An Improved E-Mail Security Protocol

Bruce Schneier Chris Hall
Counterpane Systems
101 East Minnehaha Parkway
Minneapolis, MN 55419
{schneier,hall}@counterpane.com

Abstract

Current e-mail security systems base their security on the secrecy of the long-term private key. If this private key is ever compromised, an attacker can decrypt any messages—past, present, or future—encrypted with the corresponding public key. The system described in this paper uses short term private-key/public-key key pairs to reduce the magnitude of this vulnerability.

1. Introduction

E-Mail systems such as PGP [19, 16, 7, 17], PEM [11, 9, 2, 8, 16, 17], Entrust [5, 6], and S/MIME [14, 3] use hashed passphrases to protect the private key in a public-key encryption system. These systems were designed with the intention that the user generates a public/private key pair, and uses that pair for a long period of time. If an attacker manages to obtain the ling-term private key, then the security of the system is severely compromised; he can use it to decrypt and read all electronic communications encrypted with the corresponding public key, past, present, and future.

This sort of attack, while expensive, can be very costly to the victim. In this paper we focus on protocols which minimize the amount of information gained by an attacker in the private key. These are store-and-forward systems (e.g., as e-mail encryption programs), where the encryption is meant to protect a message in transit.

2. Points of Entry

Most popular e-mail encryption programs use a combination of public-key and private-key encryption. These systems use symmetric-key encryption, such as DES [12], IDEA [10], or Blowfish [15], to encrypt the messages and public-key encryption, such as RSA [13] or ElGamal [4], to encrypt the key. Details of the cryptography involved in these systems can be found in [18, 16, 17].

An attacker who wishes to decrypt a message has several "points of entry" they can use to try and decrypt the message:

(1) Guess or deduce the one-time symmetric encryption key.

(2) Mathematically derive the recipient's private key. For RSA, this means factoring the recipient's public modulus and using the information to deduce the private key.

(3) Obtain the recipient's passphrase which protects their private key. Then, use it to obtain the private key and thus decrypt the message. There are several ways to do this including guessing, monitoring keystrokes with a Trojan-horse program introduced into the recipient's computer, TEMPEST-based attacks, or even bribing the recipient to reveal his passphrase.

There are other attacks against the message: attacking the sender's computer through a variety of means, obtaining a copy of the message after it has been printed, convincing the receiver (or sender) to send an unencrypted copy of the message across an insecure network, etc. This research concentrates on attacks against the recipient's private key.

If we assume that a one-time symmetric encryption key is truly random (or at least independent of other one-time encryption keys), then the Attack (1) will not compromise the entire system. The attacker will only be able to use the encryption key that he obtains to decrypt the one message, but that will not help him to decrypt other messages. Thus, we are not really interested in modifying how current systems solve this problem.

Attacks (2) and (3) on the system represent the sort of global attacks mentioned above. If an attacker succeeds in either attack, then he can use the information hs obtains to decrypt all messages—past, present, and future—sent to the recipient. In addition, some systems have the added weakness that the attacker can use the information to forge signatures on the recipient's behalf. It is these "points of entry" which we focus on closing (or at least shrinking).

3. Notation

Before introducing protocols we must first introduce some notation.

Sophie Germain prime. A Sophie Germain prime of first order is a prime p such that $2p+1$ is also prime. For second order Sophie Germain primes, $2(2p+1)+1=4p+3$ is also prime and a similar pattern holds for general n-order primes.

$E_k(P)$**.** The encryption of the plaintext P with a symmetric encryption algorithm and key k.

$D_k(C)$**.** The decryption of the ciphertext C with a symmetric encryption algorithm and key k.

$GF(p^n)$ The Galois Field with p^n elements.

4. Multiple Encryption Keys

In RSA, knowledge of the private exponent is equivalent to knowledge of the factorization of the modulus [18]. Every message encrypted with the public exponent is vulnerable if the attacker gains knowledge of the private exponent. One solution to this problem is to use multiple short-lived public/private key pairs. Thus, a key pair could be valid for a short period of time, say a day, or valid for only one message. Then an attacker who deduces the private key for decrypting one message can decrypt a few other messages at best, i.e. only those other messages sent during the lifetime of the key pair.

For RSA, picking a new key pair means picking a new modulus in addition to picking new public/private exponents. This involves generating two large prime numbers, and can be a costly operation on most computers [16, 19]. Hence, RSA may not be suitable for this kind of e-mail encrytion scheme.

There are actually several possible solutions to short-lived public/private keys. In addition to using RSA with different exponents and moduli, one can also use a form of Diffie-Hellman key exchange first published in the SKIP protocol [1]. Consider the following protocol in which Alice publishes several public keys for people to use to send messages to her:

(1) Alice chooses a first-order Sophie Germain prime p and a primitive element g of the multiplicative group of the field $GF(2p+1)$.

(2) Alice chooses several random exponents $e_1,\ldots,e_k$ and computes $PK_i = g^{e_i} \pmod p$ for $i=1,\ldots,k$.

(3) Alice assigns a lifetime to each key so that at most two keys are valid at any point in time, and every key is valid for only a short period of time, say a day. Alternatively, the key could only be good for one message.

(4) Alice publishes the PK_i, lifetimes, p, and g as her public keys and stores the e_i as her private keys.

Note that in Step (2) Alice is essentially performing the first half of a Diffie-Hellman key exchange. Now suppose that Bob wishes to send a message to Alice. He performs the following steps:

(5) Bob selects a public key from Alice. If the keys have short lifetimes, he selects the one with the appropriate lifetime. If they keys are one-time keys, he selects the next key in the list (and then Alice's computer deletes it from the list).

(6) Bob chooses a random exponent e_b and computes $PK_b = g^{e_b} \pmod p$.

(7) Bob computes $k = PK_i^{e_b} \pmod p$ and uses the result as the private encryption key for the message he wishes to send.

(8) Bob encrypts his message M with the key k and sends $PK_i, PK_b, E_k(M)$ to Alice.

Once Alice receives the message she performs the following steps:

(9) Alice looks up the private key e_i that corresponds to the PK_i in the message sent to her by Bob.

(10) Alice checks the lifetime of the key against the time that Bob sent the message. If she doubts that the message was sent while the key was valid, then she can choose whether or not to decrypt the message. If she no longer has the decrypting key because she threw it away when it expired, then she can simply ask Bob to resend the message.

(11) She computes $k = PK_b^{e_i} \pmod p$ and uses the result to decrypt the message Bob sent her: $M = D_k(E_k(M))$.

If Alice wants to she can even choose a different Sophie Germain prime for each public key she publishes.

This kind of system provides a way to minimize the number of encrypted messages that are common to any public/private key pair. The fewer messages tied to a key pair, the fewer messages that can be decrypted if an attacker recovers the private key.

Once Alice has a way of generating multiple encryption keys she still needs a way of distributing them. After generating a list of one-time public encryption keys Alice can sign each of them with her long-term signature key and then upload them to a server. Ideally the server should accept network-based requests to obtain a public key for Alice. Once it receives such a request it should pick the current key from Alice's list, according to the lifetimes specified by Alice, and sends it to the recipient. When a key expires the server should throw away the key in order to conserve space.

5. Key Management

Whether a user has one private decryption key or many, key management is a difficult issue. Rather than forcing a user to memorize their private decryption key, current systems allow the user to chose a passphrase which the system then uses to encrypt the private decryption key. Then, whenever the user wants to decrypt a message sent to him, he uses his passphrase to decrypt the private decryption key and then decrypt the message.

5.1. Multiple Passphrases

The other attack we mentioned was guessing the user's passphrase. In most systems, the user's passphrase is not directly used to encrypt their private key. Rather, the passphrase is hashed with a salt and the resulting hash is used. Thus, there are actually two different attacks against the passphrase:

1. Guess the encryption key used to encrypt a particular private key. This amounts to guessing all or part of the result of hashing the actual passphrase with the salt.

2. Guess the actual passphrase. Then given any salt, the attacker can deduce the corresponding encryption key.

The former attack is clearly no harder than the latter because it is trivial to find the salted hash of the passphrase given the passphrase and salt value. However, if a cryptographic hash function is used, then the first attack is not equivalent to the second, i.e. an attacker cannot easily derive other salted hash values of the passphrase given one salted hash value. This suggests that we could encrypt multiple private keys with the same passphrase as long as we used a different salt for each exponent. The system would be more secure than if we used the same salt for each key.

While using different salts protects against the first attack, it does not protect against the second attack. An alternative approach is to use an entirely different passphrase for each private key. Since each public/private key pair is used to encrypt only one message, each message will be independent of every other message in an attack on the passphrase. If an attacker manages to guess a passphrase, then s/he can only use it to find one private key and hence decrypt only one message.

However, there is a difficulty with the latter suggestion. Each passphrase has to be remembered. For even a moderate number of public/private key pairs this can be a fairly daunting task. People are likely to forget some of their passphrases, or choose passphrases that are similar. The former has the disadvantage that someone would not be able to decrypt all messages sent to them, and the latter has the disadvantage that an attacker may be able to derive other passphrases with only a little bit of effort once they obtain one. A mix of the above two solutions seems to provide the best solution.

A user uses one passphrase to protect several private keys. For each key, they use a different salt. Since the salt and passphrase are used to generate an encryption key for each private key, the different salts protect the user against the first attack listed above. If the user minimizes the number of private keys they protect with each passphrase, then they also minimize the amount of damage done when an attacker guesses a passphrase. Since not every private key is protected by the same passphrase, the user gains partial protection against the second attack listed above.

If the system is designed properly, then the user could have to remember only one or two passphrases at any point in time. Each time the user generates a new set of public/private key pairs they can choose a new passphrase to protect the set. As the set of keys expire the user can throw them away until the entire set is empty. Since a user really only needs to generate a new set when their most of the keys from the previous one are about to expire, they only need to keep track of at most two sets of keys at any point in time.

6. Conclusion

As we point out in previous sections, current encryption programs have the common weakness that obtaining a user's private encryption key can cause extensive damage. We pointed out the various ways that an attacker can learn the private encryption key and also gave suggestions as to how to minimize the damage done by the respective attacks. In general, our suggestions amount to minimizing the amount of information that is protected by an encryption key. When combined with long term signature keys, a powerful e-mail security system can be written.

References

[1] A. Aziz, T. Markson, H. Prafullchandra, "Simple Key-Management for Internet Protocols (SKIP)," Internet-Draft, work in progress, August 1996.

[2] D. Balenson, “Privacy Enhancement for Internet Electronic Mail: Part III—Algorithms, Modes, and Identifiers,” RFC 1423, Feb 1993.
[3] S. Dusse, “S/MIME Message Specification: PKCS Security Services for MIME,” IETF Networking Group Internet Draft, Sep 1996.
[4] T. ElGamal, “A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,” *IEEE Transactions on Information Theory,* V. IT-31, n. 4, 1985, pp. 469–472.
[5] I. Curry, “Entrust Overview, Version 1.0,” Entrust Technologies, Oct. 96.
[6] P.C. van Ooorschot, ”Standards Supported by Entrust, Version 2.0,” Entrust Technologies, Dec 1996.
[7] S. Garfinkel, *PGP: Pretty Good Privacy*, O’Reilly & Associates, 1995.
[8] B.S. Kaliski, “Privacy Enhancement for Internet Electronic Mail: Part IV—Key Certificates and Related Services,” RFC 1424, Feb 1993.
[9] S.T Kent, “Privacy Enhancement for Internet Electronic Mail: Part II—Certificate Based Key Management,” RFC 1422, Feb 1993.
[10] X. Lai, J. Massey, and S. Murphy, “Markov Ciphers and Differential Cryptanalysis,” *Advances in Cryptology—CRYPTO ’91*, Springer-Verlag, 1991, pp. 17–38.
[11] J. Linn, “Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures,” RFC 1421, Feb 1993.
[12] National Bureau of Standards, NBS FIPS PUB 46, “Data Encryption Standard,” National Bureau of Standards, U.S. Department of Commerce, Jan 1977.
[13] R. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120–126.
[14] RSA Data Security, Inc., “S/MIME Implementation Guide Interoperability Profiles, Version 2,” S/MIME Editor, Draft, Oct 1996.
[15] B. Schneier, “Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish),” *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191-204.
[16] B. Schneier, *E-Mail Security,* John Wiley & Sons, 1995.
[17] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, 1996.
[18] D. Stinson, *Cryptography Theory and Practice*, CRC Press, 1995, pp. 138–145.
[19] P. Zimmermann, *The Official PGP User’s Guide*, MIT Press, 1995.