Towards Privacy-enhancing Identity Management in Mashup-providing Platforms

Jan Zibuschka, Matthias Herbert, and Heiko Roßnagel

Fraunhofer Institute for Industrial Engineering (IAO),
Nobelstr.12, 70569 Stuttgart, Germany
(first_name.last_name)@iao.fraunhofer.de

Abstract. Mashups empower users to easily combine and connect resources from independent Web-based sources and domains. However, these characteristics also introduce new and amplify existing security and privacy problems. This is especially critical in the emerging field of enterprise Mashups. Despite several contributions in the field of Mashup security the issue of protecting exchanged resources against the Mashup-providing Platform has generally been neglected. In this contribution we address the security challenges of server-side Mashup-providing Platforms with the aim of minimizing the required amount of trust. We achieve this by implementing a privacy-enhancing identity management system into the Mashup-providing Platform using Reverse Identity Based Encryption.

1 Introduction

Mashups empower developers to combine and connect resources from independent Web-based sources and domains. Mashups are dynamic and easy to create [1]. This combination of resources provides a considerable added value to both providers and end users, who can now manage a host of services inside one consistent environment. On the other hand, these characteristics also introduce new and amplify existing and arising security and privacy problems. These security shortcomings are especially relevant as Mashups are now moving into the enterprise space, where enterprise Mashups offer the compelling perspective of empowering business end users to compose business services as they are needed, allowing for a direct adaptation to changing business needs [2]. There have been several contributions in the field of Mashup security, such as De Keukelaere et al's SMash [3], Jackson and Wang's Subspace [4], Crites et al's OMash [5], or Zaradioon et al.'s OMOS [6]. However, the issue of protecting exchanged resources against the Mashup-providing Platform has generally been neglected. Providing the necessary privacy is a key issue for the broad success of enterprise Mashups [7], as especially smaller enterprises will not want to deploy their own Mashup infrastructure, and prefer to consume hosted services [8], allowing them minimum investment and maximum benefit from Mashups' ease of use. In this contribution we address the security challenges of server-side Mashup-providing

Platforms. In the spirit of multilateral security we aim at minimizing the required amount of trust of (enterprise and end) users in the platform. To achieve this we implement a privacy-enhancing identity management system as described in [9] into the Mashup-providing Platform. The further structure of this paper is as follows: First we present a generic architecture of a Mashup-providing Platform in section 2. We use this architecture to illustrate the security challenges in section 3 and provide an overview and discussion of related work in section 4. We present our approach to address the security challenges in section 5. We wrap up by discussing merits and limitations of our solution (section 6), before we conclude our findings.

2 Mashup Architecture

This section describes general assumptions we make about Mashups, including terminology, structure and API. We aim to give clear definitions based on earlier work, to ensure applicability of our results and readability of this work. Figure 1 (taken from [2]) gives an overview of the platform we are envisioning.

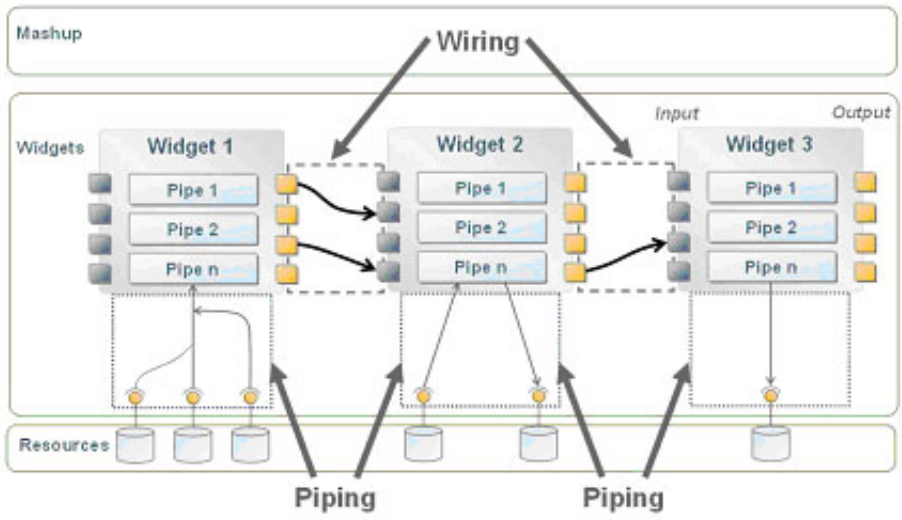

Fig. 1. Overview of Mashup-providing Platform [2]

2.1 Definitions

As Hoyer and Fischer [2] point out in the course of a comprehensive literature survey, there are various definitions of the term 'Mashup'. There seems to be a strong tendency of authors, even in scientific literature, to establish their own

definition of Mashups without referring to earlier work. To make our work more readily comparable (and readable), we offer clear definitions based on earlier work of the terms we are using in this section.

Mashup We slightly adapt the definition from [2] here, removing the enterprise aspect, as we think the definition is quite precise and convincing also outside of the enterprise setting, and in line with definitions used in other works, e.g. [3]. We define Mashup as: a Web-based resource that combines existing resources, be it content, data or application functionality, from more than one resource by empowering the actual end-users to create and adapt individual information centric and situational applications.

Widgets and Backend service Still drawing on the definitions in [2], as well as [10] and others, we define Widgets as the visual representations of aforementioned Web-based resources that are combined into a Mashup. We refer to backend services when referring to the Web-based services piping those resources into the Mashup-providing Platform and to the Widgets.

Mashup-providing Platform (MPP) We define a Mashup-providing Platform as a Web-based server-side platform offering APIs and hosting functionalities that enable the creation of Mashups, specifically offering functionality for definition of Widgets and their Wiring as described in [2], and APIs such as the one described in [3].

Pipes and Wiring We define Wiring as communication between Widgets within the platform, and Piping as the transfer of external resources into the platform via backend services and Widgets (Figure 1, [2]). Securing the Wiring-based communications (as discussed in [3]) is of special interest to us. We also address issues arising in the context of (cascaded) piping, as also described by [11].

Adapter (Backend) Services We define an Adapter (Backend) Service as a Backend Service who does not have direct access to the resources it pipes into the Mashup. Consequently, it needs to cascade requests to other services which do have the necessary resources (but which usually do not integrate with the Mashup-providing Platform, making the adapter necessary). We are especially interested in the case where those services holding the resources are following a walled garden security model, requiring delegation of credentials to the adapter (as described by [11]).

2.2 API and Relevant Implementation Details

There are many Mashup-providing Platforms in the market, with many different approaches to communication between backend services. In this paper, we

assume that the MPP implements an API and component model similar to the one presented in [3], using communication channels that respect the services home domain and thus do not break the browser security model, but still are embedded in and communicate via the Mashup-providing Platform using layered iFrames (see Figure 2). We deviate from the API presented in [3] in that we do

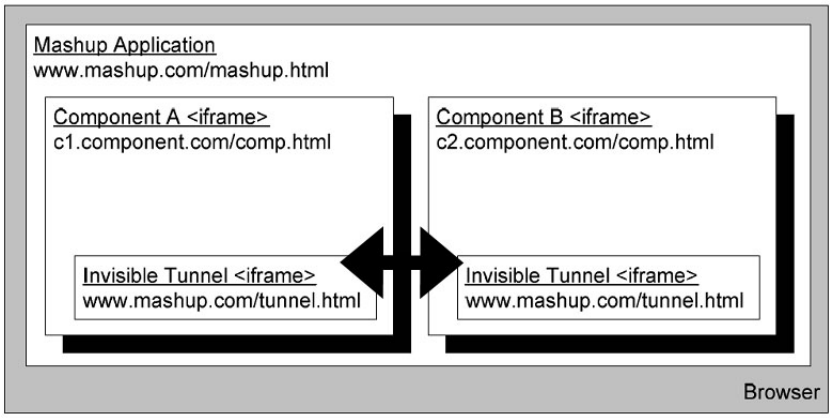

Fig. 2. Mashup Component Model (adapted from [3], now using direct tunneling)

not implement communication busses shared between different services, but use the Wiring model illustrated in Figure 1. This is a choice made for the sake of simplicity. As we use strong cryptography, shared busses would not be an issue from the security standpoint. However, they introduce an unnecessary level of complexity into the description of the system presented here. For simplicity, we assume a simple but flexible API, where the Wiring is managed using the API calls `requestValues()`, `requestPermits()` and `sendMessage()`, as illustrated by Figure 3. Table 1 gives an overview of the API we assume the MPP implements. This is an abstraction of APIs given in related work such as Hasan et al [12] and surveys of current Mashup systems [1][2]. It has also been verified by checking compliance with current approaches, such as the Enterprise Mashup Markup Language EMML [13] or JackBe Mashup Composer [14]. We also assume that the Piping is used mainly to integrate legacy services, accessed over HTTP, using passwords for authentication. However, we also demonstrate how access control to those backend services could be secured using mechanisms we describe in the context of the (more integrated) Wiring for Mashup Widgets.

3 Security Challenges of Mashup-Platforms

Since providing privacy is a key issue for the broad success of enterprise Mashups [7], the issue of protecting exchanged resources against the Mashup-providing

API call	Description
`requestValues(serviceId, attributeId[], permit[])`	Request a set of attributes from a service
`sendMessage(serviceId, message)`	Send a message (e.g. attribute values) to another service
`requestPermits(serviceId[], attributeId[])`	Request permits to access a set of attributes from a service

Table 1. Simplified Mashup API

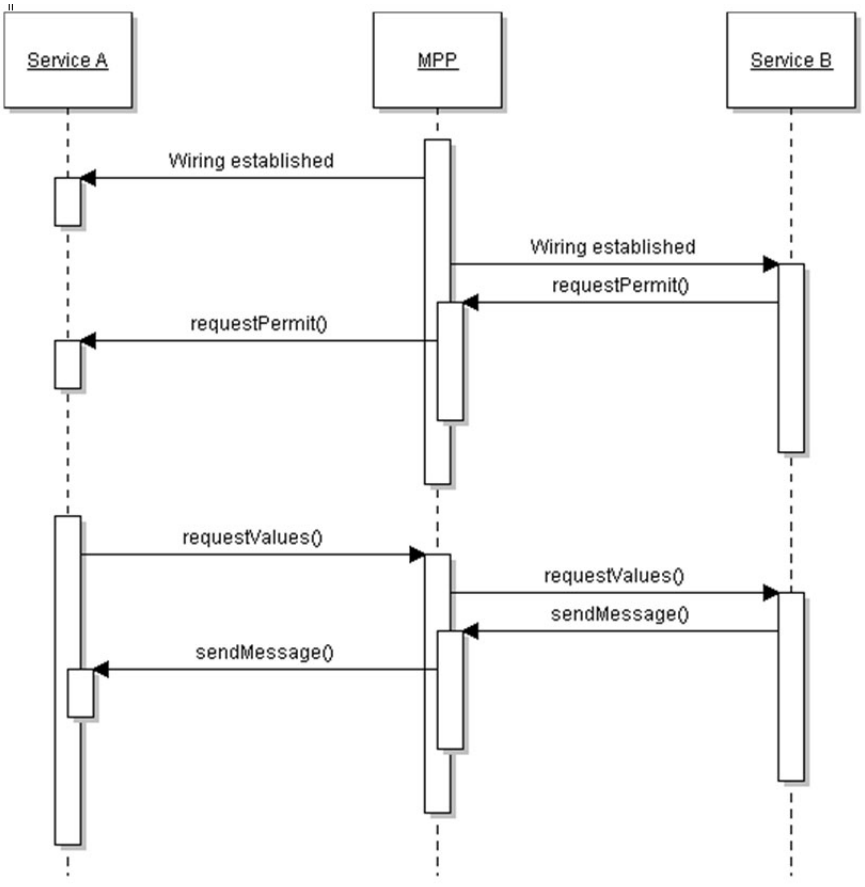

Fig. 3. Simplified Communication API High Level Overview

Platform is one of the major security challenges. In the component model we assume, the communication channels between Widgets (Wires) are embedded in and communicate via the Mashup-providing Platform. To protect these Wires against the MPP itself as well as other services or attackers, the traffic that is sent over the Wire should be encrypted. Therefore, we can formulate a first security requirement.

Requirement I: Resources transmitted via Wiring should be kept confidential against the MPP or other attackers

The way contemporary Mashups combine their underlying resources, the established browser security model assumes that all services are from the identical source, as services are integrated in a MPP acting as a web proxy. This behavior of internet browsers is one of the most pressing security issues associated with Mashups, because this enables external services to freely retrieve information from other services inside the Mashup. A malicious service can read, write and modify the content and even the behavior of all other services [3]. To mitigate this problem some sort of data flow control under the supervision of the user should be established. This leads to a second security requirement:

Requirement II: A facility for controlling the flow of resources across the Wiring should be established.

Another example for the lack of security in contemporary MPPs concerns the user's authentication credentials. The user provides her personal data to the MPP or a service inside the MPP to enable it to access backend-services. Both the MPP and individual services acting as adapters between the MPP and external data sources may impersonate the user to those external backend-services [12]. Also, a malicious provider of this proxy-service could abuse the user's authentication information to gain access to services outside of the MPP if the user has reused his password [15]. Therefore, the amount of credentials that have to be stored at the adapter or proxy services should be minimized to reduce the risks of impersonation.

Requirement III: The amount of credentials stored at adapter services should be minimized.

One possibility to address Requirement III is to store all credentials on the user side. However, managing lots of different credentials can be very cumbersome and will most likely overburden the users [16] [24], reducing the benefits from Mashups' ease of use. Therefore, we formulate a usability requirement:

Requirement IV: The burden placed on users for the management of credentials should be minimized

Requirements III and IV can be achieved by storing the credentials with a trusted third party on behalf and under the control of the user. This, however, requires (as suggested by the name) that the user trusts the third party. In the spirit of multilateral security we aim at minimizing the required amount of trust of users in the platform. Introducing a trusted third party will not minimize the required trust. It will only shift it to another stakeholder. This leads to another security requirement.

Requirement V: The required amount of trust in the system should be minimized, specifically, trust in third parties should be minimized

What is needed is a method which addresses the described security and privacy problems associated with MPPs without restricting their role in creating useful and functional combinations of services, such as acting as an intermediary platform facilitating discovery of compatible services, and offering hosting of the Mashup infrastructure. The focus is an end-to-end encrypted communication between the services within the platform, without the help of an additional trusted third party, and without trust in the platform.

4 Related Work

We are not the first to address security in Mashup environments. Hasan et al [12] present a system for controlling data flow within a MPP, using an independent 'Permit Grant Service' that acts on behalf of the MPP. Crites et al [5] propose a new browser security model, while Jackson and Wang [4] describe a solution using the existing browser origin policies. Both are valuable contributions in fixing the proxy/security model problem, however, none of the systems offer protection versus the platform itself. In the same vein, Keukelaere et al [3] propose a secure channel communication model. We extend on their work, adding encryption to protect confidentiality of the users' information versus the MPP. Zarandioon et al [6][17] offer the most comprehensive approach to date, offering both a client-side identity management providing data flow control as well as a single-sign-on (SSO) solution. However, it requires that each user administers a specific

identity-providing server outside the MPP, and understands a set of (non-trivial) mediating components. It also requires that inter-Widget (which are referred to as 'mashlets') communication is executed using specialized Widgets, which is a counterintuitive modification of Mashup programming practices, threatening Mashups' main selling point, namely their ease of modification by non-experts. We also build on Close's work [11], which describes transmitting fine-grained access credentials in HTTPS fragments, and provide additional protection in the cascading piping mechanism employed by adapter services.

5 Approach

We present an integrated approach, offering a comprehensive identity management system that can be integrated in today's mash-up platforms, while offering advanced security guarantees. This is realized by building on two main components: one system for securely delivering differentiated authorization tokens to backend services, building on the solutions presented in [11][12], meeting Requirements III, IV and V, and one system for preserving confidentiality and permitting data flow control of data exchanged within the Mashup-providing Platform, extending the work presented in [3][6][17], meeting Requirements I, II, IV and V. This section illustrates how the components are integrated within the platform, how data flows between them, and what functionality each component offers in the context of the larger system. It also gives details of the underlying system, specifically the cryptographic components used.

5.1 Key Management

To encrypt the Wiring-based communication between two Widgets in the Mashup, it seems logical to employ some sort of asymmetric cryptography to address the key distribution problem associated with n-to-n communication scenarios [18]. This approach has also been proposed in the context of Mashups by [17]. However, we want to avoid burdening the user with the management of a key server specifically for the purpose of using it in our Mashup platform. Studies have shown that it is hard for users to even use PKI systems [16], so having them manage key distribution seems like a long shot, especially in the context of Mashups, where the business value is in making things easier for users [1].

5.2 Identity-based Encryption

Identity-based Encryption (IBE) [19] offers a possibility to reduce the complexity associated with key distribution and management, directly using identifiers (e.g. users' email addresses) as public keys. While early IBE systems were not feasible for Web usage, recently a system has been proposed that can even be used as a client side JavaScript implementation in the Web context (namely, Guan et al's WebIBC [20]). This should be sufficient to meet performance requirements associated with Mashups. Conventional IBE requires a trusted third party acting as a

Private Key Generator (PKG) for the individual entities communicating within the system (usually users). We turn this system upside down in that in our approach, the user acts as a PKG for communicating Backend Services within a MPP, using service identifiers as public keys. This results in only the receiving Backend Service and the User being able to decrypt messages sent, which meets our confidentiality requirement (I). In addition, it is much more efficient to generate keys for the CPK cryptosystem used in WebIBC [20], than for RSA (as presented by Zarandioon et al [6]) as we do not need to perform primality tests, and even a relatively small CPK master key matrix scales to very high numbers of services (e.g. a 128x16 matrix supports 10^{32} services [20]). Thus, using asynchronous threads with native code embedded in JavaScript [6] is not necessary, and the system presented here stays much more responsive, and does not require native code running outside the browser sandbox, which adds a whole new layer of vulnerability [21]. Using this reversed approach to IBE (where the user

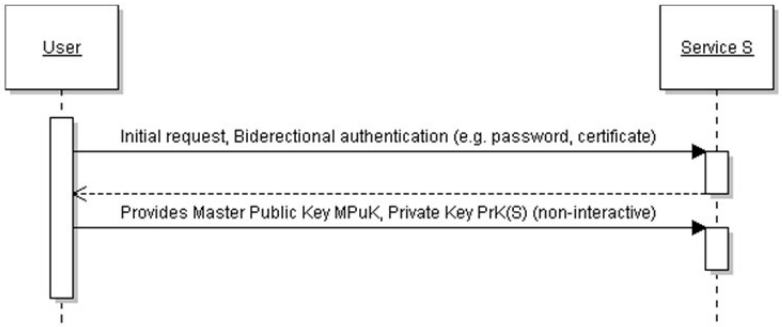

Fig. 4. Initial Setup and Key Exchange

acts as PKG for Backend Services) enables an initialization phase where, after mutual authentication between user and Backend Service, the key exchange can be executed in a non-interactive manner (see Figure 4). To avoid impact on the user interaction, we employ a cryptographically secure pseudorandom function at the user side to derive the master key pair needed for IBE and the password for authenticating to the service from a master credential supplied by the user (e.g. a password, a cryptographic key on a smart card). In future usage, the specific algorithms used will depend on the available (enterprise and/or Web) infrastructure and Mashup-providing Platform, so the algorithms and identifiers given here should only be seen as a proof-of-concept realization based on today's systems. The detailed steps executed are: 1. If there is only one credential available (e.g. password supplied by user), harden it using e.g. iterated hashing using the method described by Halderman et al [22] for hardening the password, creating the hardened credential $cred_{hard}$. Apply PBKDF2 from PKCS #5 if expansion is needed [23]. Then derive from the hardened credential a Master

Key Pair for Identity Based Encryption. Also derive a credential for mutual authentication with the Backend Service, using e.g. the method also described in [22] to derive a service specific password, using a deterministic pseudorandom function (PRF) [24]. If an enterprise smart card infrastructure can be used, the credentials may be stored directly on the smartcard, or passwords may be derived for interoperability with legacy services not supporting authentication by certificate, using e.g. the method proposed in [25]. Based on WebIBC, we use the revised CPK cryptosystem [20], which is based on elliptic curve cryptography, and generate keys as described in [24] [20], extracting them from matrices holding key pairs. We refer to them here as Master Public Key and Master Private Key ($MPuK$, $MPrK$). Formalized in equations:

$$cred_{hard} = h^n(cred)$$

$$(MPrK, MPuK, base_{auth}) = PBKDF2(cred_{hard})$$

$$cred_{auth} = PRF(base_{auth}, id(S))$$

2. Perform mutual authentication of user and Backend Service (e.g. by using certificates on both sides, or the user authenticating to a Backend Service providing an SSL certificate, generically referred to here as $cred_{authSrv}(id(S))$). This step can be performed within the Mashup-providing Platform if (and only if) it provides direct contact to the backend service domains. Our architecture meets this requirement by using the iFrame approach given in section 2.2 [3]. In that case, direct communication with the service domain is possible, and e.g. SSL may be used normally. This is used to establish an encrypted communication channel between service and user (e.g. using SSL/TLS). Of course, more elaborate and reliable mutual authentication schemes for the web would be preferable, but how to realize them is still unclear, as pointed out by Dhamija and Dusseault [26], among others.

$$cred_{auth} < - - - - - - - - - - - - - - - - - - > cred_{authSrv}(id(S))$$
$$\textit{mutual authentication}$$

3. Using the Master Private Key, the user extracts the service's private Key $PrK(S)$ based on the service's identifier also used as for the public key (e.g. domain name or Widget identifier used for API calls within the MPP) (a detailed description of the extraction procedure from the master matrices in CPK/WebIBC is given in [20]). He then sends Master Public Key and the Backend Service's private key he just generated to the service (across the established encrypted connection).

$$PrK(S) = extract(id(S), MPrK)$$

$$(PrK(S); MPuK) - >$$
$$\textit{encrypted transmission}$$

5.3 Confidentiality of Personal Information (Wiring)

Using those keys, services can now establish encrypted communication channels via the platform. Sent messages are encrypted with the target Backend Service's public key, which is derived from the $MPuK$ and the service identifier $id(S)$. The extracted key may be cached when the credential certifying the receiving Backend Service's permission to access the data is received (as described in the next section).

$$sendMessage(id(S), enc_{extract(MpuK,id(S))}(message))$$

5.4 Data Flow Control

As already described in section 2.2, the data flow control is realized by requiring certification of the user's permission for the transaction from the requesting service. The sequence is generally similar to the one proposed by Hasan et al [12], but we remove the need for the trusted third party. The general sequence of the operation is given in Figure 5: Upon establishment of the Wiring, permits for access of the relevant information are requested from the user via the requesting Backend Service.

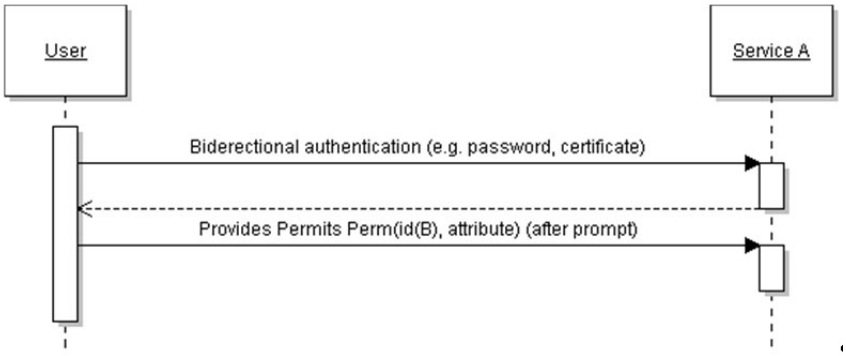

Fig. 5. Permit Granting Sequence

When a new Wiring is established (within the MPP's Wiring editor), the user is prompted to confirm (examples of Mashup permit dialogues are given in [12]). This is represented by the user's signature of a tupel consisting of requesting Backend Service, target Backend Service, and transmitted attributes. Using the terminology from 2.2:

$$Perm(id(A), id(B), attributes[]) = \\ Sign_{extract(id(U),MPrK)}(id(A), id(B), attributes[])$$

The pseudonymous communication via the MPP has the added advantage that the identity of Backend Services providing infomation resources is not necessarily revealed to consuming Backend Services.

5.5 Delegated Authentication and Authorization (Piping)

To minimize the credential exposure versus adapter services, we apply an approach consisting of two components. We address phishing and man in the middle attacks by employing PRFs to seed the credential (e.g. password) with the receiving domain (that is assumed to be able to authenticate itself using e.g. an SSL cert), an approach also employed in [22][25]. Our basic approach is to integrate with existing Single Sign On (SSO) systems to minimize credential exposure. In the context of the current web, this means providing some sort of Pseudo-SSO in the sense of de Clerq [27]. We chose to implement a password hashing approach, as several such systems exist, e.g. Halderman et al [22] or Zibuschka and Roßnagel [25], and are readily usable in all current browsers. To this end, we aim to establish direct channels within the MPP's Piping whenever possible. We use cascading iFrames for this, as proposed by [3], and illustrated in Figure 2. So, we derive the password, as already described in section 5.2, and implemented in [22][25]:

$$cred_{auth} = PRF(base_{auth}, id(S))$$

To further minimize risk of exposure, the next step would be to employ transmission in a HTTP fragment [5][11], which woild require minimal modifications of current practices. If the Backend Service behind the adapter supports it, fine-grained permits (based on the permits described in 5.4) can also be used to further minimize exposure. However, this is not possible given the current state of the art of Web SSO.

6 Discussion and Limitations

Our system does not require trust into the MPP. It is the first solution to offer meaningful cryptographic security versus the MPP to end users of mashups. It still supports the current practice of integrating Mashup building blocks into a common MPP proxy, which is the basis of operations for current MPP providers. Our solution goes beyond earlier work in that it provides strong security guarantees for the user, using strong cryptography, without requiring substantially modified user interaction, neither during authentication and identity management, nor during programming of Mashups. It integrates more readily into existing Mashup-providing Platforms, as it does not need client-side JavaScript Widgets. To realize this, we introduced the user as a private key generator within an identity-based encryption scheme securing Widget communication against the provider of a MPP. These mechanisms enable the providers of MPPs to complete their service with an efficient business model. Futhermore it makes mashups ready for enterprise use by offering a vector for integration in existing enterprise PKIs, while offering security across perimeter on individual level. To this end, we extend and integrate several earlier works, as well as propose new mechanisms like having the user act as a PKG in IBE systems for service communication. While the user-side components of our system may be implemented

using JavaScript, this of course results in a high risk of phishing attacks. This is not specific to our solution, but a general weakness of systems using JavaScript and/or redirects [26][28]. In enterprise systems, which are our main focus, implementing e.g. Browser plugins is plausible, and also allows for integration of enterprise infrastructure, such as smart card infrastructures. Most of the MPPs (e.g. Yahoo Pipes, Intel Mash Maker or iGoogle) provide a free playground to the users, where they can combine several services. But thinking about the scenario where a company provides business-critical, sensitive data to the Backend Services in a Mashup, the necessity of an enduring security model can't be dismissed. Only an elaborated MPP which protects privacy and offers a secure encrypted and still usable communication, without making development and usability significantly more complicated, will be viable in the field of enterprise Mashups.

7 Conclusion

In this contribution we have presented a comprehensive privacy enhancing identity management system for implementation in the context of Mashup-providing Platforms minimizing required trust in the platform. We achieved this by using Reverse Identity Based Encryption to preserve the confidentiality of resources that are transmitted via Wiring. Our system does not require trust into the MPP. Our solution goes beyond earlier work in that it provides strong security guarantees for the user, using strong cryptography, without requiring substantially modified user interaction, neither during authentication and identity management, nor during programming of Mashups. Due to these advantages, it is now possible for providers of MPPs to adapt their services to the enterprise domain, where sensitive data will be processed. Having a robust security model is therefore a key prerequisite for developing business models in the context of enterprise Mashups.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant Number 01BS0824 (COCKTAIL)

References

1. Merrill, D.: Mashups: The new breed of Web app, IBM developerWorks, Aug. 2006.
2. Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools. Service-Oriented Computing - ICSOC 2008. pp. 708-721 (2008).
3. Keukelaere, F.D., Bhola, S., Steiner, M., Chari, S., Yoshihama, S.: SMash: secure component model for cross-domain mashups on unmodified browsers. Proceeding of the 17th international conference on World Wide Web. pp. 535-544 ACM, Beijing, China (2008).

4. Jackson, C., Wang, H.J.: Subspace: secure cross-domain communication for web mashups. Proceedings of the 16th international conference on World Wide Web. pp. 611-620 ACM, Banff, Alberta, Canada (2007).
5. Crites, S., Hsu, F., Chen, H.: OMash: enabling secure web mashups via object abstractions. Proceedings of the 15th ACM conference on Computer and communications security. pp. 99-108 ACM, Alexandria, Virginia, USA (2008).
6. Zarandioon, S., Yao, D., Ganapathy, V.: OMOS: A Framework for Secure Communication in Mashup Applications. Proceedings of the 2008 Annual Computer Security Applications Conference. pp. 355-364 IEEE Computer Society (2008).
7. Mather, T., Kumaraswamy, S., Latif, S.: Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance, O'Reilly Media (2009).
8. Brown, D.H., Lockett, N.: E-business, innovation and SMEs: the significance of hosted services and firm aggregations, International Journal of Entrepreneurship and Innovation Management, vol. 7, 2007, pp. 92 - 112.
9. Hansen, M., Berlich, P., Camenisch, J., Clauß, S., Pfitzmann, A., Waidner, M.: Privacy-enhancing identity management, Information Security Technical Report, vol. 9, 2004, pp. 35-44.
10. Ennals, R.J., Garofalakis, M.N.: MashMaker: mashups for the masses. Proceedings of the 2007 ACM SIGMOD international conference on Management of data. pp. 1116-1118 ACM, Beijing, China (2007).
11. Close, T.: Web-key: Mashing with permission. W2SP 2008: Web 2.0 Security and Privacy 2008. IEEE Computer Society, Oakland, California (2008).
12. Hasan, R., Winslett, M., Conlan, R., Slesinsky, B., Ramani, N.: Please Permit Me: Stateless Delegated Authorization in Mashups. Proceedings of the 2008 Annual Computer Security Applications Conference. pp. 173-182 IEEE Computer Society (2008).
13. Open Mashup Alliance: OMA EMML Specification 1.0, http://www.openmashup.org/omadocs/v1.0/index.html.
14. JackBe: JackBe Mashup Editor and Composer, http://www.jackbe.com/products/composers.php.
15. Ives, B., Walsh, K.R., Schneider, H.: The domino effect of password reuse, Commun. ACM, vol. 47, 2004, pp. 75-78.
16. Whitten, A., Tygar, J.D.: Why Johnny can't encrypt: a usability evaluation of PGP 5.0. Proceedings of the 8th conference on USENIX Security Symposium - Volume 8. p. 14 USENIX Association, Washington, D.C. (1999).
17. Zarandioon, S., Yao, D., Ganapathy, V.: Privacy-aware identity management for client-side mashup applications. Proceedings of the 5th ACM workshop on Digital identity management. pp. 21-30 ACM, Chicago, Illinois, USA (2009).
18. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, vol. 21, 1978, pp. 120-126.
19. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. Advances in Cryptology. pp. 47-53 (1985).
20. Guan, Z., Cao, Z., Zhao, X., Chen, R., Chen, Z., Nan, X.: WebIBC: Identity Based Cryptography for Client Side Security in Web Applications. Distributed Computing Systems, International Conference on. pp. 689-696 IEEE Computer Society, Los Alamitos, CA, USA (2008).
21. Kemmerer, R.A.: Security issues in distributed software, SIGSOFT Softw. Eng. Notes, vol. 22, 1997, pp. 52-59.
22. Halderman, J.A., Waters, B., Felten, E.W.: A convenient method for securely managing passwords. Proceedings of the 14th international conference on World Wide Web. pp. 471-479 ACM, Chiba, Japan (2005).

23. Kaliski, B.: PKCS #5: Password-Based Cryptography Specification Version 2.0, 2000.
24. Abadi, M., Bharat, K., Marais, J.: System and method for generating unique passwords, U.S. Patent 6141760.
25. Zibuschka, J., Roßnagel, H.: Implementing Strong Authentication Interoperability with Legacy Systems. Policies and Research in Identity Management. pp. 149-160 Springer (2008).
26. Dhamija, R., Dusseault, L.: The Seven Flaws of Identity Management: Usability and Security Challenges, IEEE Security & Privacy Magazine, vol. 6, 2008, pp. 24-29.
27. de Clerq, J.: Single Sign-on Architectures. Proceedings of Infrastructure Security, International Conference. pp. 40-58 , Bristol, UK (2002).
28. Erlingsson, U., Livshits, B., Xie, Y.: End-to-end Web Application Security. 11th Workshop on Hot Topics in Operating Systems. USENIX Association, San Diego, CA (2007).