Noname manuscript No.
(will be inserted by the editor)

Secure Mashups For ID Management With In-Browser Cryptography Support *

Saman Zarandioon, Danfeng (Daphne) Yao, Vinod Ganapathy
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
{samanz,danfeng,vinodg}@cs.rutgers.edu

Abstract This paper addresses the identity management problems in modern Web-2.0-based mashup applications. We present Web2ID, a new identity management framework tailored for mashup applications. Web2ID leverages a secure mashup framework and enables transfer of credentials between service providers and consumers. We also describe a new relay framework in which communication between two service providers is mediated by a relay agent within the mashup. We show that Web2ID is privacy-preserving and prevents service providers from learning a user's surfing habits.

We present an implementation of Web2ID and the relay framework using a JavaScript-based library that executes within the browser. Our implementation does not require client-side changes and is therefore fully compatible even with legacy browsers. Finally, we demonstrate how Web2ID can provide identity management for Web-based desktops, a popular new class of client-side mashup applications.

1 Introduction

Mashup applications integrate information from multiple autonomous data sources within the Web browser for a seamless browsing experience. For example, iGoogle allows users to create personal pages containing "gadgets" from multiple Web domains, such as NY-Times, Weather.com and Google Maps. Despite their popularity, mashups are still not in widespread use for sensitive Web applications, such as banking, investment, online shopping and bill payment. Such mashup applications currently require user authentication to prevent unauthorized access to sensitive information. A Web user who includes such sensitive applications in a mashup must authenticate herself individually to each of these applications. For example, mint.com and yodlee.com allow users to view a summary of their financial activities by accessing back-end services, such as banks and credit card companies.

For historical reasons the Internet lacks unified provisions for identifying who communicates with whom [26]. Therefore, currently service providers use ad-hoc solutions to

* The preliminary version of the work appeared in the Proceedings of the Fifth ACM Workshop on Digital Identity Management (DIM) [32]. Collocated with ACM Conference on Computer and Communications Security (CCS). Chicago, IL. Nov. 2009.

Address(es) of author(s) should be given

identify and authenticate their users. These ah-hoc solutions require the user to create a new identity for each service provider. This leads to multiple isolated identities for each user; a problem which is particularly troublesome in mashup applications as they need to authenticate and cores-reference users across multiple service providers in order to share users' protected resources with their consent.

We present *Web2ID*, a new protocol for identity management in mashup applications. By leveraging a secure mashup framework developed in our prior work [31], Web2ID enables transfer of credentials without requiring page redirections, and works seamlessly with Ajax-based Web-2.0 applications. We also describe a new *relay mashup framework*, based on which a trusted client-side *relay agent* can be built in the mashup to transmit credentials between providers.

Most existing identity management protocols for the Web, including OpenID [1], use a unique URL to represent the identity of a principal. The advantage of using a URL as opposed to a name or email address is that a URL is tangible, clickable, user-friendly, and can contain information that facilitates the authentication process. This URL is called the principal's *identity URL* . The static page that is located at identity URL is called the *identity page*. The server that hosts the identity page is called the *identity host*. Therefore, during authentication, users claim ownership of their identity URL and proves their claims to a service provider by following the corresponding authentication protocol. However, all these authentication protocols require a trusted third party, called *Identity Provider (IdP)*, to validate the user's claim. Users first create an account with IdP and use the identity page to delegate the authentication of their identity URL to that IdP. But this feature may compromise user privacy as the IdP can learn the surfing habits of the user.

In this work, user privacy refers to the protection of a user's transaction history. Better privacy refers to the separation of providers (service providers or identity providers) in a way so that they do not directly communicate to each other regarding to the users and their activities. This notion of privacy follows the one used by Goodrich, Tamassia, and Yao in a federated identity management or single sign-on (SSO) model [13]. Unlike [13], our solution is decentralized and does not require any centralized server.

The main advantage of Web2ID is that it uses public-key cryptography to enable users to prove the ownership of their identity URL without relying on third parties. Authentication in Web2ID and PGP [2, 29] are similar as they both eliminate the need for a centralized identity provider, and thus support decentralized trust management. However, the main difference is that Web2ID is specifically designed for modern web 2.0 mashup application. In Web2ID, the identity of each user is represented by a URL whereas in PGP the public keys directly represent the identity of users. This feature makes Web2ID more user friendly as URLs are easier to remember. Moreover, Web2ID provides powerful and flexible APIs that can be easily imported and integrated into existing modern Mashup applications.

Web2ID relies on client-side components that run within browser and are able to securely communicate with each other. In this paper we use the term *mashlet*, which we introduced in [31], to refer to these components. We provide more details on mashlets in Section 2. In Web2ID, users are represented by a mashlet hosted at their identity URL, in much the same way that service providers are represented on the client-side by their mashlets. We call the mashlet that is hosted at the identity URL an *identity mashlet*. That is, in Web2ID, the identity page is a mashlet, (i.e., it includes JavaScript libraries required for communication) which provides authentication services.

During the authentication protocol, users first presents their identity credentials to their identity mashlet. In turn, the identity mashlet acts on behalf of the user and interacts with other mashlets to prove that the user owns the identity that corresponds to its URL. The iden-

tity mashlet enables other desirable features including authorization delegation and attribute exchange.

We have three main technical contributions in this paper.

1. We design and implement a new identity-management framework – Web2ID – for sensitive client-side mashup applications. Web2ID supports identity authentication on the Internet without any centralized trusted party. In addition, user's privacy (in terms of service history) is protected because identity providers and service providers do not communicate directly about the user's requests.
   We also describe how attribute exchange and the delegation of authorization are done in Web2ID.
2. We implement an in-browser public-key cryptosystem in JavaScript for Web2ID cryptographic operations can be completed solely in the browser. Our library is general-purpose and is useful beyond the specific identity management problem studied.
3. We generalize Web2ID to the identity management in Webtop applications, where a browser provides a desktop-like environment for office applications and data management. We describe our open-source Webtop application and how to integrate Web2ID with it. The source code for Webtop and cryptographic libraries are available on SourceForge ([34]).

In our Web2ID architecture, we define a new communication structure which we call *mashlet relay*. The mashlet relay is a mashlet that passes information between two mashlets belonging to different domains. Thus, it enables *indirect* cross-domain communication. For identity management, mashlet relay enables a service provider to send a query to another provider without revealing its identity. The mashlet-relay framework protects user privacy in mashup environments because service providers that host user data are unable to learn how users consume their data. This feature is especially important when users wants to provide their identity attributes (certified by a trusted party) to another service provider.

Our framework is implemented as a JavaScript library without browser modifications or specialized plugins to operate. It is fully portable across browsers and execution platforms. We illustrate the portability of our framework by incorporating it with several popular browsers, including Firefox, Opera, Apple Safari, IE and Google Chrome. Moreover, we avoid using HTTP redirections for communication; consequently, our protocol is compatible with modern Ajax-based Web applications.

Organization of the paper. The rest of the paper is organized as follows. We provide the background information on mashup-related concepts in Section 2. We give our threat model in Section 3. Our basic Web2ID protocol is in Section 4. Extensions of Web2ID are presented in 5 and 6. Our implementation is presented in Section 7. We apply Web2ID to the identity management problem in webtop applications in Section 8. Related work is described and compared in Section 9. We give the conclusions at the end of the paper.

2 Background on Mashups and Mashlets

Mashup applications aggregate content from a number of providers and display them within Web browsers. Such applications can be designed either as *server-side* mashups or *client-side* mashups. In server-side mashups, a proxy (called the mashup server) aggregates content from multiple sources. The Web browser loads the mashup application by visiting a URL corresponding to the proxy. In contrast, client-side mashups directly aggregate content within the Web browser. Several frameworks have recently been proposed to support

safe yet expressive client-side mashups. For example, OMOS [31] and SMash [17] are two frameworks that support secure inter-domain communication without requiring any change to the browser. MashupOS [28] is another framework that provides as set of powerful API for secure inter-domain communication but requires change in the browser. [4] analyses security of existing client-side communication protocols.

The client-side components of a mashup application are called *mashlets*. Mashlets represent the service provider that is hosting them in the client side and run in the browser with the privileges given to their hosts. To be concrete, a mashlet is simply a HTML page which loads to an iframe and contains some JavaScript code that enables it to communicate with other mashlets in the page. A *mashup application* is a Web application that aggregates a number of mashlets, possibly from different sources on the Web. We also use the term *mashlet container* to refer to the mashup application.

A secure inter-mashlet communication protocol is one that guarantees mutual authentication, data confidentiality, and message integrity. *Mutual authentication* in inter-mashlet communication means that two mashlets that communicate with each other must be able to verify each other's domain name. *Message integrity* requires that any attempt to tamper with the messages exchanged between two mashlets should be detected/prevented. *Data confidentiality* means a mashlet should not be able to listen to the communication between two other mashlets running under different domains.

For concreteness, the rest of this paper describes mashups and mashlets in the context of OpenMashupOS (OMOS) [22,31], a secure client-side mashup framework that we developed in prior work. However, the concepts developed in this paper are applicable to any client-side mashup framework that provides the above properties.

3 Our Threat Model

In Web2ID, there are several types of players: user, service provider, service consumer, attribute provider, and outside attacker (or stranger). Each player has a different degree of trustworthiness, as we explain next.

Users may be malicious. As is standard with AJAX-based applications, some messages of the Web2ID protocol are exchanged on the client-side, within the user's browser via inter-mashlet communication. Because the user has complete control over the browser, a malicious user may alter the client-side component of the Web2ID protocol, for example, by forging the identity of another user or providing forged identity attributes to an attribute requester. Consequently, for transactions in which the user must not be trusted, the correctness and integrity of the Web2ID protocol must not rely on the client-side portion of the protocol executing correctly. Web2ID uses cryptographic techniques to ensure the integrity of data that passes through the client.

Service providers may be malicious. When a service provider authenticates a user, it must receive certain information that enables it to ensure the authenticity of the user. A malicious service provider may misuse this information to impersonate the user to a second service provider using a *relay attack*. For example, a malicious service provider attacker.com that authenticates Alice may use her credentials to impersonate her to another service provider honest.com. In this attack, attacker.com tries to log into honest.com claiming the ownership of Alice's identity URL (e.g., alice.me). When honest.com challenges attacker.com, it relays that challenge to Alice when she tries to prove her identity to attacker.com. In turn, attacker.com uses this information to convince honest.com of Alice's identity.

Service consumers may be malicious. In the authorization delegation protocol, a malicious consumer may try to convince a service provider to give it access to a user's protected resources without possessing appropriate authorization (i.e., explicit consent from the user). In the case where users wishe to protect their privacy from the consumer, a malicious consumer may try to learn the user's identity during the course of authorization.

Attribute providers may be malicious. We refer to a service provider that requests user's attributes as an *attribute requester* and the service provider that stores user attributes and settings as an *attribute provider* (also called a wallet [24]). An attribute provider may optionally certify user attributes (e.g., for attribute-based authorization) or simply send non-certified values (e.g., for providing settings and preferences).

In an attribute exchange, a malicious attribute provider may try to violate a user's privacy by learning the identity of requesters that try to obtain the user's attributes. As a result, the attribute provider may learn the user's surfing habits. Similarly, a malicious attribute requester may also try to learn the identity or attributes of the user without user's agreement.

Man-in-the-Middle (MitM) attacks. Based on their capabilities, man-in-the-middle attackers (MitM) [15] can be either active or passive. A passive MitM attacker only listens to the conversation between two parties in the protocol. The goal of a passive attacker is to obtain information that can be used to impersonate the users, get unauthorized access to their private resources or violate their privacy. In contrast, an active attacker can also modify the content of conversation. An active MitM may try to change the result of an authentication or authorization check by modifying or replaying data transmitted in the protocol. MitMs can also be classified based upon their location in the network. Client-side MitMs involve a malicious mashlet that tries to spoof mashlet-to-mashlet communication in the protocol. A network MitM spoofs network communication, such as those between a mashlet and its server, or between two servers. In the Web2ID protocol, we assume that the point to point network communications are safe against active MitM attacks, which can be guaranteed by using secure lower level protocols like SSL. Finally, malicious mashlets may also try to subvert the protocol by launching frame phishing attacks against the user [17].

4 Basic Web2ID Protocol

The basic Web2ID protocol enables users to prove their identity to a service provider website without the use of a trusted third party. This protocol enables users to independently prove their identities and prevent any third party from learning their surfing habits. We achieve this goal using public-key cryptographic primitives in a manner akin to public-key client authentication in SSH (RFC 4252 [3]).

In Sections 5 and 6, we extend our basic Web2ID protocol to support attribute exchange and the delegation of authorization, respectively.

Suppose that a principal P (e.g., Alice) wishes to adopt an identity I (e.g., an identity URL, such as alice.me) and prove her ownership of that URL to a service provider SP.com. There are two main operations in the basic Web2ID protocol: *identity adoption* and *user authentication*, as described in the following.

Identity Adoption. To adopt an identity URL I, say alice.me, Alice first hosts an identity mashlet at this URL. The identity mashlet is a component that is trusted by Alice and represents her within a mashup application. To configure her identity mashlet, Alice must navigate to her identity mashlet using a browser. When the identity mashlet loads for the first time, it detects that it is not configured, and generates a public/private-key pair

($Pu(I)$,$Pr(I)$). The public key is embedded within the identity mashlet, while the private key must be stored safely by Alice.

User Authentication. When a user such as Alice attempts to authenticate herself with a service provider, she claims the ownership of an identity URL, such as alice.me. In turn, the service provider sends a session token encrypted under the public key associated with the identity URL alice.me. When the user then sends requests to access resources, she must prove ownership of the session token corresponding to her claimed identity. Figure 1 illustrates how service provider SP.com assigns a session token to the user who claims the ownership of identity alice.me. As Figure 1 illustrates, authentication happens in seven steps, as described below:

1. The user claims to own an identity I. For instance, this identity could be an identity URL alice.me. This claim can be communicated to the mashlet of the service provider SP.com. For example, the user may enter the URL in a form provided by SP.com.
2. The service provider's mashlet sends the claimed ID I to the service provider and, if not already loaded, loads the identity mashlet located at the claimed identity URL (i.e. alice.me).
3. The service provider extracts the public key $Pu(I)$ and the type/version of the corresponding public-key encryption algorithm Alg from the claimed identity page.
4. The service provider first generates a session token χ, and encrypts χ and the domain name of its mashlet SP.com with the public key $Pu(I)$. It then sends the result $\Delta = E_{Pu(I)}(\chi$, SP.com) back to the service provider's mashlet as response. Note that the domain name of the service provider must be included in Δ to protect users against relay attacks by malicious service provider (see Section 3).
5. The service provider's mashlet sends $\Delta = E_{Pu(I)}(\chi$, SP.com) to the identity mashlet for decryption.
6. If the identity mashlet does not already have the private key $Pr(I)$, it asks the user to provide her login credentials. Using the user's login credentials identity mashlet computes the private key. For example, the user can load the encrypted value of her private key from a USB memory stick and provide a passphrase that can be used by the mashlet to compute the private key. Alternatively, the user may enter the private key directly by swiping a smart card that contains her private key.
   Once the identity mashlet has the private key and user permits the authentication, the identity mashlet decrypts Δ and verifies that the domain name of the service provider (SP.com) matches the domain name in the token.
7. The identity mashlet sends the computed session token χ back to the service provider mashlet.

In our implementation, inter-mashlet communication is facilitated by the OMOS framework [31] which provides mutual authentication, and confidentiality and integrity guarantees for the data exchanged between two mashlets. *This ensures that a malicious mashlet in the mashup application will not be able to compromise communication between the identity mashlet and the service provider's mashlet* (so the value χ will not be available to an eavesdropper).

Upon the completion of the above protocol, the service provider can verify that the value of the session token received from the identity I is valid. This proves the user's claim of ownership of I to SP.com. Existing identity management protocols prove the possession of the session token by including it with each request, and are therefore vulnerable to session hijacking via MitM attacks. Our implementation of Web2ID uses a MAC (message authen-

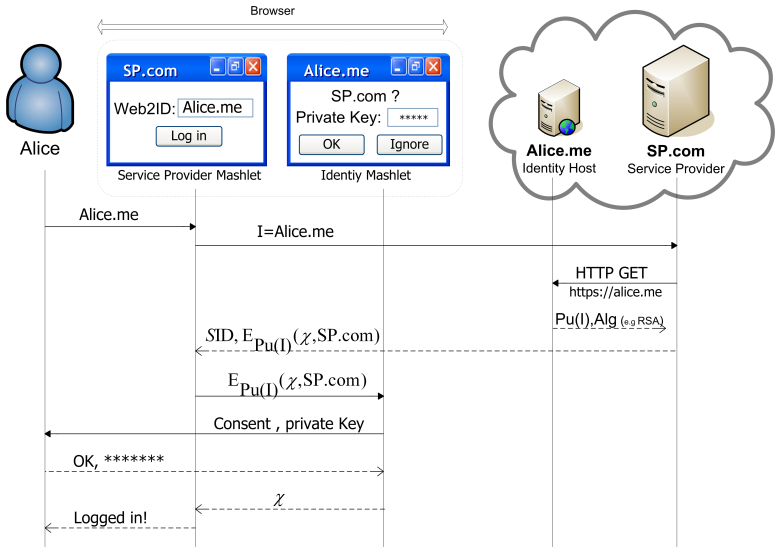

Fig. 1 An identity mashlet represents the user within the application. The user can prove ownership of the identity mashlet by proving the possession of the private key that corresponds to the public key located at URL of the identity mashlet.

tication code) to prove possession of the session token.[1] In this approach, the MAC value of each `XMLHttpRequest` request is computed using the session token and is included in every request. The service provider serves a request only if the included MAC value is correct. Note that during the above protocol does not require the service provider to keep any protocol-specific state, thereby ensuring a stateless implementation of the web application at the service provider. In addition, the user's credentials are never transmitted over the network; instead such communication happens on the client-side, where communication is secured using OMOS.

The Web2ID authentication protocol can also be used by a service provider to prove the ownership of its mashlet. We use this feature as part of authorization delegation protocol that we describe next. The authorization delegation and attribute exchange protocols build upon the authentication protocol described above.

The above basic Web2ID protocol supports user authentication. It can be generalized to support more complex operations such as identity attribute exchange and authorization delegation. In Section 5, we will present a mashup relay framework and explain how it facilitates attribute exchange in Web2ID. Our authorization delegation protocol is described in Section 6.

Security Analysis of the User Authentication Protocol. Because Web2ID uses client-side inter-mashlet communication, its security relies on the client-side communication protocol that is used in its implementation. We assume that the mashlet framework that is used for implementation of Web2ID guarantees confidentiality of inter-mashlet communication. This assumption implies that the mashlet framework protects the protocol against MitM attacks

[1] To do so, we ported the necessary cryptographic functions HMAC-SHA1 and HMAC-SHA256 (RFC2104 [18], RFC3174 [9]) into the OMOS framework.

by malicious mashlets. Next, we analyze how the user authentication protocol resists against attacks launch by adversaries.

Since the session token χ is encrypted by the public key that is associated with the claimed identity URL (located at the identity page), the user can get access to the session token only if she owns the corresponding private key. Therefore, assuming that only the owner of an identity URL has access to the private key that corresponds to the public key embedded in the corresponding identity page, she will be the only person that can use that session token. This prevents malicious users from forging identities that does not belong to them.

To protect users against relay attacks, Web2ID requires service providers to encrypt the domain name of their mashlet besides the session token. This way the identity mashlet can ensure that the mashlet that is requesting the session token is not relaying an encrypted session token issued by another service provider. Finally, since user's credentials and session tokens are never sent over network in clear text, Web2ID authentication is immune to passive MitM attacks. As discussed earlier, Web2ID relies on the underlying network protocols (e.g., https) to protect integrity of point to point communication against active MitM attackers. However, to prevent active MitM attackers from replaying a successful authentication transaction, service providers need to record tokens and reject transactions which contain a token which is already used. Number of tokens that need to be saved can be reduced by introducing a timestamp into each token and rejecting tokens that are older than a threshold.

5 Attribute Exchange and Relay Mashlet in Web2ID

An important feature supported by most identity management frameworks is that of *attribute exchange*, in which one service provider requests a user's identity attributes (e.g. age) or preferences from another service provider. Attribute exchange protocol allows sensitive applications to bind an identity URL with a physical entity (person) by querying attributes of its owner certified by a trusted third party. For example, SP.com can query certified address of alice.me from a trusted third-party (e.g. Division of Motor Vehicle Bureau or dmv.org) to ensure that URL belongs to the same physical entity that claims its ownership. Attribute exchange is especially important for mashup applications, in which interaction between mashlets is the norm and user may have multiple identities for each service provider as attribute exchange can be used to correlate these identities.

In this section, we introduce a new mashlet-relay structure that enables user-centric client-side communications between two domains. Then, we explain why such a mashlet relay framework is useful in the implementation of identity attribute exchange in Web2ID. The main purpose of mashlet relay is for better user privacy, as it provides an indirect communication channel between two service providers. The providers do not directly interact with each other with regard to a user's request, and thus cannot share information of a user.

5.1 Mashlet-Relay Framework As a Building Block For Privacy-Aware Client-Side Mashups

We define *mashlet-relay framework* as a special client-side mashup framework with three mashlets within a browser environment where the communication of two mashlets, each hosting contents of a remote server, is indirect and realized through a third mashlet that is hosted by the local host. We refer to the two mashlets hosted by remote servers as *server*

mashlets. A server mashlet also communicates to its corresponding remote server via the mashlet-to-server communication mechanism. We refer to the mashlet that bridges the communication of the two server mashlets as the *relay mashlet*. All inter-mashlet communication follows the mashlet-to-mashlet messaging mechanism. The relay mashlet effectively passes messages between two server mashlets and is able to modify the messages based on user's inputs. Figure 2 shows a schematic drawing of such a mashlet relay framework, where the mashlet in the middle (Mediator) mediates the communication between a requester (e.g., SP.com) and a provider (e.g., AttProvider.com). The mediator mashlet is launched by the local host of the individual user. It anonymizes the identity of the requester (e.g., SP.com), as the provider (e.g., AttProvider.com) learns nothing about who issues the request. Such a mashlet relay framework, although simple, supports a user-centric design where the user is able to monitor and actively control the messages being communicated among server mashlets. As a consequence, the client-side relay mashlet eliminates the need of direct communication between the two server mashlets. This feature plays a key role in enabling privacy-aware identity management in Web2ID.

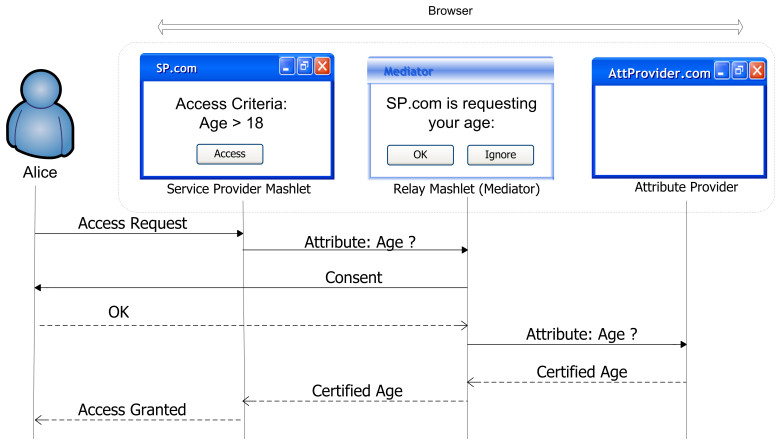

Fig. 2 Web2ID users mashlet relay communication framework for attribute exchange. In mashlet relay framework, a mashlet (center) mediates the communication between requester (left) and provider (right) and anonymizes the identity of requester.

This mashup-based relay framework naturally facilitates the construction of a privacy-aware identity management protocol, namely identity attribute exchange in SSO, that enables the exchange of user's identity credentials without the direct communication between the identity provider and service provider. In existing (federated) identity management systems, *direct communications* between providers on user's ID information are typically required, which, however, is undesirable as providers may learn sensitive attributes of the user. Therefore, the segregation of providers in their communication protects user privacy and prevents providers from colluding to discover user activities. Yet, in the meantime, proper message exchanges among providers should be allowed, e.g., a service provider may need to verify Alice's identity attributes hosted by an identity provider. Next, we explain why such a mashlet relay framework is useful in the identity attribute exchange in Web2ID.

5.2 Identity Attribute Exchange Based on Mashlet Relay

When a service provider requests a user's identity attributes from another service provider, the user may wish to anonymize the identity of the provider requesting these attributes. Doing so prevents the attribute providing service from learning the user's surfing habits. To implement privacy-aware identity attribute exchange, Web2ID avails of the mashup relay framework. In particular, the relay mashlet mediates the exchange of identity attributes between service providers. Because the relay mashlet forwards the request to the attribute provider only after obtaining the user's consent, users have full control over what attributes can be exchanged.

Figure 2 presents an example that shows how using Web2ID a service provider SP.com can query user's age certified by AttProvider.com. If the attribute requester already knows the user's identity, the identity mashlet of the user can itself be used as a relay mashlet. Alternatively, a mashlet loaded from a trusted third party or the local machine can act as the relay mashlet. We omit the security definition and analysis for our identity attribute exchange protocol, as they can be easily deduced following the analysis in the basic Web2ID protocol.

6 Web2ID Extension: Realizing Authorization Delegation

Web application mashlets included in a mashup typically access resources hosted at other domains. In this context, the mashlet that accesses resources is typically called the *Consumer*, while the domain that hosts the resource is called the *Service Provider*. Consumers should not be able to access a user's protected resources unless the user grants them the required access permission.

An *authorization delegation protocol* allows the users to delegate permissions to a consumer to access their resources hosted at a service provider. For example, users may be able to delegate permissions needed to access their files on a photo-sharing website (the service provider) to a website that provides photo editing utilities (the consumer). An authorization delegation protocol should be privacy-preserving in that it must not reveal the user's identity. For instance, users may wish to grant the photo editing service read access to their photos hosted on the photo sharing website without revealing their identity to the photo editing service.

Users may wish to delegate to a consumer the rights to access their resources hosted on a service provider. There are two cases that arise in the implementation of authorization delegation, based upon the privacy guarantees that the user requires.

Case 1: Protecting user identity from the consumer. In the first case, users may not want to disclose their identity to the consumer. For example, a user Alice may wish to print her photos hosted at a photo sharing website SP.com by allowing a printing website Consumer.com to access her photos at SP.com. Yet, she may not wish disclose her identity (i.e. Alice.me) to Consumer.com. To support this case, the authorization delegation protocol should not give any information to the consumer that reveals her identity.

Figure 3 illustrates the authorization delegation protocol, via which Consumer.com acquires an opaque token AC to access Alice's resource (e.g., /a/v.jpg) without learning her identity I (e.g., Alice.me). As this figure illustrates, the service provider SP.com uses a secret key SK, known only to the service provider, to generate an opaque token AC = E_{SK}(Consumer.com, GET, /a/v.jpg, I) that grants Consumer.com read access (i.e., a GET request) to the resource /a/v.jpg, which belongs to I.

When the service provider SP.com receives a request from Consumer.com (via back-end server-to-server communication) containing the access token AC, it first decrypts AC and ensures that that the identity of the requester matches the principal that the token is granted to (Consumer.com); if so, it allows the request.

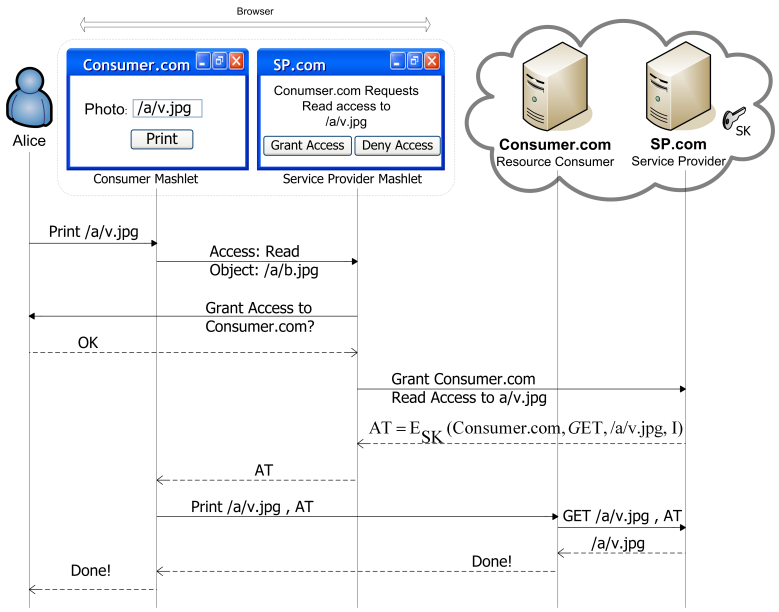

Fig. 3 In Web2ID, a service provider can issue an opaque token to a consumer to access user's resources. In doing so, Web2ID does not reveal the user's identity to the consumer.

Case 2: User identity known to consumer In this case, the consumer already knows the user's identity (e.g., because the user has authenticated to the consumer). Figure 4 illustrates the protocol used in this case. The identity mashlet of users can independently issue an access delegation certificate using their private keys to grant the consumer access to their protected resources hosted on a service provider. In turn, the service provider can validate the certificate using the user's public key. The service provider can obtain the public key using the identity URL of the user that the resource belongs to.

Our Web2ID authorization-delegation protocol does not require the consumer to pre-register with the service provider. This property is in sharp contrast to similar protocols (such as OAuth), which require the consumer to pre-register with the service provider. Additionally, Web2ID does not require the service provider or the consumer to maintain protocol-related state during delegation, therefore it is scalable and easy to implement.

Security Analysis. Before serving a request, service providers verify that the access tokens are either issued using their own secret keys or the private key of the owner of the resource. Since these types of tokens can be issued only with user's consent, consumers will not be able to access users resources without agreement of their owner. To prevent MitMs from using h

ijacked access tokens, Web2ID requires that all access tokens be bound to the domain name of the mashlet that the token is granted to. Therefore, these tokens can be used only by

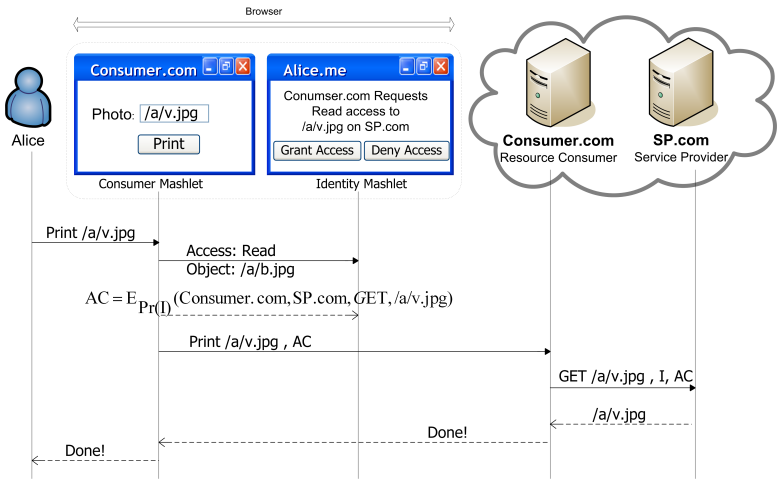

Fig. 4 The identity mashlet issues a delegation certificate for read access to resource /a/v.jpg. Using this certificate the consumer can access /a/v.jpg on SP.com.

the service provider that owns the mashlet. Service providers can use Web2ID authentication to prove ownership of the mashlet that the token is issued for. In the access tokens issued by the service provider, the identity URL of the user is encrypted by service provider's secret key. Therefore, the consumer will not be able to learn the identity of user and this protects the privacy of the user.

7 Implementation and Evaluation

Realizing Web2ID requires in-browser symmetric and public-key cryptographic primitives. However, there is no JavaScript cryptographic libraries that provide all the operations that are required for implementation of Web2ID (i.e., HMAC, public-key encryption and public/private key generation). The only JavaScript-based library that implements public-key cryptography [27] does not support public/private key generation, which is required by Web2ID.

As one of the main technical contributions of this paper, we developed a JavaScript-based cryptographic library that not only supports operations that are required by Web2ID but also can be easily extended to support other cryptographic operations. Our library is fully compatible with commodity browsers, such as IE, Firefox, Chrome, Opera and Safari, and does not require any browser modifications.

7.1 Implementation Details

Since development of a JavaScript library from scratch is very time-consuming and error-prone, we based our implementation of the JavaScript cryptographic library on the Java Cryptography Architecture (JCA) [16, 21], an open-source Java-based cryptographic toolkit. We used Google Web Toolkit (GWT) to translate code from Java to JavaScript. However, in implementing this library and porting it to commodity browser platforms, we encountered

three technical challenges, namely *performance*, *browser interference*, and *code complexity*, that we describe below.

Performance. Directly compiling the JCA library into JavaScript resulted in extremely poor performance of cryptographic operations. We found that the main performance bottlenecks were BigInteger operations, such as modInverse, mod, and multiplication operations, that are frequently used in cryptographic operations. We addressed this problem by replacing the JCA implementation of BigInteger with the native JavaScript code using the JavaScript Native Interface (JSNI). This replacement significantly improved the performance, with encryption and decryption operations consuming less than a second (see also Section 7.2).

Browser Interference. The implementation of the Web2ID protocol requires generation of public/private key pairs when the identity mashlet is first loaded. We observed that key generation algorithms for public-key cryptographic algorithms such as RSA were quite expensive. Because most browsers (and JavaScript interpreters) are single-threaded, users cannot interact with the browser during key generation. Most browsers time out JavaScript functions that execute for long durations of time (typically about 10 seconds). As a result, key generation algorithms are interrupted by the browser.

To overcome browser interference during our key generation operations and keep the browser responsive, we used an *incremental and deferred computation* technique. We observed that the most expensive operation during the generation of public/private RSA key pairs was the generation of probable prime numbers p and q. The BigInteger.getProbablePrime function continuously generates random odd integers until it finds one that passes Miller-Rabin primality test, thereby resulting in long execution times. We changed this procedure so that each iteration runs in a continuous time slice. We then scheduled the next iteration for another time slice and returned control to the browser, as illustrated in Figure 5. This process continues until the key generation algorithm finds a number that passes Miller-Rabin test. We found that this approach was effective at keeping the browser responsive and preventing browser timeouts of JavaScript execution.

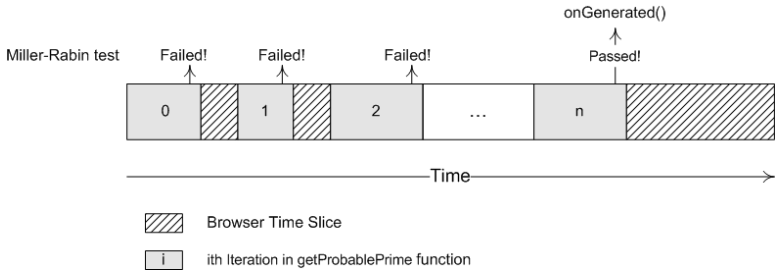

Fig. 5 Deferred execution of prime number generation.

Code complexity. JCA, upon which our JavaScript library is based, uses several Java features, such as reflection, that are not supported by GWT. Consequently, we first modified JCA to a set of core components that were sufficient to implement cryptographic operations needed for Web2ID. We then used this stripped-down version of JCA with GWT to produce our JavaScript library.

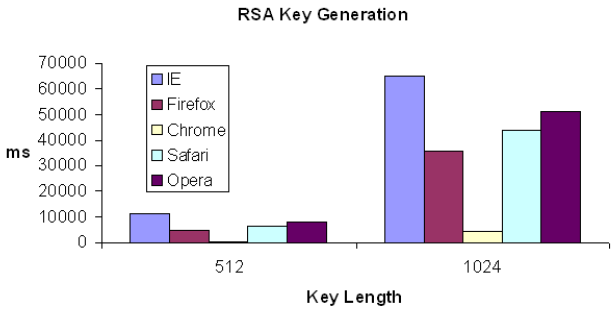

Fig. 6 Key generation performance in milliseconds of our cryptographic library on different browsers.

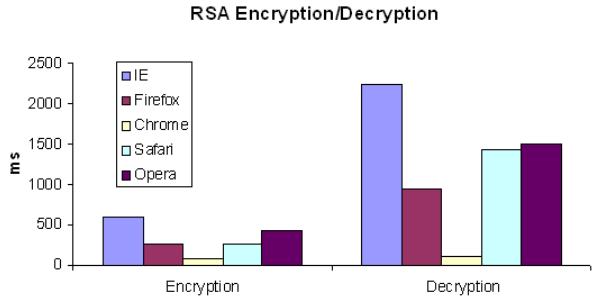

Fig. 7 The performance in milliseconds of RSA encryption and decryption on different browsers.

7.2 **Experimental Results**

Our goal is to study feasibility and overhead of using in-browser cryptographic operations. We ran experiments on a machine with the following configuration: Intel Core 2 CPU, 980 MHz, 1.99 GB RAM, Microsoft Windows XP 2002 SP2. We tested our implementation using the following browsers: Google chrome v1.0.154.53 Firefox v3.0.8, Internet Explorer v7.0.5730.13, Opera v9.27, and Apple Safari v3.1.1. The most expensive cryptographic operation that is required by Web2ID is key generation. Figure 6 shows the runtime of our RSA keypair generation function for keys of size 512 and 1024 bits. Since key generation is a probabilistic process, the values reported are averaged results over ten runs. As this Figure shows, Google Chrome, which uses a fast JavaScript Engine (V8), generates a 1024-bit key pair in under 4 seconds. The slowest browser was IE, which took about one minute to generate a 1024-bit key pair. Because key generation is a one-time operation and the browser stays responsive during this time, we feel that this delay is acceptable.

Figure 7 shows the performance of RSA encryption/decryption using keys of length 1024 bits. As expected, decryption is more costly compared to encryption and the performance is quite reasonable for web applications. Of the browsers that we tested, Google Chrome had the best performance (less than 100ms for decryption using 1024-bit key).

8 Applying Web2ID to Web-based Desktop Applications

In this section, we present an application of the Web2ID protocol to a web-based desktop application (in short, a *Webtop*). Web-based desktop applications (or webtops) provide a desktop-like environment within the browser. Users can open multiple office applications within a webtop, and can easily share data between these applications (e.g., using drag-and-drop). A number of popular Webtops are now available, including Glide OS, eyeOS, and G.ho.st [12, 10, 11].

To demonstrate the application of Web2ID to Webtops, we build a Webtop application called *Zaranux* [33]. To the best of our knowledge, this application is the first linux-based open-source Webtop. Zaranux emulates a desktop environment, such as Gnome or KDE, within the Web browser. It provides several applications, including a command-line interface (i.e., a terminal), via which users can easily browse and access their remote file system, upload/download files, and run third-party applications. Each third-party application, such as a word processor, runs within its own protection domain and can access user data in a controlled manner after obtain the user's consent.

Because Webtops support a variety of office applications, typically from different sources, users often have to authenticate themselves with each such application. A Webtop that integrates an identity management solution can therefore greatly improve end-user experience. However, existing identity management solutions are not directly applicable to Webtop environments due to the heavy use of redirection and privacy concerns. Office applications are typically stateful and contain unsaved data. The identity management solutions implemented via a series of HTTP redirections would result in the loss of unsaved data.

We therefore integrated our implementation of Web2ID with Zaranux. Below, we discuss how Web2ID provides single sign-on, authorization delegation and resource access in Zaranux.

User Authentication and Single Sign-On. A user first logs into Zaranux and enters his credentials into a mashlet provided by an identity provider (as discussed in Section 4). The implementation of Web2ID in Zaranux ensures that any applications that require authentication can seamlessly verify the identity of the user without requiring the user to authenticate again. Zaranux shares the identity of its users with applications only after getting their consent. The example below explains a common authentication scenario.

Suppose that Alice has logged into Zaranux, and has started a financial application, e.g., located at the URL http://investment.com. When Alice tries to access her data at investment.com, it must authenticate her. To do so, the mashlet from investment.com requests Alice's identity URL by making a client-side system call to Zaranux. After getting Alice's consent, Zaranux returns her identity to investment.com. In turn, according to Web2ID protocol, to verify this claim, investment.com mashlet forwards the claimed URL to the investment.com server, which retrieves the public key from Alice's identity URL, encrypts a session token and returns it to the client side (as in the Web2ID protocol). Note that all these steps are transparent to Alice, once she has authenticated herself to Zaranux, which in turn provides identity management services to other office applications that require authentication.

Authorization Delegation and Resource Access. In Zaranux, an office application that wishes to access a resource, such as a file or directory, invokes a client-side API akin to the `open` system call on traditional desktop operating systems. This API call returns a file handle that can be used to access the resource. This file handle serves as an opaque capability token that delegates a certain access permission (e.g., read, write or delete) to the

token holder. Zaranux also implements the authorization delegation protocol (discussed in Section 6), and uses relays to enable delegation in a privacy-preserving manner.

9 Related work

OpenID implements decentralized user authentication on the Internet via a series of HTTP redirections within the user's browser. These redirections perform inter-domain communication between the IdP and SP and transmit the user's credentials from the IdP to the SP. However, redirections are ill-suited for stateful Ajax-based applications, such as Web desktops and Web-based office applications, because they involve unloading/reloading the application upon each redirection. Without application-level support, unloading/reloading operations will result in the loss of unsaved data. In addition, the use of an identity provider to manage credentials and personal information raises privacy concerns. Web2ID provides technical solutions for both problems.

Our Web2ID protocol can be realized with any secure mashup frameworks. They provide general infrastructure and environments for content providers to communicate in our identity management applications. There have been a couple of recent work that proposed secure mashup solutions including MashupOS [28], SMash [17], PostMessage method [4], and OMOS [31]. The main goal of these solutions is two-fold: to isolate contents from different sources in sandbox structures such as frames and to achieve frame-frame communication.

SMash [17] uses the concepts in publish-subscribe systems and creates an efficient event hub abstraction that allows the mashup integrator to securely coordinate and manage contents and information sharing from multiple domains. SMash mashup integrator (i.e., the event hub) is assumed to be trusted by all the web services. MashupOS [28] applies concepts in operating systems in mashup and develops sophisticated browser extensions and environments that enable the separation and communication of frames similar to inter-process communication management in the operating system. As mentioned earlier, the OpenMashupOS (OMOS) framework contains a key-based protocol providing secure frame-to-frame communication [31].

Despite the recent progress on mashup applications, the identity management in mashup environments has not been systematically investigated in the literature. Camenisch *et al.* presented the architecture of PRIME (Privacy and Identity Management for Europe), which implements a technical framework for processing personal data [7]. PRIME focuses on enabling users to actively manage and control the release of their private information. Privacy policies for liberty single sign-on [8, 19] have been presented [23] by Pfitzmann. The paper identifies a number of privacy ambiguities in Liberty V1.0 specifications [20] and propose privacy policies for resolving them. A good article on the issues and guidelines for user privacy in identity management systems was written by Hansen, Schwartz, and Cooper [14].

In the federated identity management (FIM) solution by Bhargav-Spantzel *et al.*, personal data such as a social security number is never transmitted in cleartext to help prevent identity theft [5]. Commitment schemes and zero-knowledge proofs are used to commit data and prove the knowledge of the data. BBAE is the federated identity-management protocol proposed by Pfitzmann and Waidner [25]. They gave a concrete browser-based single sign-on protocol that aims at the security of communications and the privacy of user's attributes. Goodrich *et al.* proposed a notarized FIM protocol that uses a trusted third-party, called notary server, to effectively eliminate the direct communication between identity provider and

service provider [13]. The main difference with these proposed privacy-aware ID management solutions and our approach is that we study ID management in the client-side mashup environment through a novel and efficient mashlet relay framework.

In the access control area, the closest work to ours is the framework for regulating service access and release of private information in web-services by Bonatti and Samarati [6]. They study the information disclosure using a language and policy approach. We designed cryptographic solutions to control and manage information exchange. Another related work aiming to protect user privacy in web-services is the point-based trust management model [30], which is a quantitative authorization model. Point-based authorization allows a consumer to optimize privacy loss by choosing a subset of attributes to disclose based on personal privacy preferences. The above two models mainly focus on the client-server model, whereas our architecture include two different types of providers.

10 Conclusions

As mashup applications increase in popularity, we expect that they will also be used with sensitive Web services, such as financial and banking applications. When mashups are used in such scenarios, it is important to provide features such as identity management. We presented Web2ID, an identity management protocol for mashup applications. Web2ID preserves the privacy of the end user and eliminates the need for a trusted identity provider in the online single sign-on process. We described how this feature can be realized with conventional public-key cryptography. We also described a mashlet-relay framework that enables efficient yet indirect communication between two server mashlets via a local relay mashlet controlled by the user. Such a relay framework allows for attribute exchange without disclosing the user's surfing habits to service providers. Our implementation of Web2ID and the relay framework is implemented as an in-browser library and is fully compatible with commodity browsers.

We overcome several technical difficulties and successfully implemented a public-key cryptographic JavaScript library for the browser to perform cryptographic operations such as key-pair generation, encryption, and decryption. This technical contribution is beyond the specific identity management problem studied. Last but not the least, we also described how Web2ID applies to the emergent Webtop environments.

11 Acknowledgements

This work has been supported in part by NSF grant CAREER CNS-0831186. This material is based upon work supported by the U.S. Department of Homeland Security under grant number 2008-ST-104-000016. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

The first author would like to thank the help of professors at Bahai Institute for Higher Education (BIHE).

References

1. OpenID Specification. http://openid.net/developers/specs/.
2. Philip Zimmermann, http://www.philzimmermann.com.

3. RFC 4252, The Secure Shell (SSH) Authentication Protocol http://tools.ietf.org/html/rfc4252.
4. Adam Barth, Collin Jackson, and John C. Mitchell. Securing Browser Frame Communication. In *Proceedings of the 17th USENIX Security Symposium*, 2008.
5. Abhilasha Bhargav-Spantzel, Anna Cinzia Squicciarini, and Elisa Bertino. Establishing and Protecting Digital Identity in Federation Systems. *Journal of Computer Security*, 14(3):269–300, 2006.
6. Piero A. Bonatti and Pierangela Samarati. A Uniform Framework for Regulating Service Access and Information Release on the Web. *Journal of Computer Security*, 10(3):241–272, 2002.
7. Jan Camenisch, Abhi Shelat, Dieter Sommer, Simone Fischer-Hübner, Marit Hansen, Henry Krasemann, G. Lacoste, Ronald Leenes, and Jimmy Tseng. Privacy and Identity Management for Everyone. In *Proceedings of the 2005 ACM Workshop on Digital Identity Management*, pages 20–27, November 2005.
8. S. Cantor, F. Hirsch, J. Kemp, R. Philpott, E. Maler, J. Hughes, J. Hodges, P. Mishra, and J. Moreh. Security Assertion Markup Language (SAML) V2.0. Version 2.0. OASIS Standards.
9. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). In *RFC3147*.
10. eyeOS, http://eyeos.org/.
11. G.ho.st, http://g.ho.st/.
12. Glide OS, http://www.glidedigital.com.
13. Michael T. Goodrich, Roberto Tamassia, and Danfeng (Daphne) Yao. Notarized federated ID management and authentication. *Journal of Computer Security*, 16(4):399–418, 2008.
14. Marit Hansen, Ari Schwartz, and Alissa Cooper. Privacy and Identity Management. *IEEE Security and Privacy*, 6(2):38–45, 2008.
15. Hyunuk Hwang, Gyeok Jung, Kiwook Sohn, and Sangseo Park. A study on mitm (man in the middle) vulnerability in wireless network using 802.1x and eap. In *ICISS '08: Proceedings of the 2008 International Conference on Information Science and Security*, pages 164–170, Washington, DC, USA, 2008. IEEE Computer Society.
16. Java Cryptography Architecture. http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html.
17. Frederik De Keukelaere, Sumeer Bhola, Michael Steiner, Suresh Chari, and Sachiko Yoshihama. SMash: Secure Component Model for Cross-Domain Mashups on Unmodified Browsers. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.
18. Krawczyk, Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. In *RFC2104*.
19. Liberty Alliance Project. http://www.projectliberty.org.
20. July 2002. Liberty Alliance Project: Liberty Protocols and Schemas Specification, Version 1.0.
21. OpenJDK, http://openjdk.java.net/.
22. OpenMashupOS, http://omos.zaranux.com/.
23. Birgit Pfitzmann. Privacy in Enterprise Identity Federation - Policies for Liberty Single Signon. In *Proceedings of the Third International Workshop on Privacy Enhancing Technologies (PET 2003)*, volume 2760, pages 189–204, 2003.
24. Birgit Pfitzmann and Michael Waidner. Privacy in browser-based attribute exchange. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 52–62. ACM, 2002.
25. Birgit Pfitzmann and Michael Waidner. Federated Identity-Management Protocols. In *Security Protocols Workshop*, pages 153–174, 2003.
26. R. Leenes, J. Schallabck, and M. Hansen. Privacy and identity management for europe. Prime whitepaper, 15 May 2008.
27. RSA JS library. http://www-cs-students.stanford.edu/ tjw/jsbn/.
28. Helen J. Wang, Xiaofeng Fan, Jon Howell, and Collin Jackson. Protection and Communication Abstractions for Web Browsers in MashupOS. In *ACM Symposium on Operating Systems Principle (SOSP)*, pages 1–16. ACM Press, 2007.
29. A. Whitten and J.D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *8th Usenix security symposium*, pages 169–184, 1999.
30. Danfeng Yao, Keith B. Frikken, Mikhail J. Atallah, and Roberto Tamassia. Point-Based Trust: Define How Much Privacy Is Worth. In *Proc. Int. Conf. on Information and Communications Security (ICICS)*, volume 4307 of *LNCS*, pages 190–209. Springer, 2006.
31. Saman Zarandioon, Danfeng Yao, and Vinod Ganapathy. OMOS: A Framework for Secure Communication in Mashup Applications. In *ACSAC'08: Proceedings of the 24th Annual Computer Security Applications Conference*, December 2008.
32. Saman Zarandioon, Danfeng Yao, and Vinod Ganapathy. Privacy-aware identity management for client-side mashup applications. In *DIM '09: Proceedings of the 5th ACM workshop on Digital identity management*, pages 21–30, New York, NY, USA, 2009. ACM.
33. Zaranux, http://zaranux.com/, Saman Zarandioon.
34. Zaranux Open Source Project, http://zaranux.sourceforge.net/.