

WSEmail: Secure Internet Messaging Based on Web Services

Kevin Lux, Carl A. Gunter, and Michael J. May
University of Pennsylvania

April 2004

Abstract

Internet messaging (email) faces a variety of challenges not envisioned in its original design. Efforts have been made to compensate for limitations in security, flexibility, and integration within the existing standards and software base, but many aspects of these objectives can be obtained more easily from a fresh design. A fresh design will be easier to deploy if it is based on an emerging suite of high-level standards. In this work we explore the idea of redesigning email as a family of web services, an approach we call *WSEmail*. Existing standards and software for web services provide a high-level foundation with many of the desired properties. We illustrate a design, some sample applications, steps toward a theoretical foundation for its security, and an implementation. Our prototype illustrates a number of useful features: messages secured under federated identities, flexible treatment of access control, on-demand attachments, semantic routing, modular extensibility, and integration with instant messaging.

1 Introduction

Internet electronic mail (email) is based on a collection of protocols—SMTP, POP, IMAP, S/MIME—that have evolved over a vast installed base. The resulting system has shortcomings in the areas of flexibility, security, and integration with other messaging systems. To address these problems there are essentially three choices: (1) make incremental changes and overlays for the existing protocols, (2) redesign the system from a low-level, or (3) create a design from different foundations. The first option has been extensively explored without fully resolving the issues with flexibility, security, and integration. Some systems have pursued the second option; for instance, instant messaging uses different protocols, routing, and software from those on which Internet Email is based. This has the drawback of creating ‘stove pipe’ messaging systems and has not addressed problems with scalability and security. The third option has also been explored. For example, various custom systems allow a form of secure email through the use of HTML pages over HTTP and SSL. However, such systems have not been standardized and face scalability problems.

This paper explores the idea of basing email on the emerging infrastructure for web services, an approach we call *WSEmail*. Web services are data interchange protocols based on SOAP, WSDL, XMLDSIG, and other XML-based formats developed at W3C and other standards bodies. These protocols aim to support B2B transactions such as the transfer of an invoice between businesses of any kind. As such, a great deal of attention has been paid to security and flexibility. The desire for widespread adoption has also inspired attention to integration, such as making the protocols suitable for use over a variety of transport layers such as HTTP, TCP, and even SMTP. Moreover, several major vendors have produced software packages for implementing web services, and these packages are able to produce interoperable distributed systems, thus relieving the need for a single implementation base.

Our exploration of WSEmail is based on a prototype architecture and implementation. WSEmail messages are SOAP messages that use web service security features to support integrity, authentication, and

access control for both end-to-end and hop-by-hop message transmissions. Web services provide a foundation for extensibility by enabling many forms of SOAP services based on WSDL as a kind of type system. We exploit this by envisioning WSEmail as a collection of messaging services that can be added to the base system to provide features like semantic routing of messages. This also provides a way to integrate different messaging systems such as the usual email SMTP-style messages with various kinds of instant messaging systems. Our prototype system is built using Microsoft .Net and illustrates potential for new ideas in security, flexibility, and integration using applications such as on-demand attachments, routed forms, and integrated instant messaging.

The paper is organized in six sections. Following this introductory section we sketch the architecture of WSEmail focusing on its security assumptions. In the third section we introduce three applications we have explored with WSEmail. In the fourth section we describe one of our application protocols theoretically and argue that basing email on web services can aid the application of advances in security analysis to new messaging protocols. In the fifth section we highlight points about our implementation. The sixth section concludes.

Interested readers can find more information on our project web page at wsemail.ws. This page provides optional supplements like sample XML messages, screen shots of the prototype interface, formal (theorem prover) specifications, UML for the implementation classes, and other information too bulky to include in the paper.

2 Architecture

WSEmail is designed to perform the functions of ordinary email but enable additional security functions and more flexibility. The primary strategy for the design is to import these virtues from the standards and development platforms for web services. Web services are server functions in which messages are based on a high-level representation using XML and a standard called the Simple Object Access Protocol (SOAP) (www.w3.org/TR/soap). SOAP messages provide a header and a body. The header of the SOAP call indicates the type of service and provides security information based the XML security standards for digital signatures (XMLDSIG) (www.w3.org/TR/xmlsig-core) and encryption (XENC) (www.w3.org/TR/xmlenc-core). The header typically also contains other information like a time stamp. A primary application target of these standards is automating business-to-business (B2B) data interchange. This drives the fundamental premises that also make web services well suited as a foundation for messaging. It is assumed that there will be a wide range of types of B2B services so the framework needs to be very flexible. The adoption of the system depends significantly on its widespread usability so there is considerable attention to integration. Most business interchange functions require careful security so a wide range of security functions are provided. By contrast, systems like SMTP and S/MIME provide less to address these concerns. Examples of difficulties arise in areas like extending the SMTP implementations to include new functions and adding security features between hops rather than end-to-end. A listing of a number of these limitations can be found on the mail-ng list (www.imc.org/mail-ng) of the Internet Mail Consortium (IMC) (www.imc.org).

We now explore how we aim to deliver some of the virtues of web services as a foundation for messaging by describing WSEmail architectural concepts for the distributed components and messages, the client, and the server.

The baseline protocols illustrated in Figure 1. In the most common case, similar to an SMTP message, a Sender Client SC_1 makes a call on its Sender Server SS to send a message M_1 . This and other calls are SOAP calls over TCP; the message M_1 is in the body of the SOAP message and the SOAP header contains information like the type of call and security parameters. The message itself is structured as a collection of XML elements, including, for instance, a subject header. A sample trace of WSEmail messages can

be found at wsemail.ws/messages.html. After receiving the call from SC_1 , the server SS makes a call on the Receiver Server RS to deliver the mail from the Sender Domain SD into the Receiver Domain RD . The Receiver Client RC makes calls to RC to inquire about new messages or download message bodies. In particular, RC makes a call to RC to obtain message headers and then requests the message M_1 if it wishes. Security is based on the standards mentioned above for web service security, possibly supported by encrypted tunnels. In general we assume that hop-by-hop confidentiality will satisfy most applications, so communications between the nodes can be protected by TLS. Clients like SC_1 and RC are typically authenticated to their servers by a password, while servers authenticate themselves using certificates. Such certificates are used to sign messages using XMLDSIG and may also be used with TLS. For instance, the message from SC_1 to RC will be given an XMLDSIG signature by SS that is checked by both RS and RC .

A variety of variations on this basic call sequence and security model are possible. As an example, consider an instant message M_2 dispatched from a client SC_2 to RC while SC_2 is outside its home domain SD . In this case SC_2 contacts SS to obtain a security token T that will be recognized by RS . Once this is obtained, SC_2 sends M_2 authenticated with this credential to RS and indicates (in a SOAP header) that it should be treated as an instant message by RS and RC . Instant messages are posted directly to the client, with the client now viewed as a server that accepts the instant message call. RS and RC are able to apply access control for this function based on the security token from SC_2 . This token is recognized because of a prior arrangement between SS and RS .

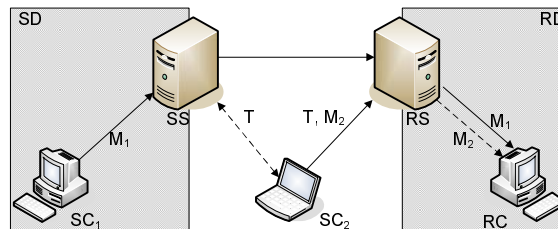


Figure 1: Messaging Architecture

These examples help to frame our discussion of the security architecture of WSEmail. While many approaches are possible, we have explored a system that aims to address some of the issues that have impeded the deployment of a satisfactory security infrastructure for email in the Internet. Much hope was attached to the idea of a global Public Key Infrastructure (PKI) capable of securing email with public keys based on S/MIME. However, several issues that have held back the widespread use of public keys. These include: user management problems, challenges with revocation, and limited flexibility. When one compares this to the widespread use of SSL certificates, some themes emerge about the asymmetry between ‘client’ machines like SC_i , RC , which are typically managed by non-professional end users, and ‘server’ machines like SS , RS , which are typically managed by professional system administrators. End users have difficulty managing certificates, especially when the user has multiple client machines (such as an office machine, home machine, and laptop). In particular, certificates of end users often are not trusted with long revocation windows, and the costs and complexity of online revocation diminish the value of using public keys. Rigid certificate formats and problems with CA business models provide disincentives to many potential users. The protocols and software platforms for web services offer opportunities for addressing these challenges. In particular, there are several efforts underway to develop forms of *federated identity* in which security tokens need not have universally standardized formats. A typical application arises when business partners need to authenticate exchanges without merging their internal authentication systems. Notable efforts in this area include the Liberty Alliance Project (www.projectliberty.org) and the inclusion of supporting features in the Microsoft .Net platform (www.microsoft.com/net).

Our design is based on a three-tier authentication system combined with an extensible system of federated identities. At the top is a certificate authority such as Verisign (verisign.com) which is capable of providing certificates to the system administrators that represent enterprises and ISPs. These certificates suffice to authenticate servers to each other in the way SSL certificates currently do. The private key of the root certificate is highly protected—probably used only offline—and has a lifetime measured in decades. The server certificates have lifetimes measured in months (typically 12) and may be compared regularly against

Certificate Revocation Lists (CRLs). Clients, by contrast, typically authenticate themselves to their servers using a shared password. This part of the design is fairly standard. Problems arise when a client, which authenticates with a password, needs to authenticate directly to an outside entity such as a server or client at another enterprise. For this we employ two strategies. First, we compromise by delegating confidentiality functions to servers on a hop-by-hop basis. This means that servers may be able to see cleartext messages, but this will be acceptable for many applications and it is the norm for Internet email currently. Second, we exploit the emerging standards and platform support for federated identities and provide extensible support for inter-enterprise authentication based on the ideas that are intended to make this possible for web services B2B transactions. The resulting system achieves less end-to-end security than S/MIME encryption, but is more deployable, especially if secure web services become widespread. We illustrate the approach in the next section to provide security for ‘on-demand’ attachments: email that leaves attachments on the sender’s server.

The WSEmail client is highly extensible but also hides complicated details of the overall system from the end-user. After an initial authentication of the user, additional authentication information (such as a federated identity) can be created transparently. Plugins are supported by the client and allow for new functionality to be downloaded from the web. On-demand attachments are an example of such a plugin, as are a variety of kinds of attachments with special semantics. A party that sends a message with such an attachment automatically includes information for the receiver on where to obtain the software necessary to process the attachment. The client provides hooks for plugins to access security tokens, after first performing an access control check on the plugin. A figure illustrating the client components is available at wsemail.ws/client.html and screen shots of the GUI can be seen at wsemail.ws/screenshots.html.

The WSEmail server exploits plugins extensively to enable flexibility. The core server has very limited functionality. Plugins provide access to databases, queue messages for delivery, retrieve stored messages and provide message formatting. Each of those subsystems can be replaced with alternative plugins to allow access to new types of data stores or alternative queuing strategies. WSEmail external plugins are programs that are not critical to proper execution of the server but provide enhanced features. There are two types of external plugins which we classify as *message-dependent* or *RPC-like*. As the name implies, message-dependent plugins require a message and then perform an action on it. Instant messaging is a message-dependent plugin; it examines incoming messages and can send the message directly to a client’s screen instead of allowing default delivery on the server. RPC-like extensions provide a capability to interact with the server without the need for a message. An example is the plugin for processing requests for federated identities, which requires information about keys but is not a message in the sense of email. A figure illustrating the server components is available at wsemail.ws/server.html.

3 Applications

WSEmail offers the possibility to have rich XML formats, extensible semantics on clients and routers, and a range of security tokens. Since there are substantial development platforms for these features from major software vendors, it is easy to use WSEmail as a foundation for a suite of integrated applications that share common code, routing, security, and other features. As an illustration, we sketch three applications that we implemented with our prototype system.

On-Demand Attachments Suppose SC wishes to send an email to a large collection of recipients containing a large document. Suppose, moreover, that only a fairly small, but unknown, subset of the recipients are likely to actually want to look at the document. A straight-forward but expensive approach is to send email to all of the recipients with the document as an attachment. A more efficient approach is to place the document on a web page and send a URL to the recipients. If the document is sensitive, it may be necessary

to insist on some access control for the web page, which assumes there is a way to authenticate the recipients if the visit they page to get the document. Systems have been implemented to assist this form of distribution; for instance, it is possible with Microsoft Sharepoint (www.microsoft.com/sharepoint) including hooks into the Outlook email client (office.microsoft.com).

WSEmail can be used to provide a similar functionality based on federated identities. Figure 2 illustrates such a design based on a nine message protocol. In the first message, the client SC sends a WSEmail message with an attachment and an indication that the attachment should remain on the server. For the illustration we assume only one recipient RC located in another administrative domain RD with its own access control system different from the one in SD, the domain of RC. The server SS in SD accepts the call, stores the attachment, and sends the message and a reference to the attachment to the server RS of the recipient. Clients typically poll their server to inquire about messages. WSEmail can do this or the server can inform the client of messages. In the third, fourth, and fifth messages, RC is informed of the message, requests, and receives it. Upon finding the reference to the attachment at a server in another domain, RC requests a token for communicating with that server and gets one from its server RS. The client RC then uses this token to request and obtain the attachment from SS. We discuss the security analysis of this protocol in the next section. WSEmail can be used to implement similar protocols like certified email or digital postage stamps.

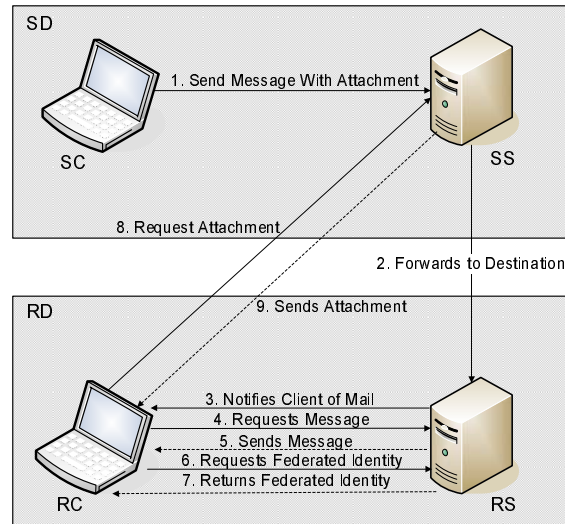


Figure 2: On-Demand Attachments

Integrated Instant Messaging Instant messaging is similar to email but is intended for communicating short text messages synchronously. Instant messaging systems are typically disjoint from email systems using different clients, servers, routing, and security. This is unfortunate since the two messaging systems have many things in common. We experimented with a form of integration for the two by allowing WSEmails to be marked as instant messages. Such messages are posted directly to a window on the recipient client by the client server, subject to an access control decision. Our implementation uses the same client, server, software, and security as the email functions. There is an option that allows multiple parties to use SSL tunnels to a single server.

Routed Forms Many organizations are working to carry out more of their management of workflow (*viz.* forms) using web forms or other online techniques. In implementing such a system, there is choice between a centralized system where a single web server is used by all parties, versus a decentralized system where information is routed by email. Email systems tend to work better with loosely described workflow and loosely coupled participants, such as *ad hoc* collaborations between enterprises where neither organization is willing or able to carry out all functions on a web server managed by the other party.

WSEmail can be used to restore some of the structure of web-based centralized solutions while maintaining the decentralization and flexibility of email. Our prototype can be expanded with plugins for specific message semantics so we developed an experimental plugin for processing approvals. An example is shown in Figure 3. The general idea is that a collection of approvals are required for an action as requested in a form. The form is an XML document that includes routing information. In the example, a claimant Q seeks

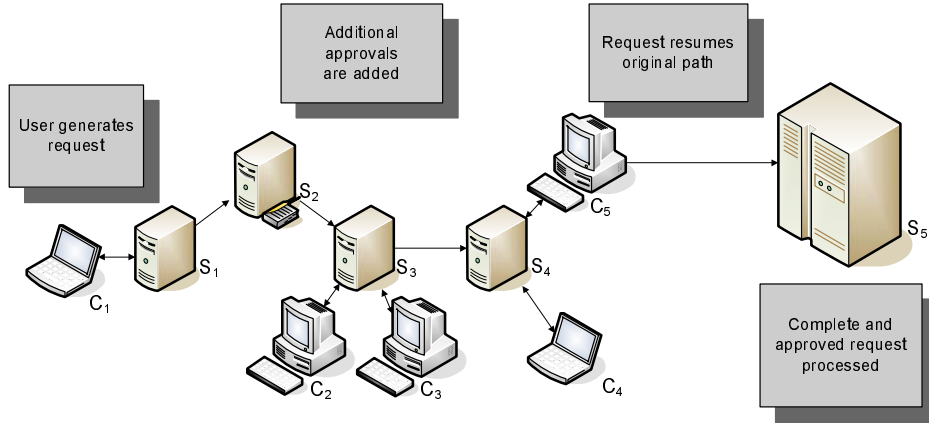


Figure 3: Routed Forms

the approval, which requires the approvals determined by router S_2 and by a manager C_5 . WSEmail is sent from the claimant to his server and from there toward C_5 via S_2 . The router S_2 determines that approvals are needed from parties C_2 and C_3 before the form is relayed to C_5 . It is forwarded to the server S_3 for C_2 , which also happens to be the server for C_3 . After processing by C_2 , the party C_3 receives the request and decides to delegate his decision to C_4 . The message is then sent to his server S_4 and, after approval by C_4 , resumes its original path to C_5 , who approves and forwards to S_5 for checking and processing.

This workflow and its implementation have several features of interest. The routing by S_2 can be based on the detailed semantics of the form, including features like the size, nature, and maker of the request. Our implementation is based on Microsoft's WS-Routing, which supports dynamic changes in routing. If any of the parties lacks the software to understand the semantics of the form, the software can be downloaded and installed dynamically. Dynamic plugins are enabled based on digital signatures and security rules of the party making the installation. In particular, there is no requirement for a central server to be created in advance. On the other hand, the semantics of the plugin can dynamically create a centralized server functionality (like S_3) possibly even across enterprise boundaries. Our ongoing work focuses on extending workflows like this to include constraints [2, 4] that influence routing. Screenshots of the workflow application GUI are given at wsemail.ws/workflow.html.

4 Theory

Trends in the analysis of security protocols will aid the development of WSEmail security by supporting the analysis of web services generally [3] (`securing.ws`) and email specifically [1]. To illustrate this, we now sketch a formal version of the on-demand attachments protocol in Figure 1 and state a formally-verified correctness property. We assume familiarity with digital certificates Γ and hope the notation is sufficiently self-explanatory. Let S be a public key signature function and H be a one way hash.

Definition: (Public Key Authentication Using a Timed Nonce) We write $A \rightarrow B : M$ (pkey Γ, r, t) to indicate that A sends to B a message of the form: $A \mid M \mid r \mid t \mid S(\text{priv}(\Gamma), H(A \mid M \mid r \mid t))$. Here t is the current time according to the clock of A and r is a random number selected by A . The principal B processes this message by checking the following conditions in this order: the time t is not older than a given threshold; the validity interval of Γ includes the current time; the nonce r is not in the replay cache of B ; the signature checks using public key in Γ ; the certificate Γ is trusted. If any of these fails then the remaining steps are omitted and the message is discarded. If all of the conditions succeed, then r is added to the replay cache with an expiration time determined by a given threshold. In this case the message is said to be *valid*. \square

Definition: (Salted Password Authentication) We write $A \rightarrow B : M$ (pswd P, r, t) if A sends B a message of the following form $A | M | r | t | \text{MAC}(P, A | M | r | t)$. Here t is the current time according to the clock of A and r is a random number selected by A . The principal B processes this message by checking the following conditions in this order: the time t is not older than a given threshold; the nonce r is not in the replay cache of B ; the MAC is correct for the password associated with A . If any of these fails then the remaining steps are omitted and the message is discarded. If all of the conditions succeed, then r is added to the replay cache with an expiration time determined by a given threshold. In this case the message is said to be *valid*. \square

Protocol: On-Demand Attachments

Initiation Distribution procedures are used to ensure that the following passwords and keys are known only to the specified principals: principals SC and SS share a password P_{SC} ; principal SS has the private key for a certificate Γ_{SS} that is trusted by the other principals; principal RS has the private key for a certificate Γ_{RS} that is trusted by the other principals; principals RS and RC share a password P_{RC} . Principal SC creates a message M that includes the return address SC and an attachment N and sends:

Msg 1 $SC \rightarrow SS : SS | (RC | RS) | M | N$ (pswd P_{SC}, r_1, t_1)

If SS gets a message of this form, it checks this using the password P_{SC} . If the message is valid, SS selects a locally unique reference N^* , stores the attachment N with this reference and the name SC and sends:

Msg 2 $SS \rightarrow RS : RS | (RC | RS) | M | (SS | U^*)$ (pkey Γ_{SS}, r_2, t_2)

If RS gets a message of this form, it checks this using $\text{priv}(\Gamma_{SS})$. If the message is valid, RS selects a locally unique reference V^* , stores $M | (SS | U^*)$ with this reference and the name SC and sends:

Msg 3 $RS \rightarrow RC : RC | (RC | RS) | V^*$ (pkey Γ_{RS}, r_3, t_3)

If RC gets a message of this form, it checks this using $\text{priv}(\Gamma_{RS})$. If the message is valid, RC may send the following request:

Msg 4 $RC \rightarrow RS : RS | V^*?$ (pswd P_{RS}, r_4, t_4)

If RS gets a message of this form, it first checks whether V^* is associated with RC. If it is and the message is valid based on the password of RC, then it sends the following message:

Msg 5 $RS \rightarrow RC : RC | V^* | (SS | U^*)$ (pkey Γ_{RS}, r_5, t_5)

If RC receives a message of this form, it checks this using Γ_{RS} and may choose to retrieve the attachment N . The reference V^* is included to ensure that this is the response to Msg 4. To do this it needs a token to authenticate to SS. If it does not have one it may request this by a key pair with K as its public key and sending:

Msg 6 $RC \rightarrow RS : RS | \text{token}(K)?$ (pswd P_{RC}, r_6, t_6)

When RS receives a message of this form, it checks validity and creates a certificate Γ_{RC} for use by RC and sends:

Msg 7 $RS \rightarrow RC : RC | \Gamma_{RC}$

When RC receives a message of this form, it checks and stores the certificate for use in accordance with its lifetime. It may choose to obtain the attachment by sending:

Msg 8 $RC \rightarrow SS : SS | U^*?$ (pkey Γ_{RC}, r_7, t_7)

When SS receives a message of this form, it checks validity using its code for credentials from RS and confirms that the reference U^* is associated with RC. It sends:

Msg 9 $SS \rightarrow RC : RC | U^* | N$ (pkey Γ_{SS}, r_8, t_8)

When RC receives a message of this form, it checks validity and makes the attachment available on RC

We specified this protocol formally using the protocol verifier of Bruno Blanchet (www.mpi-sb.mpg.de/~blanchet/crypto, version 1.10 patch level 2) and proved the following correspondence theorem: *if RC retrieves an on-demand attachment with SC as return address, then SC sent the attachment*. The specification can be found at wsemail.ws/spec.html and the output of the proof is at wsemail.ws/proof.html.

5 Implementation

Our WSEmail prototype runs on Windows server and client systems. It was implemented over the .Net framework and relies on Web Services Enhancement (WSE) 1.0, CAPICOM 2.00, SQL Server 2000 (to store messages for the server), and IIS 5.0. The current version consists of 68 interfaces and 343 classes organized into 30 projects (see wsemail.ws/uml.html for a UML model illustrating the design). About 98% of the software is C# .Net-managed code created with Microsoft Visual Studio. Our instant messaging system also exploits a TLS package from Mentalis (mentalis.org) since the .Net platform does not provide native support for TLS. A beta version of WSE 2.0 was released recently, but the documentation was too thin to allow us to use the replay protection mechanisms or support for federated identities in it, so we implemented these things ourselves on top of WSE 1.0.

WSEmail uses DNS SRV records (ietf.org/rfc/rfc2782.txt) to determine routing. This makes it possible to run WSEmail over other protocols without changing the way DNS is queried, and we can exploit the priority and weight attributes in the records. These properties of the SRV record allow for future enhancement and present day configuration that is extremely similar to the way SMTP is deployed now.

6 Conclusions

We have explored WSEmail, the development of email functions as a family of web services, by developing a prototype system based on an architecture that emphasizes flexibility, security, and integration. We have shown that WSEmail has advantages in implementation because of the existence of substantial development environments for web services. WSEmail also has advantages in theoretical analysis because of advances in models and automated analysis for new security protocols. We have shown a range of applications, but these only scratch the surface of what is possible given the inherent flexibility of WSEmail. For widespread use, WSEmail faces substantial problems with standardization, and its efficiency needs further review. However, it well-suited to some high-security applications even now, and offers ideas in exploring the general design space for Internet messaging.

This work was supported by a gift from Microsoft University Relations, NSF grants CCR02-08996 and EIA00-88028, ONR grant N000014-02-1-0715, and ARO grant DAAD-19-01-1-0473. We are grateful for discussions of WSEmail that we had with Martín Abadi, Karthikeyan Bhargavan, Luca Cardelli, Dan Fay, Cedric Fournet, Andy Gordon, Bjorn Knutsson, Eric Freudenthal, Kaijun Tan, and Jianqing Zhang.

References

- [1] Martn Abadi and Bruno Blanchet. Computer-assisted verification of a protocol for certified email. In Radhia Cousot, editor, *Static Analysis, 10th International Symposium (SAS'03)*, volume 2694 of *Lecture Notes on Computer Scienc*, pages 316–335, San Diego, CA, June 2003. Springer.
- [2] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, 1999.
- [3] Karthikeyan Bhargavan, Cedric Fournet, and Andrew Gordon. A semantics for web services authentication. In *2004 ACM Symposium on Principles of Programming Language*, pages 198–209, Venice, January 2004. ACM.
- [4] Kaijun Tan, Jason Crampton, and Carl A. Gunter. The consistency of task-based authorization constraints in workflow systems. Technical report, Penn, January 2004. To appear in CSFW '04.