RZ 3458 (# 93844) 11/04/02
Computer Science 11 pages

Research Report

Token-based Web Single Signon with Enabled Clients

Birgit Pfitzmann, Michael Waidner

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Token-based Web Single Signon with Enabled Clients

Birgit Pfitzmann, Michael Waidner

{bpf,wmi}@zurich.ibm.com, IBM Zurich Research Lab

Abstract. We study a type of web single signon recently introduced by one of four proposed standard protocols of the Liberty Alliance. In contrast to the other three Liberty protocols and prior protocols like Microsoft Passport and the SAML standard, the client is not only a browser, but aware of the protocol, for instance a web-service client. We investigate how this protocol differs from standard three-party authentication, and possible benefits. We call the new protocol class token-based web single signon with enabled clients. We show a man-in-the-middle attack on the original Liberty V1.0 protocol and countermeasures against it. (Such a countermeasure was now added as an erratum, and no deployed implementation will use V1.0.) We also give general guidance for designing secure protocols in this class.[1]

Key words. Authentication, web single signon, man-in-the-middle attack, Liberty, token

1 Introduction

A web single-signon protocol allows a person to log in to many different services offered on the Internet while needing to authenticate only once, or at least always in the same way. The naive approach at simplifying web signon is that a user chooses the same username and password with all these services. There are two problems with this:

1. Each service can impersonate the user towards the others. This is not acceptable for a user's overall web experience. Even among the services of one enterprise, one often prefers a smaller trust core.
2. Services needing an initial identification of the user with respect to a preexisting name still all have to perform this identification.

Single signon protocols aim at solving at least one of these problems. Typically the user registers with one party only, called identity provider by Liberty, who becomes the only party to directly authenticate the user later.

Well-known prior protocols for web single signon are Microsoft Passport, SAML, an OASIS standard in the voting phase, and the Internet2 project Shibboleth [Mic01, SAM02, Shi02]. In July, the Liberty Alliance published its widely awaited proposals. They contain message formats extending those of SAML, a classification of authentication classes, and four concrete protocols in a six-part specification starting with [Lib02]. Because of the strong membership in this alliance, these protocols have a good chance of becoming important in practice.

[1] This erratum in [Lib02f] is a reaction on our vulnerability notification to Liberty on Sept. 4. Before the errata publication, the problem was found independently by Jonathan Sergent of Sun.

Here we concentrate on the fourth of the concrete protocols, the Liberty-Enabled Client and Proxy Profile, now abbreviated LEC protocol [Lib02d]. Its specific aspect is that it assumes a special client aware of this protocol, while the first three Liberty protocols and the prior proposals we mentioned assume an unmodified web or WAP browser as the client. Thus essentially we are looking at three-party authentication and channel establishment in the standard setting, where all three parties (client, identity provider and service provider) run specific protocol engines. However, the LEC protocol uses a novel technique, which we call token-based. Its main benefit is not to require cryptography in the client beyond secure channels, such as SSL or TLS, used through their standard interfaces.

We focus on the protocol security and efficiency of this new protocol class. We show that the original LEC protocol (Version 1.0) is vulnerable against a man-in-the-middle attack. We discuss generally how to design secure protocols in this class, including the particular countermeasure selected in the Liberty errata. We also discuss the applicability of this protocol class, because the Liberty Alliance simply proposed the protocol without publicly identifying a class or weighing its benefits. At the end, we briefly discuss the man-in-the-middle security of prior web single-signon protocols such as Passport.

Outside our focus topics of protocol security and efficiency, there are serious other concerns about web single signon. They have even led to political and judicial debates. For general security discussions, in particular operational and user-interface security, see [KR00, Sle00]. While some of these topics are specific to Microsoft Passport, many apply to browser-based protocols generally. Several of these concerns can disappear with an enabled client as in the LEC protocol, if its operational and user-interface aspects are well designed. Privacy requirements and their consequences on better protocol design are treated systematically in [PW02]. A third concern is that a large identity provider may 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. However, so far the specification only covers each federation individually and does not scale easily.

2 Token-based Web Single Signon with Enabled Clients

We first describe the protocol that motivated our work, and then define a class of similar protocols and discuss how they might be used.

2.1 Overview of the Liberty-Enabled Client Protocol

Figure 1 gives an overview of the LEC protocol. It corresponds to Figure 5 from [Lib02d], except that we abbreviate some elements and show some additional details for reference. The client application is called Liberty-enabled client, abbreviated LE client. The gaps in the step numbers are for compatibility among all Liberty protocols.

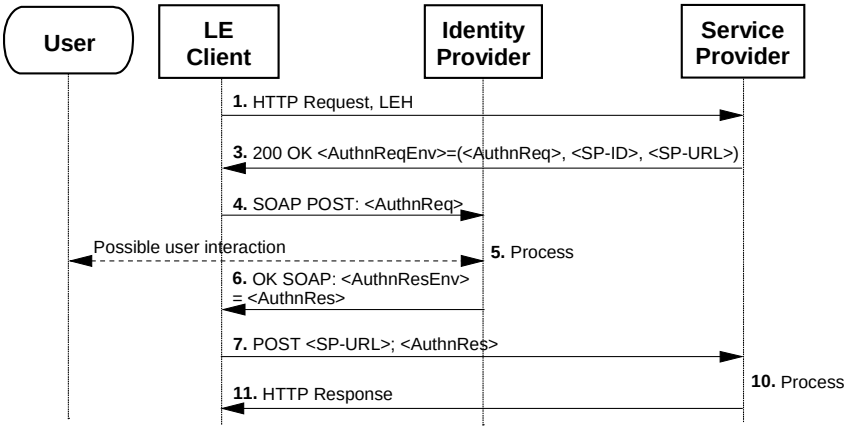

Figure 1. LEC (Liberty-enabled client and proxy) protocol

1. Initially, the client is interacting with a service provider via HTTP, e.g., the user is browsing in a normal way. The client indicates by a specific header LEH that it is Liberty-enabled. In the figure we omit that all further messages also have this fixed header.
3. When the service provider wants to authenticate the client, or typically the user at this client, and if it understands the LEH header, it sends an authentication request `<AuthnReq>` to the client in an envelope `<AuthnReqEnv>`. The main other elements of this envelope are an identity `<SP-ID>` of the service provider and the address `<SP-URL>` where the service provider wants to receive the response.
4. The client takes the request from the envelope and forwards it to its identity provider in a SOAP message [SOA00].
5. The identity provider ascertains the user's identity (this may involve user interaction) and prepares a response `<AuthnRes>` for the service provider. It sends this to the client in a response envelope `<AuthnResEnv>`, where it is the only relevant element.
7. The client takes the response from the envelope and posts it to the URL `<SP-URL>` of the service provider.

10., 11. The service provider processes the response and continues the interaction depending on it.

2.2 Generalization

We can generalize the LEC protocol in several ways. For instance, the protocol can be based entirely on web services if the client and the service provider also interact via SOAP. Or it can be based entirely on classical Internet standards if the client and the identity provider also interact via HTTP. The authentication request and response formats might be slightly different from the formats defined by Liberty [Lib02b], e.g., original SAML [SAM02], and similarly the envelopes might be

slightly different. What, however, is the really distinguishing feature from other authentication and key-exchange protocols?

Classical three-party authentication protocols like Kerberos and Needham-Schroeder start with a key-exchange or key-confirmation phase (see, e.g., [MOV97]). The client application then uses the new or confirmed key for encryption and authentication. (Thus with symmetric keys only, the identity provider is a key-distribution center, otherwise an online certification authority.) No such three-party key exchange is present in the LEC protocol. Instead, the authentication response `<AuthnRes>`, which serves as an authentication token, is sent over an independently established secure channel in Step 7. In other words, a secure channel is established without an authenticated client key, just as SSL/TLS is usually used with browsers, and then an authentication token is sent in this channel without conveying a key.

2.3 Token-based versus Key-exchange Protocols

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

- a majority of service providers already has SSL server certificates,
- via the browsers, a suitable cryptographic implementation is available on all client machines; this offers easy transition if the enabled client is a slight variation of a browser, and
- many servers have specific front-ends for dealing with SSL connections efficiently, up to hardware accelerators.

Another advantage is that one can use several unrelated authentication tokens, even from different identity providers, to provide information about the user in the same secure channel with the service provider.

A disadvantage in a closed scenario like intra-enterprise single signon for employees is that a symmetric-key-only solution can be computationally faster; recall that SSL has an asymmetric key-establishment phase. In a wider environment, token-based protocols share the disadvantage of all three-party authentication protocols of requiring an additional Internet roundtrip to the identity provider before the interaction can continue. This is typically far more time consuming than the SSL computations. Once establishing a client certificate into the client (permanently for a one-user client, and per session from the identity provider after user authentication for a multi-user client) is then more efficient. Note that the client must be trusted anyway because it learns the user's single-signon password. Also note that browsers already have key-loading capabilities. An enabled client could simplify that further and additionally provide an interface for managing multiple keys for different user roles or pseudonyms.

2.4 Discussion of Enabled Clients

As mentioned, the other Liberty protocols and well-known prior web single-signon protocols work with usual browsers as clients.

We see the following benefits of a protocol with enabled clients:

- An enabled client avoids a further Internet roundtrip between Step 1 and 3 in the case where the service provider does not know the user's identity provider. The enabled client may either know this or ask the user locally. With a browser, the service provider must ask the user directly in order to prepare Step 3 as a suitable browser redirect.
- An enabled client can transfer arbitrarily long authentication requests and responses directly. With a browser, the relation of Steps 3-4, and 6-7, is given by a browser redirect. The primary way to transfer information is in the search string of the URL, which should not be longer than 255 bytes. This is not enough for a signed message, in particular with certificates, and thus back-channels are typically needed. (POSTs need user interaction or scripting; the latter already gives an enabled client. Cookies are only possible if identity provider and service provider are in the same domain.)
- An enabled client can decrease some security problems. In particular, it may store authentication tokens particularly securely and out of reach of potential scripts, and it may improve the user interface, e.g., by specific notice of the quality of the connection to the supposed identity provider.

The disadvantage of enabled clients is clear: They have to be installed at the users, who are very reluctant to install anything, at least additions to make web browsing simpler or more secure. This was the reason to design browser-based ("zero-footprint") web single signon in the first place. Nevertheless, new functionality makes its way to large user groups:

- Browser evolution. Only a few browsers have a significant user base, and they all offer much more than standard HTTP and HTML. In particular, security and identity-management additions are already common, such as SSL support, personality settings, and password management. Thus adding related protocols is conceivable.
- Web services. Browser may soon be web-service enabled anyway, and this may come with additional security features, such as [WSS02]. Some other client applications may be web-service enabled even earlier and can run other single-signon protocols than browsers.

Active content is another possibility, but was attempted before the recent move towards zero-footprint web single signon; it also makes browsers much more insecure.

3 A Man-in-the-Middle Attack

We now show a man-in-the middle attack on the original version of the LEC protocol.

3.1 Setting of the Attack

We consider an impersonation attempt by a dishonest service provider *DSP*. The goal of *DSP* is to impersonate a person (user) who is currently browsing there to

another service provider *SP*. This only makes sense if the person is known at *SP*, which an attacker can often guess or find out from other sources. For instance, a dishonest web shop may want to access the bank accounts of its customers, or their employers' intranets.

Given the Liberty focus on small, closed federations, dishonest service providers are a smaller concern than in a world-wide setting. Nevertheless, Liberty considers dishonest service providers; this is also shown by the prompt repair of the vulnerability.

The dishonest service provider need not even be a federation member, as long as the user does not notice this. This is likely to happen often even for small federations, either because the user does not think about it or does not know the federation members, or because the user is not involved in the protocol (and thus cannot stop the protocol) because the client application or the identity provider cache the user authentication.

3.2 The Attack

Figure 2 gives an overview of our attack.

Initially the user is browsing at the dishonest service provider *DSP*. This attacker starts a concurrent session to an honest service provider *SP* (e.g., a bank) where it wants to impersonate this user. The essential step is Step 3*, where the attacker combines *SP*'s request and ID with its own (i.e., the attacker's) URL `<DSP-URL>`. The envelope has no outer signature element which could prevent this, see Section 3.2.4.1 of [Lib02b]. Thus in Steps 4-6, the identity provider processes the request as if it came directly from the honest service provider *SP*, but in Step 7 the client sends this response to the attacker at `<DSP-URL>`. The attacker then forwards it to the honest service provider, thus impersonating the user at this service provider.

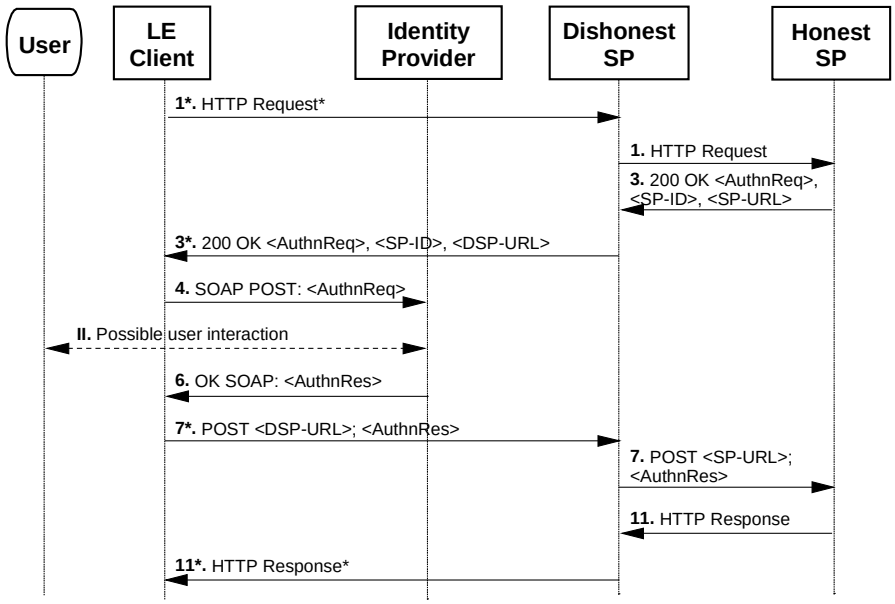

Figure 2. Overview of the man-in-the-middle attack

To verify that the attack indeed works, we studied the messages and processing requirements in detail. This is not trivial due to a specification with 4 layers which are quite intertwined in places. Top down, they are:

a) The specific LEC protocol from Section 3.2.5 of [Lib02d].
b) The common requirements from Section 3.1 and the common interactions and processing rules from Sections 3.2.1 of [Lib02d].
c) Liberty messages, which extend several SAML types, together with the general processing rules from Sections 3.1 and 3.2 of [Lib02b].
d) SAML requests and responses [SAM02], with SAML assertions as a lower sublayer.

We walked through the attack in a concrete LEC protocol derived by top-down substitution of all these specification parts. This ran to six pages. As the vulnerability was acknowledged and repaired by Liberty, we need not bore the reader with these details. Instead, we present the constructive parts of the analysis in a generalized form as security measures.

4 Securing the Protocols

The goal of an immediate countermeasure against the attack from Section 3 is to ensure that when the client sends the token meant for the honest service provider *SP* to an address `<(D)SP-URL>` in Step 7, this is indeed a safe address for *SP*, i.e., only *SP* receives the token. We present such countermeasures in Section 4.2.

4.1 Underlying Existing Security Measures

Our countermeasures rely on security measures present in the LEC protocol (although not all explicit). We also assume them in generalizations.

1. **Service-provider-specific token:** The token <AuthnRes> is only valid for one service provider by containing the identity <SP-ID>; in Liberty this is a field <AudienceRestrictionCondition>. An honest service provider only accepts an authentication response with its own <SP-ID>. (Liberty does not specify the content of this field, nor the acceptance restriction, but this is the natural instantiation.) The identity provider obtains <SP-ID> in the authentication request <AuthnReq> as a field <ProviderID>. (The outer occurrence of <SP-ID> in the envelope <AuthnReqEnv>, as shown in Figure 1, has no security function.)
2. **Secure channels:** The token is only sent over secure channels. In the LEC protocol this is realized by requiring <SR-URL> to be https, and by using a secure channel between client and identity provider.
3. **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.)

4.2 Countermeasures

We now present four countermeasures, i.e., four possible ways to attain the goal that the client sends the token only to a safe address of *SP*. Given Section 4.1, we can make the notion of safe address precise: If the identity <SP-ID> in the token belongs to *SP*, then the address <SP-URL> used in Step 7 is an https address and is controlled by *SP* (i.e., a trusted process of *SP*), and only *SP* can get a server certificate acceptable for this address.

1. **Client derives:** The client may have a list or infrastructure of safe service-provider addresses available, and derives <SP-URL> from the service provider's identity <SP-ID> with that. This must be the "inner" <SP-ID> from <AuthnReq>, because this is the one used by the identity provider. Additionally, *SP* may still propose a specific <SP-URL> in the request envelope as a hint.
2. ***SP* authenticates for client:** The honest service provider *SP* authenticates the request envelope containing a safe <SP-URL>, and the client verifies the authenticity with respect to the "inner" identity <SP-ID> in the request.
3. **Identity provider derives:** Instead of the client, the identity provider may 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 where the client takes it. Again, *SP* may propose a specific <SP-URL> as a hint.
4. ***SP* authenticates for identity provider:** The honest service provider *SP* includes a safe <SP-URL> in the request <AuthnReq>, which it authenticates

for the identity provider. In the LEC protocol this authentication is already mandatory. Again the identity provider includes <SP-URL> in the response envelope <AuthnResEnv> for the client.

The Liberty erratum is Solution 3 without using the *SP*-provided <SP-URL> as a hint. Indeed this seems optimal for the Liberty focus of small federations where all parties exchange information in a set-up phase. For greater scalability, it seems better to use only a standard server-key infrastructure and to send all other information in the protocol, i.e., Solutions 2 and 4. Where it is relevant that token-based protocols only need client cryptography in the form of secure channels, Solution 4 should be chosen.

4.3 Security Considerations

Security of a single signon protocol means this: If an honest service provider *SP* believes that it has a secure channel with a certain honest user *U* under a name id_U, then this is true. Clearly, we also have to trust the client application, and every identity provider whom *SP* trusts to certify a user under this name id_U and the quality of its registration and authentication procedures.

We sketch that the generalized LEC protocol with each countermeasure from Section 4.2 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.

It is easy to see that each countermeasure is correct with respect to the goal of Section 4.2, i.e., the address <SR-URL> used in Step 7 is safe.

A service provider *SP* believes that it talks with user id_U when it gets a Step-7 token with its own identity <SP-ID> and this user name, and authenticated by an identity provider it trusts for this identity. The identity provider only issues such a token (Step 6) when it has a secure channel with the user *U* whom it originally registered under this identity. It only sends the token in that same channel, i.e., to the trusted client acting for *U*. (Here we assume that all other current and future Liberty profiles using the same token types do not leak tokens to parties other than *U* and *SP*. We do not see how to demonstrate security without this assumption in spite of a request ID intended to prevent replay, because its usage in the response is not secured, and it does not remain secret against a man in the middle.) By the correctness of the countermeasures, the client only forwards the token to a safe address <SR-URL>, i.e., an https address of the party whose identity <SR-ID> included in the token, i.e., to *SP*. Thus no party except *SP* and *U* obtains the token, which prevents impersonation with this token.

5 Remarks on Related Protocols

Let us briefly discuss man-in-the-middle security in the related class of browser-based web single signon.

We have a security analysis for our privacy-enabling protocol BBAE on about the same level of detail as Section 4.3 [PW02], and (unpublished) for the SAML artifact profile [SAM02a].

For the other three Liberty V1.0 profiles, we are neither aware of a protocol weakness nor do we claim security. They have one explicitly stated user-interface vulnerability in allowing embedded forms for authentication, which is an invitation for fake-screen attacks; see Section 5.7.1.3 of [Lib02]. 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.

In Microsoft Passport, security against man-in-the-middle attacks was only addressed with the security level "Secure Sign-In" of Passport V2.0 [Mic01]. We believe that the security is not yet optimal. The protocols are not public, but from the documentation we see the main security measures as follows: Tokens are specific to one service provider by encryption for the identity `<SP-ID>`, and sent to the service provider only over https addresses `<SP-URL>`. Ensuring that `<SP-URL>` belongs to `<SP-ID>` is done similar to our Solution 4 with a hint from the service provider. 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 top-most domain name of your site"; see "Registering Your .NET Passport Site" in [Mic01]. However, this would mean that every attacker, e.g., an insider, who controls any URL at a site can obtain a token for the site as such. Registering a special sub-root of a secured service part would solve that problem.

6 Outlook

We have defined and discussed the benefits of the class of token-based web single-signon protocols with enabled clients as a generalization of the Liberty LEC protocol. We showed a vulnerability in the original V1.0 LEC protocol and countermeasures. They have now been reflected in the Liberty errata. We gave a brief security analysis of the resulting and generalized protocols. This required a few more assumptions and processing rules which are reasonable given the specification, but not fully explicit there.

A general conclusion for the design of XML and web-services security protocols is that the easy extensibility in almost all places and the resulting fragmentation of a specification (compare the end of Section 3) can make an analysis very hard. Implementers face the same difficulty of fitting together the layers, trying not to forget any general rule from any layer, and to implement the "reasonable" additional assumptions. We believe that a clearer modularization, i.e., modules with a rather small number of extension points and a clear specification also of the security they provide, would be helpful for the security of both future protocols and their implementations.

Acknowledgements

We thank our colleagues Heather Hinton, Thomas Kretschmer, and Anthony Nadalin for helpful discussions, and Michael Barrett (American Express) and Gary Ellison (Sun) for their friendly reaction on the vulnerability report and the prompt repair.

References

[KR00] David P. Kormann, Aviel D. Rubin: Risks of the Passport Single Signon Protocol; Computer Networks, Elsevier Science Press 33 (2001) 51-58

[Lib02] Liberty Alliance Project: Liberty Architecture Overview, Version 1.0, 11 July 2002, http://www.projectliberty.org/

[Lib02b] Liberty Alliance Project: Liberty Protocols and Schemas Specification, Version 1.0, 11 July 2002

[Lib02d] Liberty Alliance Project: Liberty Bindings and Profiles Specification, Version 1.0, 11 July 2002

[Lib02f] Liberty Alliance Project: Liberty Version 1.0 Errata, Edition 00, 11 October 2002

[MOV97] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: Handbook of Applied Cryptography, CRC Press, Boca Raton 1997

[Mic01] Microsoft Corporation: Various .NET Passport documentation (started 1999), in particular Technical Overview, Sept. 2001, and SDK 2.1 Documentation; http://www.passport.com and http://msdn.microsoft.com/downloads

[PW02] Birgit Pfitzmann, Michael Waidner: Privacy in Browser-Based Attribute Exchange; accepted for ACM Workshop on Privacy in the Electronic Society, Washington, Nov. 2002, preliminary version IBM Research Report RZ 3412 (# 93644), June 10, 2002

[SAM02] Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML); Committee specification 01, May 2002, http://www.oasis-open.org/committees/security/docs

[SAM02a] Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML); Committee specification 01, May 2002

[Shi02] Shibboleth-Architecture DRAFT v05; May 2002 (v1 in 2001) http://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-arch-v05.pdf

[Sle01] Marc Slemko: Microsoft Passport to Trouble; V1.18, Nov. 2001, http://alive.znep.com/~marcs/passport/

[SOA00] Simple Object Access Protocol (SOAP) 1.1; W3C Note, May 2000, http://www.w3.org/TR/SOAP

[WSS02] Web Services Security, draft specifications of the OASIS WSS Technical Committee, http://www.oasis-open.org/committees/wss/, 2002