the protocol (and thus cannot stop it) because the client application or the identity provider cache the user authentication.

Figure 2 gives an overview of such an attack. Initially the user is browsing at the DSP, which starts a concurrent session with an honest service provider (such as a bank) where it wants to impersonate this user. At step 3, the attacker combines the bank's request and ID with its own URL <DSP-URL>. Because the envelope has no outer signature element to prevent this,[9] the identity provider processes the request in steps 4 to 6 as if it came directly from the honest service provider. In step 7, the client sends the response to the attacker at <DSP-URL>, and the attacker forwards it to the honest service provider, thus impersonating the user.

To verify that this attack works, we studied the messages and processing requirements. This is not a trivial task because the specification has four layers that are quite intertwined in places. Top down, the layers are:

- the specific LECP protocol,[5]
- the protocol's common requirements, common

interactions, and processing rules,[5]
- Liberty messages, which extend several SAML types, together with the general message processing rules,[9] and
- SAML requests and responses,[1] with SAML assertions as a lower sublayer.

We walked through the attack in a concrete LECP protocol derived by top-down substitution of the four layers. The analysis ran six pages. We need not bore you with the details as Liberty acknowledged and repaired the vulnerability. Instead, we present the constructive parts of the analysis in a generalized form as security measures. The "Man in the Middle in Related Protocols" sidebar discusses other protocols' approaches to these attacks.

Securing the Protocols

An immediate countermeasure against man-in-the-middle attacks aims to ensure that when the client sends the token meant for the honest service provider to an address <(D)SP-URL> in step 7, it is indeed a safe address — that is, only the honest service provider receives the token.

Existing Security Measures

Our countermeasures rely on security measures that already exist in the LECP protocol, but are not all explicit. These include:

- *Service-provider-specific tokens*. The identity provider puts the identity <SP-ID> into the token <AuthnRes>. Thus the token is valid for one service provider only. In Liberty, this is a field <AudienceRestrictionCondition>. An honest service provider accepts only authentication responses with its own identity. (Liberty does not specify this field's content, or the acceptance restriction, but this is the natural instantiation.) The identity provider obtains the identity <SP-ID> in the authentication request <AuthnReq> as a field <ProviderID>; we call this the inner <SP-ID> in <AuthnReq>. The outer occurrence of <SP-ID> in the envelope <AuthnReqEnv>, shown in Figure 1, has no security function.
- *Secure channels*. The token travels only over secure channels. In the LECP protocol we achieve this by requiring the service provider's address <SP-URL> to be secure — that is, an HTTPS address, and by using a secure channel between client and identity provider.
- *Token authentication*. The identity provider authenticates the token <AuthnRes> for the service provider — at least the user identity and <SP-ID> — and the service provider verifies this. (In Liberty, the authenticated part is the contained assertion.)

Given these security measures, we state our goal of using a safe address more precisely: If the identity <SP-ID> in the token belongs to the honest service provider, the address <SP-URL> in step 7 is an HTTPS address controlled by a trusted process of the service provider — the only entity that can get a server certificate acceptable for this address.

Countermeasures

We've identified four countermeasures against our attack — that is, four ways for the client to send the token to the service provider's safe address only.

- *Client derives service provider's address*. If the client has an available list or infrastructure of safe service provider addresses, it can derive <SP-URL> from the service provider's identity <SP-ID>. This must be the inner <SP-ID> from <AuthnReq> because it is the one used by the identity provider. The service provider can also propose a specific <SP-URL> in the request envelope as a hint.
- *Service provider authenticates for client*. The honest service provider authenticates the request envelope containing a safe <SP-URL>, and the client verifies its authenticity with respect to the inner identity <SP-ID> in the request.
- *Identity provider derives service provider's address*. Instead of the client, the identity provider can use a list or infrastructure of safe service provider addresses to derive <SP-URL> from the inner <SP-ID>. The identity provider then includes <SP-URL> in the response envelope <AuthnResEnv>, from which the client takes it. Again, the service provider can propose a specific <SP-URL> as a hint.
- *Service provider authenticates for identity provider*. The honest service provider includes a safe <SP-URL> in the request <AuthnReq>, which it authenticates for the identity provider. This authentication is mandatory in the LECP protocol. Again, the identity provider includes <SP-URL> in the response envelope <AuthnResEnv> for the client.

The Liberty erratum and thus Liberty v1.1 employs the third countermeasure, but without using the service provider's <SP-URL> as a hint. Indeed, this

Man-in-the-Middle in Related Protocols

Man-in-the-middle security is a concern for all Web single-sign-on protocols.

Microsoft Passport v2.0 provides security against man-in-the-middle attacks with its "secure sign-in" (http://msdn.microsoft.com); previous versions offer no protection against such attacks. The security is not yet optimal, however. The protocols are not public, but the documentation shows that the main security measures involve generating tokens that are specific to one service provider by encryption for the identity <SP-ID> (symmetrically and with out-of-band key exchange) and sending them to the service provider over HTTPS addresses only.

Passport uses a process similar to our fourth countermeasure (see the main text) with a hint from the service provider to ensure that <SP-URL> belongs to <SP-ID>. The identity provider verifies that the <SP-URL> provided by the service provider is under the root of the organization with identity <SP-ID>. Subscribing service providers register this root as "the topmost domain name of your site" (see "Registering Your .NET Passport Site" in the .Net documentation at http://msdn.microsoft.com). However, this would mean that every attacker that controls any URL at a site can obtain a token for the site. Registering a subroot of a secured service part would solve this problem.

Man-in-the-middle attacks on one SAML protocol have only recently been discovered by including lower layers in the analysis.[1] They are possible in specific standard-conformant implementations that require underlying authentication that is weaker than secure channels. Another attack is based on the fact that browsers often include a previous URL in a message, the so-called referer header. A similar analysis of the Liberty protocols including lower layers is outstanding. We assume the attack with the referer header will also work on a Liberty protocol.

The browser-based Liberty profiles have one explicitly stated user-interface vulnerability in allowing embedded forms for authentication, which is an invitation for fake-screen attacks.[2] We would prefer that to be deprecated. The proposed federation contracts do not help, because dishonest service providers would not adhere to them, and they need not even be federation members because users will not always verify to what federations their current service provider belongs.

References

1. T. Groß, "Security Analysis of the SAML Single Sign-on Browser/Artifact Profile," to be published in *Proc. Ann. Computer Security Applications Conf.* (ACSAC), 2003.
2. Liberty Alliance Project, "Liberty Architecture Overview," v1.1, 15 Jan. 2003, www.projectliberty.org/specs/archive/v1_1/liberty-architecture-overview-v1.1.pdf.

seems optimal for the Liberty v1.0/1.1 focus on small federations in which all identity and service providers exchange information during setup. For greater scalability, it seems better to use only a standard server-key infrastructure and to send all other information in the protocol (as in the second and fourth countermeasures). The metadata exchange in Liberty v1.2 goes in this direction. When a channel-based protocol is chosen because it only needs client cryptography in the form of secure channels, the fourth countermeasure works best.

Security Considerations

A single-sign-on protocol is secure if it provides a secure channel between an honest service provider and an honest user with a certain name (that is, if the service provider thinks it is interacting with a user named *N*, it really is interacting with *N*). Clearly, for this purpose, we must trust the client application and every identity provider the service provider trusts to certify this type of name and the quality of its registration and authentication procedures.

We now show that the generalized LECP protocol with any of our four countermeasures is secure if all submodules are appropriately instantiated (in particular, the secure channels and user and message authentication), and under a few additional "reasonable" processing constraints.

A service provider believes it is talking with a certain user when it receives a step 7 token that has both its identity <SP-ID> and the user's name, and is authenticated by an identity provider the service provider trusts for this name. The identity provider issues such a token (step 6) only when it has a secure channel with the user it originally registered under this identity. It sends the token only in that secure channel — that is, to the trusted client acting for the user. Here we assume that no other current or future Liberty profiles using the same token types will leak tokens to parties other than the user and service provider involved in the transaction, because we can't prevent the same token from also occurring in a different profile. (We consider this dependence between profiles a violation of robust protocol design.[11]) In step 7, the client forwards the token to an address <SP-URL> determined with one of our countermeasures. It's easy to see that each countermeasure ensures that this is a safe address of the service provider. This prevents impersonation with the token because no party except the user and service provider obtains the token.

It would be an interesting challenge to fully formalize Web single-sign-on protocols while stay-

ing as close to the actual standards as possible, and to then formalize this sketch into a full proof.

Conclusions

Serious concerns about Web single-sign-on beyond protocol security and efficiency exist and have even led to political and judicial debates.[12] Kormann and Rubin[13] provide general security discussions, particularly regarding operational and user-interface security. Although some of these issues are specific to Microsoft Passport, many apply to all browser-based protocols. Several concerns disappear with an enabled client, as in the LECP protocol, if the operational and user-interface aspects are well designed. (This is not part of current proposals, however.) An earlier article deals with privacy requirements and their consequences on better protocol design.[14] Another concern is that a large identity provider might gain a tollbooth position and be a single point of failure. This was an explicit motivation for the Liberty Alliance, which focuses on many small federations with one or more identity providers.

A general conclusion for the design of security protocols based on XML and Web services is that the easy extensibility in almost all places, and the resulting fragmentation of a specification, can make analysis very difficult. Implementers face the same difficulty of fitting together the layers, trying not to forget any general rule from any layer, and implementing the "reasonable" additional assumptions. We believe that a clearer modularization — that is, modules with a clear specification of the security they provide and few extension points — would be helpful for the security of both future protocols and their implementations.

Acknowledgments

We thank our colleagues Heather Hinton, Thomas Kretschmer, and Anthony Nadalin for helpful discussions, and Michael Barrett of American Express and Gary Ellison of Sun for their friendly reaction and prompt response to the vulnerability report. This article represents the authors' views, which are not necessarily shared by IBM. IBM is not a member of the Liberty Alliance. Before publication of the errata, Jonathan Sergent of Sun independently found the vulnerability in Liberty v1.0.

References

1. P. Hallam-Baker and E. Malers, eds., "Assertions and Protocol for the Oasis Security Assertion Markup Language (SAML)," Oasis standard, Nov. 2002; www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf.
2. P. Mishra, ed., "Bindings and Profiles for the Oasis Security Assertion Markup Language (SAML)," Oasis standard, Nov. 2002, www.oasis-open.org/committees/security/docs/cs-sstc-bindings-01.pdf.
3. M. Erdös and S. Cantor, "Shibboleth Architecture Draft v05," May 2002; http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf.
4. Liberty Alliance Project, "Liberty Architecture Overview," v1.0, 11 July 2002; www.projectliberty.org.
5. Liberty Alliance Project, "Liberty Bindings and Profiles Specification," v1.0, 11 July 2002; www.projectliberty.org.
6. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
7. Liberty Alliance Project, "Liberty Version 1.0 Errata," ed. 00, 11 Oct. 2002, www.projectliberty.org/specs/archive/v1_0/draft-liberty-version-1-errata-00.pdf.
8. D. Box et al., "Simple Object Access Protocol (SOAP) 1.1," W3C note, May 2000; www.w3.org/TR/SOAP.
9. Liberty Alliance Project, "Liberty Protocols and Schemas Specification," v1.0, 11 July 2002; www.projectliberty.org.
10. A. Nadalin et al., eds., "Web Services Security: SOAP Message Security," Oasis working draft, Aug. 2003; www.oasis-open.org/committees/download.php/3281/WSS-SOAPMessageSecurity-17-082703-merged.pdf.
11. M. Abadi and R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," *IEEE Trans. Software Eng.*, vol. 22, no. 1, 1996, pp. 6–15.
12. Federal Trade Commission, "Microsoft Settles FTC Charges Alleging False Security and Privacy Promises," 8 Aug. 2002, www.ftc.gov/opa/2002/08/microsoft.htm.
13. D.P. Kormann and A.D. Rubin, "Risks of the Passport Single Signon Protocol," *Computer Networks*, Elsevier Science Press, vol. 33, 2000, pp. 51–58.
14. B. Pfitzmann and M. Waidner, "Privacy in Browser-Based Attribute Exchange," *Proc. ACM Workshop on Privacy in the Electronic Soc.* (WPES), ACM Press, 2003, pp. 52–62.

Birgit Pfitzmann is a researcher at the IBM Zurich Research Lab. Her research interests include federated identity management, Web services security, enterprise privacy management, and linking formal verification methods and cryptography. She received a PhD in computer science from Hildesheim University. She has authored more than 80 publications in the areas of security and privacy and has served on numerous program committees. She is a senior member of the IEEE and a member of ACM. Contact her at bpf@zurich.ibm.com.

Michael Waidner is manager of the Network Security and Cryptography research group at the IBM Zurich Research Lab. His research interests include information security, privacy, and cryptography. He received a PhD in computer science from the University of Karlsruhe. He is responsible for IBM's research agenda in privacy technology, and led the research team that developed IBM's Enterprise Privacy Architecture. Contact him at wmi@zurich.ibm.com.