Selectable Confidentiality Using WSEmail

Kevin Lux
University of Pennsylvania

June 2005

Abstract

WSEmail is a system of web services that aims to show how email and messaging, in general, could be securely redone within a SOAP/XML paradigm. It provides an open and extensible framework that can be used for a variety of messaging applications. Research projects are underway that explore increasing the functionality of WSEmail, but one area that is being neglected is encryption. WSEmail lacks message encryption, but perhaps more importantly, it lacks a conceptual design for how such an enhancement could be added to the system. This paper will explore a design that will enable WSEmail to encrypt messages in a sensible fashion, while leveraging current architecture and design principles. It will also explain how the proposed design can be integrated with other work being done on WSEmail, particularly with regard to policy work being conducted at the Illinois Security Lab.

Introduction

Since large, distributed messaging systems have been deployed, users have generally looked for ways to achieve confidentiality in their correspondence. Systems such as PGP and S/MIME have been created to attempt to solve the problem but are usually much too difficult for the average user to understand and use. Confidentially is therefore relegated to technical people who have the means to diagnose obtuse errors and strange interactions between mail clients and servers. Unfortunately, a variety of incompatible systems currently exist. Even if a power user is able to properly set up his client software and locate a similar correspondent, they may still be unable to communicate because they are using software from competing vendors. Perhaps most irritating is that neither user will know of the problem until they receive an unencrypted message from the other, detailing how their software claims the message is invalid or can't be decrypted.

This problem has plagued traditional email for years and shows no signs of being cured, mostly because of the relatively inextensible nature of current email systems. WSEmail, on the other hand, was developed to be as open and extensible as possible, allowing new functionality to be easily and dynamically added to clients and servers. This platform can allow the design of protocols that can alleviate many of the problems faced in trying to deploy wide-spread encryption in current email systems. Perhaps more interesting is how WSEmail could make the process appear transparent to the user, requiring very little interaction on their part. The majority of the work could be automatically identified and any parameters distilled dynamically based on the sender, intermediate servers and the recipient.

Selectable confidentiality is an approach that attempts to add automatic encryption extensions to WSEmail in an end-user friendly way. The central idea is that encryption mechanisms are pre-negotiated prior to the sending of a message. The client, at the time of sending, can be reasonably assured that the message will be deliverable by all intermediate server nodes and ultimately received by the final recipient. The recipient will have the ability or be willing to acquire the ability to understand the message, decrypt the contents and display it. In support of this main goal are a collection of sub goals, most of which are dedicated to simplifying the user experience.

This paper is divided into several sections. First we will explore the goals of selectable encryption and any sub goals that are required. Each goal will then be explored in more detail. Next we will run though a sample scenario to see how selectable confidentiality would affect the delivery of encrypted mail. Possible application spaces will be explored followed by a brief discussion on how the current WSEmail project would be impacted by adding selectable confidentiality. The paper will then conclude with some closing thoughts.

Goals of Selectable Confidentiality

The general aim of selectable confidentiality is provide a rather transparent application of encryption to messaging in WSEmail, while maintaining a simple end-user experience. This overarching goal can be broken into several requirements:

- A method for determining what encryption mechanisms are available, usable or able to be acquired along the delivery path.
- A design that allows encrypted messages to be transferred in an extensible and efficient manner.
- A method for distributing, acquiring and maintaining cryptographic functionality in servers and clients.

The goals of selectable confidentiality are relatively straight forward and somewhat obvious. The difficultly arises in the designs for these goals and in creating an implementation that will allow selectable encryption to grow as new features are added to WSEmail. Each goal is also somewhat orthogonal to the others. This should allow refinements on individual goals to not interfere with the others and be used as an opportunity to create implementations that can be swapped out. Changeable implementations of the various goals will keep with the WSEmail spirit of embracing extensibility.

Goal 1: Determining Mechanism Availability

Of significant importance in sending encrypted messages is some guarantee that the message can be delivered as it exists to the recipient and that said recipient is capable of making sense of the message. To successfully accomplish this goal, selectable confidentiality will need to somehow explore the delivery nodes in the path of a message and verify that they all converge on some common encryption mechanism or are willing to do so.

Ideally, the entire state of the delivery network would be known to the sender. In small deployments, this might even be the case. A server might simply refuse to deliver a message because it knows all the encryption mechanisms available to all clients and the message in question will be unrecognizable to the recipient. Realistically, however, WSEmail is aiming to be more of a widely distributed, multi-enterprise messaging platform. As such, it will be unwieldy, if not impossible for a WSEmail server to know all information about all servers. Also, when multiple enterprises are involved with message exchanges that have separate administrative domains, it can be expected that information regarding configurations can change quickly and without other groups knowing. By the same token, however, constantly querying for current configurations will waste network resources.

A solution is to have entities within the delivery network publish policies describing encryption mechanisms they will accept. Policies should be highly structured so that servers querying for information are able to select the best options for sending messages that will be accepted. Conversely, if a path cannot be created from the sender to the recipient, an analysis can be conducted which will yield the node or nodes that prevent the transfer. A server may wish to reroute the message or decide that it is unable to do so because it cannot control the route a message takes after a certain hop. The client might also choose to encapsulate the message in another format that will be accepted by all the nodes. Such encapsulation might produce a message that is inefficiently encoded but deliverable.

Included within the policy should be a TTL (time to live) of some kind. This will allow servers to cache information about other servers, similar to the way DNS functions. Caching will allow for more efficient querying and prevent overburdening of servers simply for policy updates. Assuming a sufficient TTL, senders will be able cache resultant policies and, assuming an intelligent enough policy collection algorithm, only ask for parts of policies as they become outdated.

The most obvious format for a structured encryption policy in WSEmail is an XML document. The description of an actual schema is beyond the scope of this paper. A sample policy document from a server is shown below, but should be taken as an example that needs a lot of scrutiny and careful design.

```
<EncryptionPolicy>
        <MyPolicy>
                <Entity name="server1" ttl="1 week" role="destination_server">
                        <Reject>
                                <Mechanism name="FlawedMechanism" version="*">
                                        <Reason>It is broken.</Reason>
                                        <SuggestedAlternate name="GoodMechanism" version="3.3" />
                                        <SuggestedAlternate publisher="GoodPublisher" />
                                </Mechanism>
                        </Reject>
                        <Accept>
                                <Mechanism name="*" />
                        </Accept>
                        <Administrators>
                                <Administrator name="Kevin">kevin@server1</Administrator>

                        </Administrators>
                </Entity>
        </Current>
        <DownStream>
                <Entity name="recipient@server1" ttl="1 hour" role="recipient">
                        <Reject>
                                <Sender name="badman@server2" />
                                <Mechanism name="Apple.OSX.Encryption">
                                        <Reason>I do not have an Apple.</Reason>
                                        <SuggestedAlternate name="Linux.Encryptor" version="2+" />
                                </Mechanism>
                        </Reject>
                </Entity>
        </DownStream>
</EncryptionPolicy>
```

The sample schema illustrates a few important points that warrant explanation. A downstream directive is included so that entities blocked by firewalls or that frequently offline are represented in determining the effective policy. It's primarily meant as a way for the recipient to have policy published and respected, while not requiring the client to be constantly active in policy discovery. The schema also represents nodes in the delivery path as entities that play a particular role. It is not unreasonable to assume that servers may have different policy, depending upon whether they are the destination server or simply a relay.

To determine the effective encryption mechanism to be used, the client or the client's server will query servers along the delivery path (to the extent possible), determining which particular policies are in effect. These incremental queries may result in more queries by the contacted servers to determine what policies their peers enforce. The initial queries by the client may be thought of as a recursive policy query that should obtain all the policy fragments of the delivery path servers and the recipient. These parts will then be combined in to one large policy document. A reduction of the policy will occur, eliminating delivery options due to restrictions by servers in the path. In the event of an undeliverable message, the suggested alternates should be evaluated. Although not present in the sample, it might also be the case that message encapsulation is considered to tunnel through servers that are too restrictive.

Goal 2: Encrypted Message Transfer

WSEmail will need a method to put encrypted messages out on the wire. The biggest problem is that, at present, WSEmail is not built to easily support this change. The way messages are handled in WSEmail will need to be completely overhauled, causing cascading changes in the entire system. Due to the size of the change, it seems an excellent opportunity to reengineer parts of WSEmail to be more robust. Following is a more structured design that will support selectable confidentiality and provide a rather extensible framework to support other message transformations. Before delving in to the specific design features for selectable confidentiality, it will be necessary to describe a few foundational elements of traditional email and their counterparts in WSEmail.

SMTP uses a logical “envelope” to hold information such as sender and recipient lists. This envelope information is maintained in addition to the message by the MUAs and MTAs. At the final destination, the envelope is used to determine the ultimate recipient and then discarded. This works well for the SMTP servers; they are able to deliver messages using only the envelope information and do not have to be concerned with the message contents. In fact, the message contents can be an encrypted mess (within limit) and still be delivered without problem since delivery information is based entirely on the envelope.

WSEmail, on the other hand, has no concept of envelopes. The WSEmail message is a list of properties relevant to an email message such as subject, date, recipient, etc. Mail is routed using these internal fields and some semantics such as the entity that signed the message and how the message was received by the server. This poses a serious problem for message-level encryption as it limits the fields that can be encrypted because some fields are needed to simply route the message.

I propose a rather dramatic change to the way WSEmail routes and delivers mail. WSEmail should adopt the concept of an envelope. This can be done in several ways. The easiest and perhaps most approachable would be to send along an XML representation of an envelope with the WSEmail message to deliver. The server can then process the message using the envelope, throwing out the envelope, forwarding it with the message to another server or using it for error handling. A more elegant (but perhaps unattainable solution depending upon WSE support) is to add the envelope to the routing information of the WSE request message. Routing information for WSEmail would live in the routing information for the SOAP request. Doing so would probably break the schema for WS-Routing/WS-Addressing thereby breaking cross-platform availability or creating the need for hacks. If, however, the WS standards allow the addition of routing information, this would be an ideal use.

As far as information content in the envelope, it might be useful to differentiate the use of a WSEmail message and the envelope. A WSEmail message is meant for a user/program. It should contain all the information necessary for the recipient to view or process the message. Gratuitous information that users generally do not care about should probably be reserved for the envelope. Information such as the path a message took to the recipient, which servers converted encoding types, which servers checked the message for SPAM, etc is generally useless and is usually hidden from the user when they view the message. Using the previously discussed differentiation, the envelope should be reserved for all information that would be of interest to mail administrators, servers in the delivery path, etc while the message should only contain information directly pertinent to viewing and acting upon the message.

With this distinction, it becomes a question of whether or not an end-user would have any interest at all in receiving the envelope. It is probably sufficient for the destination server to save a copy of the envelope for the user so that it is available, if necessary, but not to automatically assume the user needs or wants it. Access to envelope information can easily be accessed through an extension to the server. Rather than hiding information that is usually hidden, it seems much more sensible to not include it, but have a method for access, if needed.

In order for the servers to be able to reference and track messages between themselves, another requirement pops up. WSEmail needs to create a GUID (globally unique identification number) for each WSEmail message. Any large messaging system generally needs to be able to identify single messages from large groups for tracking, auditing and identification. With the addition of encryption and envelopes, WSEmail increases its need for such an identifier. At present, no attempt to create a GUID for each message is made. This is easily remedied by creating a random number and concatenating it with information such as the server name and time. The GUID should be placed in both the WSEmail message and its respective envelope so that they can be cross-referenced.

With all this groundwork set, it becomes necessary to discuss the changes to the actual WSEmail message objects. Presently there is no WSEmail message hierarchy. This was done for general simplicity, but is now showing its age. Part of WSEmail's "charm" was that it used a single message format that allowed the message to be a regular email, instant message, etc. Unfortunately, this inefficiently uses the message fields and will mean growing pains for WSEmail in the future unless the behavior is changed. For example, when a user sends an instant message, only the recipient and body of the message object fields are used. This creates extra overhead in the message that will never be used and means that any instant-messaging specific fields must also be added to all WSEmail messages.

An object hierarchy that supports inheritance and interfaces will allow new message types to be represented in an efficient manner, while allowing the same usability that is in WSEmail now and adding the ability to encrypt arbitrary message types. The WSEmail message object as it exists now and processing infrastructure will be broken into several pieces:

- WSEmailBaseMessage – abstract class. Contains the basics of all messages along with a reference to a HandlerInformation object.
- WSEmailMessage – class. Inherits off WSEmailBaseMessage and adds extra fields for typical email messages.
- WSEmailInstantMessage – class. Inherits off WSEmailBaseMessage and adds fields for typical instant messages.
- HandlerInfo – class. Contains information that will describe how to handle the message
- HandlerCollection – class. Contains all the handlers the client has to handle messages.
- Handler – abstract class. Contains the basic information needed to handle a message (version, type, etc) and register the existence of the handler.
- *Particular*Handler – class. Contains the specific logic to handle the processing of a *particular* type of message. Receives the message and performs any transformations on it, as required.

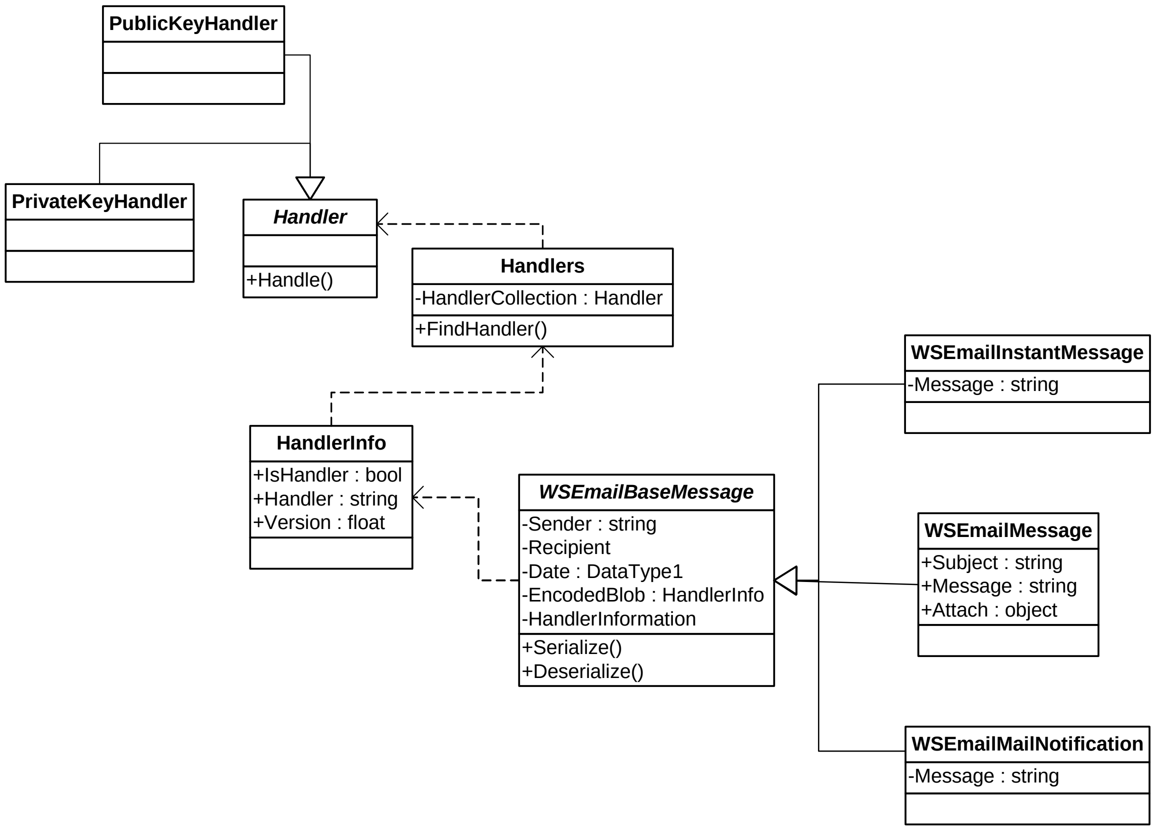

Figure 1 - Proposed design.

The WSEmailBaseMessage object would contain the most basic parts of the message structure, which could then be inherited by more specific message types. The subclasses can then add more specific fields tailored to the object's function. Examples of fields included in the base object would include the sender, recipient list, date and GUID. In addition, the base object should also contain should be the sender, recipient and date. All other information should be included in derived classes. For example, the present WSEmailMessage would inherit the base message type and add a subject, message and attachments.

In addition to the most basic message parts in the WSEmailBaseMessage would be the addition of a HandlerInfo object and a blob. The blob is used to hold the message data if it is encoded in any way. It would be messy and complicated for the application programmer to have to manipulate encoded messages field by field and information may be inferred simply by the presence or absence of fields. By moving all encoded data to a blob, no information is leaked to potential observers. This provides the handlers with one data source to deal with and should allow more efficient operation.

All further refined objects inherit this basic message model and add new fields and logic, creating richer and more specialized messages. This inheritability allows for more refined message types to be created and builds messages into a hierarchy. Having the messages adhere to a particular structured hierarchy presents opportunities to route messages based on their type, which could allow for interesting applications in the future, as long as developers can agree on what that model looks like.

The astute observer will notice the presence of ‘handlers’ and not ‘decryptors’. This is an intentional refinement on the proposed infrastructure. By handling messages, we open WSEmail up to an entirely new message processing method that allows limitless functionality. Since encoded messages are handled in a generic way and transferred as a blob of data, a variety of manipulations can now occur while the message is being “handled”. A message can now be simply decompressed, decrypted or may even be subject to testing prior to display. Such tests might include checks for digital rights management, verifying the security of the recipient’s environment prior to message decryption, using federated identities to check for newer versions of the message contents, etc. The amount of new functionality possible with such a message processing available client side is staggering and will allow next generation applications.

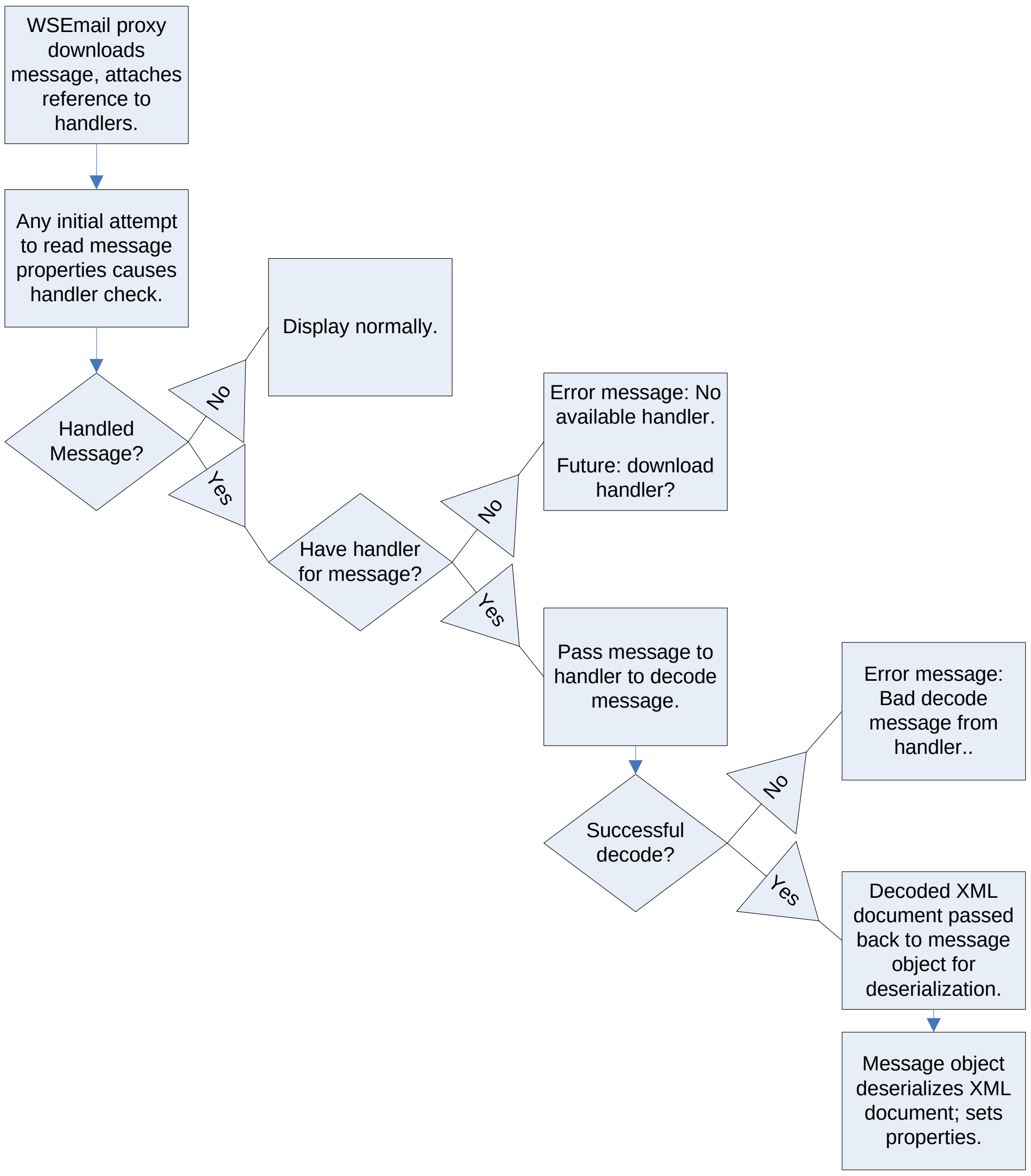
WSEmail proxy downloads message, attaches reference to handlers.
Any initial attempt to read message properties causes handler check.
Handled Message?
No
Display normally.
Yes
Have handler for message?
No
Error message: No available handler.
Future: download handler?
Yes
Pass message to handler to decode message.
Successful decode?
No
Error message: Bad decode message from handler..
Yes
Decoded XML document passed back to message object for deserialization.
Message object deserializes XML document; sets properties.

Figure 2 - Proposed data flow.

Goal 3: Client/Server Side Support Features

A client-side infrastructure needs to be built to support the handling of messages as discussed in the previous goal. The plumbing between messages coming from servers and fed to handlers would be included in this goal as well as all the user plugins that will be needed. Selectable confidentiality will create the need for more processing to be done client side, so that confidentiality can be maintained. A very obvious example of such processing is the generation of keys. Clients may not trust the server to create their encryption keys. Keys could be generated on the client machine using plugins and used for encryption, without the server being involved. Clients could also store encrypted key material on the server so that keys can be shared between various client locations. All this information will need to be managed and configured, driving home the point for a client plugin interface.

The WSEmail client already has support for plugins. These plugins have very limited access to the client, can only be run in certain instances and are not able to reference each other. A more developed plugin interface is needed. Plugins should be loadable at the startup of the client so that they can function as message handlers and policy negotiators. They also need to be tightly integrated in to the GUI (graphical user interface) so that a user could, for example, select a "manage" option, view a list of all installed crypto plugins and then manage one.

The plugins need to be easily updatable and preferably dynamically loadable. If the past is any indicator of the future, most major vendors will have their own cryptographic implementations, requiring clients to have many different versions of the same algorithm to be able to interact with other users on different platforms. If a client does not have a particular message handler, information should be included in the HandlerInformation structure (see goal #2) that details where the plugin may be acquired. The client application will likely apply system policy and ask the user to confirm the acquisition of new plugins, but the process should be relatively streamlined. It is of dire importance that the plugin acquisition system is highly scrutinized as it would be a very obvious attack vector for phishing and spyware.

The most simplistic design would be to create a directory that contains library files that can be loadable as plugins. This will quickly become unwieldy due to versioning conflicts and security. A more robust, versioning-capable system that supports digital publishing certificates that can attest to the security of the plugins is necessary. The solution to this problem will likely be platform dependent. Developers on the .Net framework can make use of strong names and the global assembly cache, Microsoft's solution for versioning problems in .Net. Further thought on secure versioning, or perhaps a platform independent system will be required.

Example Scenario

With a sketch of selectable confidentiality now laid out, it might be prudent to briefly run though message delivery on a hypothetical but somewhat realistic network infrastructure. The interactions between goals of selectable confidentiality will be highlighted, particularly as to how they make the entire system work.

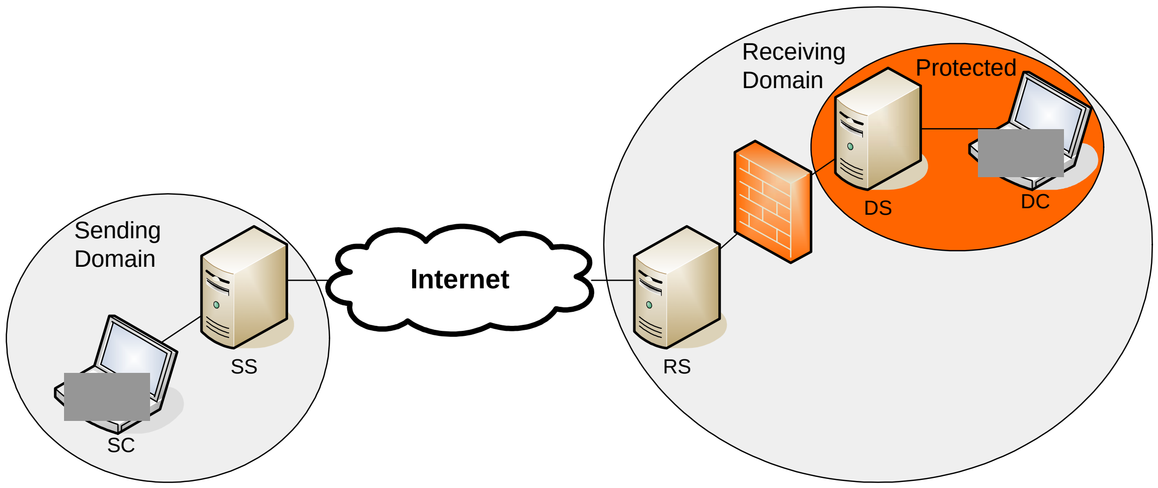

Figure 3 - Sample infrastructure.

Consider a network infrastructure as depicted. In this instance, SC (Sender Client) is trying to send a message to DC (Destination Client). The message passes through SC's local server, SS (Sender Server), traverses the internet and arrives at RS (Relay Server), located in a DMZ (demilitarized zone) for the recipient's domain. RS is allowed to communicate with DS (Destination Server) and DC (Destination Client) though the firewall. Assuming no fore knowledge, SC will have no possible method to know what encryption mechanisms DC supports. Furthermore, SC will not be able be to directly ask DC what it supports because the firewall will block communication attempts.

In a conventional messaging medium, SC would have to package up the message using its best guess for what mechanisms DC supports. The message may or not be delivered and may or may not be decipherable by DC.

Under selectable confidentiality, end users would push their policies to their local servers. SS would therefore contain SC's policy and DS would contain DC's. To begin the delivery process, SC or SS would begin by querying the next hop closest to DC for policy, in this case RS. RS would then query DS. This query is allowed because of the network architecture of the receiving domain. Also, since DS has DC's policy, RS now has a complete picture of the delivery path, as it relates to itself. RS will combine the policies of itself, DS and DC and return it to the original query from the sending domain. The returned policy will then be evaluated by sender to determine what encryption mechanisms can be supported over the entire delivery path. The sender can be reasonably assured that DC can understand the message.

Of course, if DC and SC have no common mechanisms, a second level of policy evaluation can occur. SC can determine is DC would be willing to acquire a new mechanism that SC currently has or vice versa. The acquisition of new mechanisms is the third goal of selectable confidentiality.

After finally reaching a conclusion of which mechanism to use to encrypt the message, SC can perform the transformation and actually send the message. The actual packaging of the message is the second goal of selectable confidentiality.

Upon retrieving the message from DS, DC will attach a reference to the message so that the message can access the message handlers. If the message is to be viewed or acted upon, the appropriate handler will be found and the message passed to it for transformation. If the handler is not installed, a routine will attempt to acquire and install it, depending upon local security policy. After the transformation is complete on the message, it will be returned to the client application and the normal flow of processing will continue. At this point, the workflow associated with selectable confidentiality is complete.

Possible Application Spaces

Selectable confidentiality has several application spaces outside of the obvious one of determining a shared encryption mechanism. But, since the aforementioned space is the major application, we should explore an industry that could use it.

The military would have uses for selectable confidentiality. Given that they tend to have very strict rules about communication channel use and the format in which data may travel over channels, there seems to be a possibility for the direct application of selectable confidentiality. Also, considering the nature of warfare, that is that information networks must be quickly built and functional, a system that can negotiate a common denominator between all the nodes or permit nodes to update their functional status automatically could be very useful. More work on selectable confidentiality might have to be done to support signatures on the policies (this was not addressed), such that policies can be authenticated.

Taking a step back, selectable confidentiality can almost be viewed as an integrated document conversion system. Throughout this paper, it has generally been looked at under the guise of encrypting and decrypting, but those operations could just as easily be other transformations such as converting a proprietary document format to an open format. In this regard, selectable confidentiality would act in a way very similar to the MIME accept extension in HTTP requests. Receivers would be able to specify that they only want documents of a certain kind and that they might be willing to acquire new plugins to view other documents. The sender could either convert the document to another format or expect the recipient to get the viewer plugin. Assuming a standard set of plugins are available, selectable confidentiality could eliminate a common problem businesses face, the problem of “What version of <product> is this document written in, I can’t view it!”, automatically.

Impact on Current Implementation

Included with many of the suggested changes were notes about the impact. The largest impact based on the sum of the changes is the annihilation of the basic message format. Such a change will obviously break multitudes of subcomponents in WSEmail, render it backwards incompatible with previous versions and create a reengineering nightmare for the programmer implementing the change. Incremental changes can be carried out, but ultimately the base message type will need to be changed to support increased functionality of the system as a whole. Since the system is not used in production anywhere, WSEmail is still able to make these changes and not have to worry about backward compatibility. The WSEmail team members should take advantage of this time to make radical changes while they can still be made without affecting users of the system.

Performance characteristics should not be changed much. The message sizes will be slightly larger with the inclusion of the envelope. Unfortunately, the entire message will still have to be deserialized (as no partial deserialization performance enhancements exist). Given the relatively small increase in bytes per message the overall change in processing time should be negligible. Delivery time will increase for messages that require policy to be collected and analyzed. The increased time will obviously be proportional to the size of the delivery path.

Time estimates for such a change are hard to pin down. For a programmer intimately familiar with the current codebase of WSEmail, I'd guess that all the changes would take at least one month of full-time work, mostly because of the debugging and testing required to validate the change. For a programmer not familiar with the codebase, the changes could easily drag on for months or ultimately never be completed. The changes require a rather indepth knowledge of.Net, web services, XML and the WSE or several months to learn the basics and then investigate the codebase.

Conclusion

Selectable confidentiality has the ability to provide a robust solution to problems that have plagued email systems for years. It can determine shared functionality among all participants in a delivery path or attempt to optimize how many nodes need to acquire new plugins. Encrypted messages are transferred in such a way that they could be transformed in different ways such as compression or with rights management. Finally, all the functionality is embedded and easy to access in the client.

Interesting applications that could use selectable confidentiality exist now and it's probable that new applications will be discovered after it is available in WSEmail. The system as a whole creates a content negotiation system that is integrated with the messaging platform, while still being widely deployable. That's a lot of power and significantly trumps the functionality of current email.

All these benefits do come at a cost. Significant development time will be required to implement parts of selectable confidentiality, with the entire system requiring a significant investment. There is also a large amount of design work still left to do to fully support all of the goals. A lot of time must also be spent on designing the policy schemas to ensure they are sensible and extensible.

Careful planning and implementation are required, but selectable confidentiality could be a "killer app" for the WSEmail platform and spur on adoption.