An Architecture for Private Messaging over the Web

Fred Dushin
DRAFT Version 0.9
Jan 31, 2009

Motivation

The Internet has enabled forms of communication that would have been unimaginable only a generation ago, be it through electronic mail, bulletin boards, instant messaging, or VoIP. This form of communication has become ubiquitous in modern society, in the private, commercial, and government sectors, to the point where it is almost unimaginable how society could function without it.

At the same time, security technology has only entered into these forms of communication at the periphery. Some users, if they are lucky, can access email ports over SSL, but email messages are sent across MTAs over unprotected channels. Most instant messaging systems send content in plaintext, making them vulnerable to eavesdropping either by passive observers of network traffic, or administrators of back-end routing technology.

The reasons for a paucity in security technology for internet messaging systems are many. Many users don't see the need for message confidentiality, reasoning that they have "nothing to hide", and that the value of any information transmitted over this medium is of minimal value (even if the messages themselves contain confidential information, such as a password or credit card number, as they often do).

Another reason for the lack of security technology adoption is a fundamental lack education in security technologies, generally, and in some technologies the lack of coherent tooling and infrastructure for making security technologies easy to comprehend and use. For example, the ability to sign and encrypt messages using asymmetric keys and X.509 certificates is available today for most email clients, yet the technology simply isn't used by most users (at least in the private sector).

While addressing the education issue is a laudable objective, and one which should always be pursued, the fact of the matter is that training the average user in the subtleties of asymmetric key technologies and trust decisions is likely to take years, if not decades, for the knowledge of basic security concepts to be fully common among users. When it comes to the average users, security is conceptually equivalent to provision of an identifier (username) and secret (password), perhaps with the faint confidence of a padlock icon on the browser window, even if the user doesn't exactly know what it means. Unfortunately, for the foreseeable future, we simply can't hope for much more than that.

Another obstacle to widespread adoption of asymmetric key technology in private messaging is the cost and complexity associated with PKIs. The life-cycle of key

material is especially difficult, given the lack of education and tooling. Not only must users know how to generate keys (e.g., by knowing the appropriate key type, length, strength, etc.), users must also manage these keys properly, by storing them with sufficient protection (OS permission and cryptographic), and dealing with key compromise. Many vendors, including OS vendors, have tried to provide solutions to these problems, but even with these efforts widespread adoption has been limited.

In this paper, we propose a system for private messaging which attempts to resolve some of the security and usability issues described above. The objective of this system is to provide a mechanism for guaranteed confidentiality and integrity protection of messages sent between participants in a message exchange, without the threat of eavesdropping on the part of the messaging infrastructure or passive listeners on the network.

Like any piece of security technology, the system described here is not perfect, and does suffer from some security and usability weaknesses, which we describe at the end of this paper. These weaknesses, we argue, may limit the space of users of this particular system, depending on the value of information conveyed. However, we feel that many users -- perhaps the majority -- will still find the system useful.

Rich Internet Client Technology

With the advent of rich client scripting languages (Javascript) and lightweight client platforms (mobile, iPhone), more and more applications are making use of the Web Browser as an application platform. With rich client technologies, software vendors can readily distribute applications over HTTP, which has become the de facto standard protocol for distributed applications.

Like technologies that preceded it but which ultimately failed (Java Applets, ActiveX Controls), technologies like Javascript allow the Web Browser to host a certain percentage of a web application. That is, instead of having the web server do the majority of the presentation of content, whereupon the web client refreshes or requests updated content, rich client applications leverage the Javascript interpreter (or other plugin technology) to share in the processing and presentation of content. As client-side technologies mature, with the advent of sophisticated frameworks for the presentation of information (e.g., Dojo, SproutCore), the Web Browser is playing an increasingly important role in the development of distributed applications.

Standardization is an increasingly important theme in rich client applications. While technologies like Adobe Flex, Macromedia Flash, and Microsoft Silverlight provide compelling development platforms and technologies for rich client applications, these offerings are by their nature closed and proprietary. As proprietary technologies, their portability to diverse web platforms is limited by the willingness of their sponsoring organizations to spend the effort to develop, test, and maintain ports to browser and operation systems other than the “big 3” (Windows, OS X, Linux; IE, Firefox, Safari/ WebKit).

Standardized behaviors for web content rendering, as well as standardization of languages (e.g., via ECMA) are all considered requirements for the parts of this system that use rich web client technology.

Note that while the Web browser is an important and compelling platform for rich client technology, the mobile space is quickly becoming an equally compelling platform, as well. In fact, as mobile platforms become more powerful and connected, they are starting to support rich client applications, as well as native clients. We therefore regard many mobile platforms as equivalent to the Web platform in importance, if not in fundamental architecture, even if the technologies differ slightly; they are both kinds of Web client.

Transient Key Generation

Central to the design of this system is the requirement that users not be required to manage cryptographic keys persistently. One way to do this is to derive an asymmetric key pair from a suitably complex password, or better, and arbitrarily complex pass phrase. Additionally, we require that the key pair can be reproducibly generated from the same pass phrase. As long as reproducibility is assured, users can generate transient key pairs (for the lifetime of a web session) from a pass phrase, which can be readily remembered.

We propose that the system derive a unique RSA key pair, <PRIV, PUB>, from an arbitrarily complex, though potentially easy to remember pass phrase PASS. This derivation can be achieved by using PASS as a seed for a PSRNG. Furthermore, we propose that this derivation take place on a per-session basis, and in the Web client (e.g., Javascript, iPhone App, etc) only. In particular, the PRIV part of the key pair should never escape the boundary of the web client, and should not require persistent storage. While some Web clients may optimize usability by caching derived key pairs, or even the passwords from which they are derived across Web sessions, this functionality is out of the scope of the current version of this paper.

While the cryptographic key pair is generated and stored internally in the Web client, there is some need to serialize the public key, for the purposes of key distribution to other clients. We propose that the public key be distributed in a serializable form, roughly along the following structure, using XML schema as the structural description language:

```
<complexType name="PublicKeyType">
  <sequence>
        <element name="Version" type="tns:VersionType"/>
        <any namespace="##any" processContents="lax"/>
  </sequence>
  <attribute name="algorithmIdentifier" type="string"/>
</complexType>

<complexType name="VersionType">
  <attribute name="major" type="integer"/>
  <attribute name="minor" type="integer"/>
</complexType>
```

```
<complexType name="RSAPublicKeyTpe">
  <sequence>
        <element name="e" type="integer">
        <element name="n" type="base64Binary">
  </sequence>
</complexType>
```

Note that XML is one of many possible info-sets for this structural description. Others include DER, CDR, or other proprietary equivalents.

We discuss the problem of public key distribution below.

Message Protection

With the provision of a key pair, Web clients can readily sign and encrypt messages for intended targets. The technologies here are well understood; the client's private key is used for message signing and decryption, while the public key is used for message encryption and signature verification.

We foresee no need for additional protections, such as time-stamps or nonces for replay detection, seeing as this is a messaging system, and not an RPC mechanism. Of course, for the management of messages as displayed in the Web client, some metadata about the message, such as a time-stamp or an ordinal, may be signed and encrypted. The details will become more clear with detailed use patterns.

Again, all cryptographic message protection operations are to be done in the Web client, as the private key material never moves outside of the boundary of this container. For security reasons, we require that all messages be signed and then encrypted before delivery.

A Lightweight Messaging Server

We propose a fast Web service for the transmission of signed and encrypted messages between Web clients. Using this service, a Web client can deliver a message to an intended target by posting it to the Web server, whereupon the intended recipient can pull the message off the server.

The messaging server should support the notion of a "chat room" or group, but we intend that a group in this sense is simply a way to partition the set of messages sent through the messaging server. Moreover, while messaging between more than 2 parties is feasible, we envision messaging to take place between exactly 2 participants.

We propose a RESTful service, supporting posting and retrieval of messages using POST and GET verbs. For example, a POST to the URI

http://localhost/messaging/rooms/MyChatRoom/

would result in the sending of a message to the room "MyChatRoom", whereas a GET with the same URL would retrieve all messages sent to that room. Equivalently, a GET to the URI

http://localhost/messaging/rooms/MyChatRoom/37

results in the retrieval of the 37th message posted to the room.

Note that persistence of messages posted to a room or server is outside of the scope of this paper. Some service implementations may support message persistence; others need not.

Because the cryptographic operations on messages (signature, encryption) are done in the web client, the message contents are opaque to the web server, with perhaps the exception of some metadata about the message (such as an identifier indicating the intended recipient) which otherwise cannot be encoded in the URL used to push or pull the message off the web server.

Protocols and Message Bindings

We propose HTTP as the standard protocol through which messages are delivered between Web clients through the messaging server. We choose HTTP for several reasons:

- *Ubiquity of tooling and technology.* Given the fact that the Web client technology is likely to be based on current scripting technology, HTTP is the likely candidate, given the availability of libraries and tooling for this protocol.
- *Firewall Traversal.* Many corporate firewalls block protocols other than HTTP, and additionally perform protocol analysis on individual messages to prevent data leakage. It is therefore important that messages conform to standard message protocols likely to be permitted to run over corporate networks, even if the contents themselves cannot be inspected.

The message bindings themselves are currently less important, as long as they are easily encapsulated and consumable by the messaging server. We will need to put some thought into message size limitation, in order to mitigate the possibility of unwieldy memory consumption in the message server, but these details can be worked out in the implementation.

Code Integrity

Because all of the cryptographic operations, on which the trust in the privacy of messages is based, are performed in the Web client, and likely in a manner where the actual code used to perform these cryptographic operations is downloaded from the Web server and interpreted locally in a Web browser, it is imperative that the client be able to trust that the code performing the cryptographic operations is in fact trusted (for example, that the code itself does not function as a kind of “man-in-the-middle”.

We therefore propose that at least in the case of Web applications, that HTTPS be the required protocol, where the client can place trust in the server based on the server’s endorsement by a Certificate Authority.

Further analysis of the security model operating environment may be required, for example, to understand code integrity in the context of a potentially malicious execution

environment (i.e., the Web browser), and what threats exist for interpreted applications, even if the code itself has been downloaded from a trusted source.

For standalone clients (e.g., think desktop client, mobile/iphone, etc), the HTTPS is a recommended protocol, on the assumption that the code itself is trustworthy, or at least the trust in the code has been established out-of-band. Code-signing is a mechanism whereby trust in standalone code can be established.

Trust and Key Distribution

In many ways the biggest weakness of public key cryptography is the problem of key distribution; in order for a relying party to render a trust decision on the basis of a signature, or to know that a message will be sent confidentially to another party, the relying party must have trust that the public key material used to verify the signature or encrypt the message is precisely *the* key that corresponds mathematically with the private key known only to the other party.

Several trust models have been devised to resolve this issue, with varying degrees of success. The X.509 standard, for example, whereby trust in public keys is based on implicit trust in well-defined certificate hierarchies, has had a good degree of success in e-commerce and some enterprise scenarios, even if the users of such systems are not fully aware of the mechanics of the trust decisions involved. (What percentage of users, for example, have taken the time to verify the cryptographic hashes or signatures on the Web browsers used as part of an e-commerce transaction, and if so, on what basis was that trust predicated?) Under this model, a relying party trusts a key because it was signed by a reputable authority the user implicitly trusts, and further that the authority has done due diligence in verifying the identity of the trusted party, and perhaps that the trusted party abides by standard procedures and controls for protecting it own private key material (The importance of this latter bit of implicit knowledge should not be overlooked).

While this trust model has been extremely successful in e-commerce scenarios, its adoption has been relatively spotty in messaging and email scenarios for most common users.

An alternative “web-of-trust” model has been proposed by the PGP community, whereby trust in a public key is based on a graph of trusted signatures, where trust in any one node is bootstrapped by an out-of-band trust decision, such as a a key-signing “party.” Thus, Alice may trust Claire transitively through Bob, because i) she trust Bob’s public key is in fact Bob’s through an out-of-band agreement, and ii) Bob signed Claire’s key. Widespread adoption of this model is of course dependent on widespread participation by all parties in a potential trust decision, which to date has only occurred in limited communities (e.g., the Apache release engineering community).

The system proposed in this paper has no single solution to the problem of key distribution and trust, though several models are feasible.

First, because key material is transient in this system, out-of-band communication of key material is not entirely out of the question. For example, if Alice and Bob want to

initiate a private communication session, they could choose a temporary pass word or pass phrase, and use the Web client to generate a transient key pair. The public component of this key pair could then be shared out of band (e.g., via email). While not a completely secure solution, in that plaintext email can be easily inspected and tampered with, this trust model could work in scenarios where the private information is low value, or where the parties have sufficient trust in the out-of-band communication channel (e.g., a dark alley).

Another possibility is that the web messaging service itself (or the service at which it is hosted) could server as the trust broker for key material. For example, both Alice and Bob could publish their public keys to the web service, from which they could then be retrieved. Consequently, their trust in each other's key material would be predicated on their trust in the web service, which of course they are already using for trust in the integrity of the code used to perform cryptographic operations.

In some ways, this latter trust model is weaker than the out-of-band key exchange, given its level of trust on a third party, though as with most trust decisions, the degree of trust in a security system is generally weighed against the risk that a piece of information will be disclosed, together with a measure of the value of that information. For relatively lowly valued information, the the risk of disclosure may be presumed to be low, and hence the trust in the system high.

We envision a set of services that supports full disclosure of technologies, in terms of making the technologies source-available, making them deployable independently of the provider, and providing a rich set of documentation and description of the security technologies involved and their associated risks; education is as much a part of this project, as actual running code and services.

More may be said in future versions of this paper about the business model for this application.