

Hushmail Express

Password Encryption in Hushmail

Brian Smith
Hush Communications

Introduction	2
Goals	2
Summary	2
Detailed Description	4
Message Composition	4
Message Delivery	4
Message Retrieval	5
Default Authentication Scenario.....	5
Symmetric Key Generation	5
Auth Info XML	6
The Purpose of the Answer Hash	6
The Purpose of the Answer Salt	6
The Purpose of the Extra Hash Step.....	6
Security	7
Hushmail Express and the Hush Encryption Engine	7
Email Attachments.....	7
Hushmail Express versus Public Key Encryption	7
References.....	9

Introduction

Hushmail, the world's most popular encrypted web-based email service, utilizes a public key encryption system to encrypt email messages from sender to recipient. A classic problem of public key encryption systems is that a public key for the recipient must be available to the sender within the system. Thus, until this time, it has not been possible for Hushmail users to send an encrypted email to a recipient unless a public key for the recipient has been placed on the Hush Key Server.

To address this problem, we introduce Hushmail Express. Hushmail Express is a scheme for allowing Hushmail users to communicate with any email recipient on the Internet without requiring that the email body or attachments be transmitted over any network without encryption.

Goals

Hushmail Express has been designed to fulfill the following requirements.

- It must be possible to send a Hushmail Express message to any email address, unless that email address already is associated with a public key on the Hush Key Server.
- The recipient must need only an Internet connection and a web browser supporting SSL, JavaScript, and frames to retrieve the message. Java, which is required for the Hushmail web-based email, should not be required.
- The message and associated attachments must not be stored in persistent storage.
- The key that will decrypt the message must not be stored in any persistent storage.
- The processes of sending and retrieving secure messages must be simple and unobtrusive.

Persistent storage refers to non-temporary storage of information. In some cases, information may be cached in memory or on disk for brief periods of time,

Summary

When utilizing Hushmail Express, the sender chooses a question when sending the message and the recipient must answer the question correctly in order to retrieve the message.

Hushmail Express leverages two technologies. First, it utilizes the symmetric encryption that is part of the OpenPGP message format. This encryption uses the answer to the given question as a symmetric key, and protects the message when it is in storage on the Hushmail servers.

Second, a system comprising a database and website is utilized to facilitate message retrieval. The symmetrically encrypted message is not sent to the recipient directly, but stored in a database. A notification message is sent to the recipient, containing a link to a website where the message can be retrieved if the question is answered correctly in a limited number of attempts. This is necessary for two reasons: there is no guarantee that the recipient will have the facility to decrypt an OpenPGP message, and so it should

be decrypted on the web server and returned protected by SSL to the recipient web browser; and it is also likely that a simple answer to a question will be vulnerable to dictionary and brute force attacks, so the number of decryption attempts must be limited by an intermediary.

Using the system described, the process of sending an encrypted message to a recipient without a public key requires one step more than sending an encrypted message. That is the step of choosing a question and answer to associated with the message.

The process of retrieving a message encrypted in this way requires two more steps than receiving an unencrypted message. The recipient must a) click on a URL in the notification message, and b) enter the correct answer to the question chosen by the recipient.

Detailed Description

A Hushmail Express message must be composed in a Hushmail-enabled email client, and delivered via a Hushmail email router.

All messages are RFC822 email messages that may include MIME-formatted attachments, and all encryption occurs according to the OpenPGP standard.

Message Composition

During the process of composing an email message, one or more recipients must be chosen. Based on those recipients, the encryption method for the message will be determined. The Hush Key Server is queried to see if public keys are available for all recipients. If there are public keys available for all recipients, the message is encrypted using those public keys, and sent without further user interaction.

If there are one or more recipients for whom no public keys are available, the message will be encrypted with a password in addition to the public keys of the recipients for whom public keys are available. This password could be derived in any number of ways, but the default mechanism is to derive it from the answer to a question the answer to which should be known only to the recipients of the message. We will now say that the message is password encrypted.

If a message is password encrypted, two special email headers are added to the RFC822 formatted message before it is sent out.

```
X-hush-password-message-auth: <auth info>
X-hush-password-message-recipients: <password message recipient list>
```

The auth info is a base-64 encoded block containing an XML document which defines the means by which the password can be derived. For example, in the default scheme this will include the question to which the password is the answer.

The password message recipient list is a comma separated list of RFC822 email addresses indicating which of the recipients of the email are password message recipients (as opposed to recipients with public keys, to whom the message was public key encrypted).

The message is then delivered by the normal SMTP routing mechanism.

Message Delivery

An SMTP server on the Hush network will detect any message with “X-hush-password-message-auth” and “X-hush-password-message-recipients” headers. The password message recipient list and auth info are extracted from those headers, and the headers are stripped from the message.

For any recipient who is not on the password message recipient list, the recipient is assumed to have a public key, so the message is sent on to that recipient without further alteration.

For any recipient who is on the password message recipient list, a unique message identifier is generated. The entire email message, along with the extracted auth info and the email address of the current recipient are stored in a database. The email message is then replaced with an email message containing the unique message identifier in a URL, and that email message is sent on to the recipient.

Message Retrieval

The recipient of a password encrypted message does not receive the original message, but rather a message containing a URL with a unique message identifier. Upon following that URL, the recipient is directed to a website, where the original message can be retrieved. This is done by accessing the original message in a database based on the unique message identifier. In order to view the content of the message, the recipient must then follow steps to generate the password that will decrypt the message. In the default scenario, that means that the user must provide the correct answer to the question specified by the sender. Upon accessing the message, the user also has the option to retrieve attachments, delete the message, reply to the message, or print the message.

The number of successful and failed attempts to retrieve a message may be limited. If either limit is exceeded, no further attempts to access the message will be allowed.

Default Authentication Scenario

As described above, the auth info XML defines the means by which the symmetric encryption key is derived from user information. In the future, this key generation may be extended to process variables such as the recipient's IP address and user input to other queries. However, the default authentication scenario involves the generation of a symmetric encryption key using the single user input of the answer to a question as follows.

It is expected that all user input be in the form of UTF-8 encoded strings.

Symmetric Key Generation

The user supplies the answer to a chosen question. That answer is canonicalized to a simplified form to decrease the likelihood of a conflict between the sender's interpretation of the answer and the recipient's interpretation. This process consists first of lowercasing the answer according to the Unicode 4.0 standard, and then removing all whitespace, commas, periods, and hyphens. All other characters are untouched. The only whitespace of concern should be the space characters, as the interface should prevent the user from inserting other whitespace such as newlines, carriage returns, and tabs.

A random string of characters called the "answer salt" followed by a colon is prepended to the answer. The result is then hashed using the SHA256 algorithm and the result is encoded as a hexadecimal string, high nybble first. That hexadecimal string is used as the password input to the OpenPGP symmetric encryption function where it is processed using string-to-key conversion as described in RFC2440 3.7, which creates the key to be used for the symmetric algorithm that will encrypt or decrypt the message.

Note that there may be other layers of processing at this point as specified in the OpenPGP standard. For example, the symmetric key described may not encrypt the message directly as it may encrypt a symmetric key packet containing another

symmetric key that encrypts the actual message. The details of that processing are not important here beyond stating that they follow the OpenPGP specification.

Auth Info XML

The default XML document containing the “auth info” required to generate a password to decrypt a message follows this format.

```
<?xml version="1.0"?>
<authInfo version="1.0">
<question>Question here</question>
<answerSalt>Answer salt here</answerSalt>
<answerHash>Answer hash here</answerHash>
</authInfo>
```

The question and answer salt are the values described above. The answer hash is generated by hashing the hexadecimal string used as the password above using the SHA256 algorithm and encoding that result as a hexadecimal string, high nybble first.

The Purpose of the Answer Hash

The answer hash is used to check that the answer to the question is correct without making an attempt to retrieve and decrypt the message, which may be several megabytes in size.

The Purpose of the Answer Salt

The default authentication scenario prepends a random string of characters to the canonicalized answer before encryption. This is necessary because a single Open PGP encrypted message may be delivered to external recipients who have public keys. Those recipients will also receive the symmetrically encrypted session key packets encrypted by the password encryption and intended for the password message recipients. Without the answer salt, dictionary attacks could easily be launched on those packets, and since in many cases the answer will be one or two dictionary words, the message could easily be decrypted. The addition of the random answer salt, which is stripped from the message header with the rest of the auth info before the message leaves the Hush network, ensures that the symmetrically encrypted session key packets on the message are protected with a level of security equivalent to the public key encrypted session key packets (see RFC2440 5.3 and 10.2).

Note that without the answer salt as stored in the auth info, it is impossible to decrypt the message even with the correct answer.

The Purpose of the Extra Hash Step

After the answer salt is prepended to the message, the result is hashed before being used as the input to the OpenPGP encryption process. This is so that in instances where it necessary to cache the key that will decrypt the message, the caching can be done without storing the actual answer, which may be private information with a value beyond the particular message. For example, once a recipient has decrypted a message for viewing on the website, it is desirable to cache the decryption key with the web session so that the recipient may retrieve attachments without having to re-enter the answer to the question. In this case, it is preferable not to store the actual answer to the question with the session, but rather to cache the hashed value, from which the answer cannot be derived.

Security

Hushmail Express and the Hush Encryption Engine

The Hush Encryption Engine is the encryption component utilized in all Hush technologies. In Hushmail, it is encapsulated in a Java applet, and all OpenPGP encryption operations occur within the user's web browser. When sending a Hushmail Express message using the Hushmail web-based email, the encryption occurs in the same fashion. A Hushmail Express message is symmetrically encrypted within the web browser by the Hush Encryption Engine, and the resulting OpenPGP message is uploaded over an SSL encrypted connection to the web server. Thus, while being transferred over the Internet, the message is protected by a layer of SSL encryption and a layer of OpenPGP encryption.

However, retrieval of a Hushmail Express message does not require Java. In this case, the Hush Encryption Engine is also used, but it is used on the web server. The message, as an OpenPGP message, is retrieved from a database and decrypted using the Hush Encryption Engine on the web server. The resulting plaintext is then passed through an SSL-encrypted connection to the browser. In this case there is only a single layer of SSL encryption protecting the message while it is transferred over the Internet.

Some years ago, export regulations commonly restricted SSL implementations to weak encryption, and so SSL alone was not considered sufficient to protect data transmitted to a browser. The situation is different now, as all modern browsers offer full-strength SSL. Thus for the purposes of Hushmail Express, retrieval of messages over an SSL-encrypted network connection offers an acceptable level of security.

Email Attachments

For simplicity, this document has referred to an email message and all attachments as a single entity. In reality this is not the case. An email message may have any number of attachments. In Hushmail Express, the main body of an email and each attachment are all encrypted as individual OpenPGP messages using the same key. In the case of the Hushmail web-based email, the attachments are uploaded to the web server before the email is actually sent. For that reason, the question and answer to be used must be specified upon the upload of the first attachment, and they cannot be changed unless the attachments are deleted. Attachments are *not* cached on disk in an unencrypted form.

Hushmail Express versus Public Key Encryption

The password encryption offered by Hushmail Express does not offer the same level of security as the end-to-end public key encryption from Hushmail user to Hushmail user. However, it is vastly superior to sending an email over the Internet with no encryption at all. It is not possible at this time to ensure that a public key is available for every email address on the Internet, and so it is necessary to devise other means for communicating with a level of privacy where necessary. In future incarnations of Hushmail Express, a path will be made available for recipients of Hushmail Express messages to generate a public key, thus eliminating the necessity to specify a question and answer for future emails.

More than anything else, though, the security of Hushmail Express depends, like Hushmail, upon the strength of the passphrase. In the case of Hushmail Express, the passphrase is the answer to the question. How complex that answer needs to be is something that is determined by the needs of the sender and recipient.

References

RFC2440 OpenPGP Message Format

<http://www.ietf.org/rfc/rfc2440.txt>

<http://www.ietf.org/internet-drafts/draft-ietf-openpgp-rfc2440bis-10.txt>

RFC2822 Internet Message Format

<http://www.ietf.org/rfc/rfc2822.txt>

FIPS PUB 180-2

<http://csrc.nist.gov/encryption/tkhash.html>