Network Working Group
Request for Comments: 4344
Category: Standards Track

M. Bellare
T. Kohno
UC San Diego
C. Namprempre
Thammasat University

January 2006

The Secure Shell (SSH) Transport Layer Encryption Modes

Status of This Memo

Copyright Notice

Abstract

Researchers have discovered that the authenticated encryption portion of the current SSH Transport Protocol is vulnerable to several attacks.

This document describes new symmetric encryption methods for the Secure Shell (SSH) Transport Protocol and gives specific recommendations on how frequently SSH implementations should rekey.

Table of Contents

1. Introduction

The symmetric portion of the SSH Transport Protocol was designed to provide both privacy and integrity of encapsulated data. Researchers ([DAI,BKN1,BKN2]) have, however, identified several security problems with the symmetric portion of the SSH Transport Protocol, as described in [RFC4253]. For example, the encryption mode specified in [RFC4253] is vulnerable to a chosen-plaintext privacy attack. Additionally, if not rekeyed frequently enough, the SSH Transport Protocol may leak information about payload data. This latter property is true regardless of what encryption mode is used.

In [BKN1,BKN2], Bellare, Kohno, and Namprempre show how to modify the symmetric portion of the SSH Transport Protocol so that it provably preserves privacy and integrity against chosen-plaintext, chosen-ciphertext, and reaction attacks. This document instantiates the recommendations described in [BKN1,BKN2].

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The used data types and terminology are specified in the architecture document [RFC4251].

The SSH Transport Protocol is specified in the transport document [RFC4253].

3. Rekeying

Section 9 of [RFC4253] suggests that SSH implementations rekey after every gigabyte of transmitted data. [RFC4253] does not, however, discuss all the problems that could arise if an SSH implementation does not rekey frequently enough. This section serves to strengthen the suggestion in [RFC4253] by giving firm upper bounds on the tolerable number of encryptions between rekeying operations. In Section 6, we discuss the motivation for these rekeying recommendations in more detail.

This section makes two recommendations. Informally, the first recommendation is intended to protect against possible information leakage through the MAC tag, and the second recommendation is intended to protect against possible information leakage through the block cipher. Note that, depending on the block length of the

underlying block cipher and the length of the encrypted packets, the first recommendation may supersede the second recommendation, or vice versa.

3.1. First Rekeying Recommendation

Because of possible information leakage through the MAC tag, SSH implementations SHOULD rekey at least once every 2**32 outgoing packets. More explicitly, after a key exchange, an SSH implementation SHOULD NOT send more than 2**32 packets before rekeying again.

SSH implementations SHOULD also attempt to rekey before receiving more than 2**32 packets since the last rekey operation. The preferred way to do this is to rekey after receiving more than 2**31 packets since the last rekey operation.

3.2. Second Rekeying Recommendation

Because of a birthday property of block ciphers and some modes of operation, implementations must be careful not to encrypt too many blocks with the same encryption key.

Let L be the block length (in bits) of an SSH encryption method's block cipher (e.g., 128 for AES). If L is at least 128, then, after rekeying, an SSH implementation SHOULD NOT encrypt more than 2**(L/4) blocks before rekeying again. If L is at least 128, then SSH implementations should also attempt to force a rekey before receiving more than 2**(L/4) blocks. If L is less than 128 (which is the case for older ciphers such as 3DES, Blowfish, CAST-128, and IDEA), then, although it may be too expensive to rekey every 2**(L/4) blocks, it is still advisable for SSH implementations to follow the original recommendation in [RFC4253]: rekey at least once for every gigabyte of transmitted data.

Note that if L is less than or equal to 128, then the recommendation in this subsection supersedes the recommendation in Section 3.1. If an SSH implementation uses a block cipher with a larger block size (e.g., Rijndael with 256-bit blocks), then the recommendations in Section 3.1 may supersede the recommendations in this subsection (depending on the lengths of the packets).

4. Encryption Modes

This document describes new encryption methods for use with the SSH Transport Protocol. These encryption methods are in addition to the encryption methods described in Section 6.3 of [RFC4253].

Recall from [RFC4253] that the encryption methods in each direction of an SSH connection MUST run independently of each other and that, when encryption is in effect, the packet length, padding length, payload, and padding fields of each packet MUST be encrypted with the chosen method. Further recall that the total length of the concatenation of the packet length, padding length, payload, and padding MUST be a multiple of the cipher's block size when the cipher's block size is greater than or equal to 8 bytes (which is the case for all of the following methods).

This document describes the following new methods:

aes128-ctr	RECOMMENDED	AES (Rijndael) in SDCTR mode, with 128-bit key
aes192-ctr	RECOMMENDED	AES with 192-bit key
aes256-ctr	RECOMMENDED	AES with 256-bit key
3des-ctr	RECOMMENDED	Three-key 3DES in SDCTR mode
blowfish-ctr	OPTIONAL	Blowfish in SDCTR mode
twofish128-ctr	OPTIONAL	Twofish in SDCTR mode, with 128-bit key
twofish192-ctr	OPTIONAL	Twofish with 192-bit key
twofish256-ctr	OPTIONAL	Twofish with 256-bit key
serpent128-ctr	OPTIONAL	Serpent in SDCTR mode, with 128-bit key
serpent192-ctr	OPTIONAL	Serpent with 192-bit key
serpent256-ctr	OPTIONAL	Serpent with 256-bit key
idea-ctr	OPTIONAL	IDEA in SDCTR mode
cast128-ctr	OPTIONAL	CAST-128 in SDCTR mode, with 128-bit key

The label <cipher>-ctr indicates that the block cipher <cipher> is to be used in "stateful-decryption counter" (SDCTR) mode. Let L be the block length of <cipher> in bits. In stateful-decryption counter mode, both the sender and the receiver maintain an internal L-bit counter X. The initial value of X should be the initial IV (as computed in Section 7.2 of [RFC4253]) interpreted as an L-bit unsigned integer in network-byte-order. If X=(2**L)-1, then "increment X" has the traditional semantics of "set X to 0." We use the notation <X> to mean "convert X to an L-bit string in network-byte-order." Naturally, implementations may differ in how the internal value X is stored. For example, implementations may store X as multiple unsigned 32-bit counters.

To encrypt a packet P=P1||P2||...||Pn (where P1, P2, ..., Pn are each blocks of length L), the encryptor first encrypts <X> with <cipher> to obtain a block B1. The block B1 is then XORed with P1 to generate the ciphertext block C1. The counter X is then incremented, and the process is repeated for each subsequent block in order to generate

the entire ciphertext C=C1||C2||...||Cn corresponding to the packet P. Note that the counter X is not included in the ciphertext. Also note that the keystream can be pre-computed and that encryption is parallelizable.

To decrypt a ciphertext C=C1||C2||...||Cn, the decryptor (who also maintains its own copy of X) first encrypts its copy of <X> with <cipher> to generate a block B1 and then XORs B1 to C1 to get P1. The decryptor then increments its copy of the counter X and repeats the above process for each block to obtain the plaintext packet P=P1||P2||...||Pn. As before, the keystream can be pre-computed, and decryption is parallelizable.

The "aes128-ctr" method uses AES (the Advanced Encryption Standard, formerly Rijndael) with 128-bit keys [AES]. The block size is 16 bytes.

> At this time, it appears likely that a future specification will promote aes128-ctr to be REQUIRED; implementation of this algorithm is very strongly encouraged.

The "aes192-ctr" method uses AES with 192-bit keys.

The "aes256-ctr" method uses AES with 256-bit keys.

The "3des-ctr" method uses three-key triple-DES (encrypt-decrypt-encrypt), where the first 8 bytes of the key are used for the first encryption, the next 8 bytes for the decryption, and the following 8 bytes for the final encryption. This requires 24 bytes of key data (of which 168 bits are actually used). The block size is 8 bytes. This algorithm is defined in [DES].

The "blowfish-ctr" method uses Blowfish with 256-bit keys [SCHNEIER]. The block size is 8 bytes. (Note that "blowfish-cbc" from [RFC4253] uses 128-bit keys.)

The "twofish128-ctr" method uses Twofish with 128-bit keys [TWOFISH]. The block size is 16 bytes.

The "twofish192-ctr" method uses Twofish with 192-bit keys.

The "twofish256-ctr" method uses Twofish with 256-bit keys.

The "serpent128-ctr" method uses the Serpent block cipher [SERPENT] with 128-bit keys. The block size is 16 bytes.

The "serpent192-ctr" method uses Serpent with 192-bit keys.

The "serpent256-ctr" method uses Serpent with 256-bit keys.

The "idea-ctr" method uses the IDEA cipher [SCHNEIER]. The block size is 8 bytes.

The "cast128-ctr" method uses the CAST-128 cipher with 128-bit keys [RFC2144]. The block size is 8 bytes.

5. IANA Considerations

The thirteen encryption algorithm names defined in Section 4 have been added to the Secure Shell Encryption Algorithm Name registry established by Section 4.11.1 of [RFC4250].

6. Security Considerations

This document describes additional encryption methods and recommendations for the SSH Transport Protocol [RFC4253]. [BKN1,BKN2] prove that if an SSH application incorporates the methods and recommendations described in this document, then the symmetric cryptographic portion of that application will resist a large class of privacy and integrity attacks.

This section is designed to help implementors understand the security-related motivations for, as well as possible consequences of deviating from, the methods and recommendations described in this document. Additional motivation and discussion, as well as proofs of security, appear in the research papers [BKN1,BKN2].

Please note that the notion of "prove" in the context of [BKN1,BKN2] is that of practice-oriented reductionist security: if an attacker is able to break the symmetric portion of the SSH Transport Protocol using a certain type of attack (e.g., a chosen-ciphertext attack), then the attacker will also be able to break one of the transport protocol's underlying components (e.g., the underlying block cipher or MAC). If we make the reasonable assumption that the underlying components (such as AES and HMAC-SHA1) are secure, then the attacker against the symmetric portion of the SSH protocol cannot be very successful (since otherwise there would be a contradiction). Please see [BKN1,BKN2] for details. In particular, attacks are not impossible, just extremely improbable (unless the building blocks, like AES, are insecure).

Note also that cryptography often plays only a small (but critical) role in an application's overall security. In the case of the SSH Transport Protocol, even though an application might implement the symmetric portion of the SSH protocol exactly as described in this document, the application may still be vulnerable to non-protocol-

based attacks (as an egregious example, an application might save cryptographic keys in cleartext to an unprotected file). Consequently, even though the methods described herein come with proofs of security, developers must still execute caution when developing applications that implement these methods.

6.1. Rekeying Considerations

Section 3 of this document makes two rekeying recommendations: (1) rekey at least once every 2**32 packets, and (2) rekey after a certain number of encrypted blocks (e.g., 2**(L/4) blocks if the block cipher's block length L is at least 128 bits). The motivations for recommendations (1) and (2) are different, and we consider each recommendation in turn. Briefly, (1) is designed to protect against information leakage through the SSH protocol's underlying MAC, and (2) is designed to protect against information leakage through the SSH protocol's underlying encryption scheme. Please note that, depending on the encryption method's block length L and the number of blocks encrypted per packet, recommendation (1) may supersede recommendation (2) or vice versa.

Recommendation (1) states that SSH implementations should rekey at least once every 2**32 packets. If more than 2**32 packets are encrypted and MACed by the SSH Transport Protocol between rekeyings, then the SSH Transport Protocol may become vulnerable to replay and re-ordering attacks. This means that an adversary may be able to convince the receiver to accept the same message more than once or to accept messages out of order. Additionally, the underlying MAC may begin to leak information about the protocol's payload data. In more detail, an adversary looks for a collision between the MACs associated to two packets that were MACed with the same 32-bit sequence number (see Section 4.4 of [RFC4253]). If a collision is found, then the payload data associated with those two ciphertexts is probably identical. Note that this problem occurs regardless of how secure the underlying encryption method is. Also note that although compressing payload data before encrypting and MACing and the use of random padding may reduce the risk of information leakage through the underlying MAC, compression and the use of random padding will not prevent information leakage. Implementors who decide not to rekey at least once every 2**32 packets should understand these issues. These issues are discussed further in [BKN1,BKN2].

One alternative to recommendation (1) would be to make the SSH Transport Protocol's sequence number more than 32 bits long. This document does not suggest increasing the length of the sequence number because doing so could hinder interoperability with older versions of the SSH protocol. Another alternative to recommendation (1) would be to switch from basic HMAC to a another MAC, such as a

MAC that has its own internal counter. Because of the 32-bit counter already present in the protocol, such a counter would only need to be incremented once every 2**32 packets.

Recommendation (2) states that SSH implementations should rekey before encrypting more than 2**(L/4) blocks with the same key (assuming L is at least 128). This recommendation is designed to minimize the risk of birthday attacks against the encryption method's underlying block cipher. For example, there is a theoretical privacy attack against stateful-decryption counter mode if an adversary is allowed to encrypt approximately 2**(L/2) messages with the same key. It is because of these birthday attacks that implementors are highly encouraged to use secure block ciphers with large block lengths. Additionally, recommendation (2) is designed to protect an encryptor from encrypting more than 2**L blocks with the same key. The motivation here is that, if an encryptor were to use SDCTR mode to encrypt more than 2**L blocks with the same key, then the encryptor would reuse keystream, and the reuse of keystream can lead to serious privacy attacks [SCHNEIER].

6.2. Encryption Method Considerations

Researchers have shown that the original CBC-based encryption methods in [RFC4253] are vulnerable to chosen-plaintext privacy attacks [DAI,BKN1,BKN2]. The new stateful-decryption counter mode encryption methods described in Section 4 of this document were designed to be secure replacements to the original encryption methods described in [RFC4253].

Many people shy away from counter mode-based encryption schemes because, when used incorrectly (such as when the keystream is allowed to repeat), counter mode can be very insecure. Fortunately, the common concerns with counter mode do not apply to SSH because of the rekeying recommendations and because of the additional protection provided by the transport protocol's MAC. This discussion is formalized with proofs of security in [BKN1,BKN2].

As an additional note, when one of the stateful-decryption counter mode encryption methods (Section 4) is used, then the padding included in an SSH packet (Section 4 of [RFC4253]) need not be (but can still be) random. This eliminates the need to generate cryptographically secure pseudorandom bytes for each packet.

One property of counter mode encryption is that it does not require that messages be padded to a multiple of the block cipher's block length. Although not padding messages can reduce the protocol's network consumption, this document requires that padding be a multiple of the block cipher's block length in order to (1) not alter

the packet description in [RFC4253] and (2) not leak precise information about the length of the packet's payload data. (Although there may be some network savings from padding to only 8-bytes even if the block cipher uses 16-byte blocks, because of (1) we do not make that recommendation here.)

In addition to stateful-decryption counter mode, [BKN1,BKN2] describe other provably secure encryption methods for use with the SSH Transport Protocol. The stateful-decryption counter mode methods in Section 4 are, however, the preferred alternatives to the insecure methods in [RFC4253] because stateful-decryption counter mode is the most efficient (in terms of both network consumption and the number of required cryptographic operations per packet).

Normative References

[AES] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication 197, November 2001.

[DES] National Institute of Standards and Technology, "Data Encryption Standard (DES)", Federal Information Processing Standards Publication 46-3, October 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2144] Adams, C., "The CAST-128 Encryption Algorithm", RFC 2144, May 1997.

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: Protocols algorithms and source in code in C", Wiley, 1996.

[SERPENT] Anderson, R., Biham, E., and Knudsen, L., "Serpent: A proposal for the Advanced Encryption Standard", NIST AES Proposal, 1998.

[TWOFISH] Schneier, B., et al., "The Twofish Encryptions Algorithm: A 128-bit block cipher, 1st Edition", Wiley, 1999.

Informative References

[BKN1] Bellare, M., Kohno, T., and Namprempre, C., "Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol", Ninth ACM Conference on Computer and Communications Security, 2002.

[BKN2] Bellare, M., Kohno, T., and Namprempre, C., "Breaking and Provably Repairing the SSH Authenticated Encryption Scheme: A Case Study of the Encode-then-Encrypt-and-MAC Paradigm", ACM Transactions on Information and System Security, 7(2), May 2004.

[DAI] Dai, W., "An Attack Against SSH2 Protocol", Email to the ietf-ssh@netbsd.org email list, 2002.

Authors' Addresses

Mihir Bellare
Department of Computer Science and Engineering
University of California at San Diego
9500 Gilman Drive, MC 0404
La Jolla, CA 92093-0404

Phone: +1 858-534-8833
EMail: mihir@cs.ucsd.edu

Tadayoshi Kohno
Department of Computer Science and Engineering
University of California at San Diego
9500 Gilman Drive, MC 0404
La Jolla, CA 92093-0404

Phone: +1 858-534-8833
EMail: tkohno@cs.ucsd.edu

Chanathip Namprempre
Thammasat University
Faculty of Engineering
Electrical Engineering Department
Rangsit Campus, Klong Luang
Pathumthani, Thailand 12121

EMail: meaw@alum.mit.edu

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).