

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2008

E. Rescorla
RTFM, Inc.
March 09, 2008

Notes on TCP Authentication Architectures
draft-rescorla-tcp-auth-arch-00.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 10, 2008.

Abstract

This document provides an architectural survey of a variety of approaches to key management for TCP-level authentication.

Internet-Draft

TCP Auth. Arch

March 2008

Table of Contents

1.	Introduction	3
2.	Conventions Used In This Document	3
3.	Keys and Associations	3
3.1.	Different Keying Material Per Connection	4
3.2.	Rekeying for Key Compromise	5
3.3.	Limiting Plaintext/MAC Pairs	5
4.	Credentials Interface	6
5.	Handshake and Capabilities Discovery	6
6.	Potential Architectures	7
6.1.	Architectures Without Handshakes	7
6.1.1.	Single Static Key	7
6.1.2.	Implicit Diversification	8
6.2.	Architectures with Handshakes	9
6.2.1.	Internal Key Management Protocol	9
6.2.2.	External KMP	10
6.2.3.	Layered KMP	11
7.	Security Considerations	12
8.	IANA Considerations	13
9.	Informative References	13
	Author's Address	14
	Intellectual Property and Copyright Statements	15

1. Introduction

The TCP MD5 Authentication option [[RFC2385](#)] describes an mechanism for authenticating TCP segments using an MD5-based MAC (though the document does not use that term). The MAC is keyed using a single static key shared between a pair of TCP endpoints.

Recent work by Wang et al. [[WH05](#)] has demonstrated significant weaknesses in MD5 which imply that TCP MD5 should be phased out in favor of a more modern MAC algorithm. In addition, it seems worth addressing some of the more glaring protocol flaws in TCP MD5. The consensus requirements are probably the following:

Algorithm agility - The ability to support multiple MAC algorithms.

Strong MACs - Mandatory support for some strong MAC algorithm.

Key rollover - Support for changing MAC keys during the duration of a TCP connection without breaking the connection.

Note that there is currently two documents [[I-D.ietf-tcpm-tcp-auth-opt](#)] [[I-D.bellovin-tcpsec](#)] that discuss additional requirements, but it's not clear to me that these requirements have consensus or in fact are correct.

The remainder of this document discusses some additional architectural issues and then describes a series of different design approaches formed by answering these architectural issues in different ways.

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](#)].

3. Keys and Associations

Probably the most important architectural question is the relationship of keys to associations. As indicated above, TCP MD5 uses a single static key for all associations. This key is configured in a pairwise fashion. By contrast, conventional channel security protocols such as TLS [[RFC4346](#)] or IPsec [[RFC4301](#)] establish a fresh set of keying material for each association. The proposed TCP Authentication Option [[I-D.ietf-tcpm-tcp-auth-opt](#)] also requires (though does not arrange for) a fresh key for each association.

Rescorla

Expires September 10, 2008

[Page 3]

Internet-Draft

TCP Auth. Arch

March 2008

>> New TSAD entries MUST be checked against a table of previously used TSAD entries, and key reuse MUST be prohibited.

There are two good (and one not so good) reasons why it's desirable to have mechanisms for changing keys:

- o To arrange for different cryptographic keying material to be used for each connection, thus preventing cut-and-paste attacks between connections. (Good)
- o To allow rekeying in case of key compromise. (Good)
- o To limit the amount of plaintext/MAC pairs available to the attacker (Not-so-good)

The following sections discuss each of these issues.

3.1. Different Keying Material Per Connection

It's standard cryptographic practice to try to use a cryptographic key for one purpose and by extension it's good key hygiene to use a separate key for each association. The reason for per-association keying is to prevent cut-and-paste/replay attacks between associations. To take a simple example, consider what would happen if TLS used the same key for any communication between two elements. An attacker could record your connection and replay it, thus potentially causing you to (for instance) issue multiple orders. Using a separate key for each association provides automatic cryptographic protection against this kind of attack--the replayed PDUs simply fail verification.

Note that any TCP authentication solution where the authentication tag covers the TCP header has some built-in resistance to this sort of attack because each association has a different set of ports, so you can only substitute between pairs of connections with the same port. Obviously, ports do get reused, so this isn't perfect. It's quite a bit better if client-side port randomization is used, and much worse if systems use deterministic port numbers (e.g., using the same sequence of client-side port numbers after each reboot.)

There's actually a general observation here: when you're only providing integrity, there's a (mostly) isomorphism between incorporating diversifying information into the keying material and incorporating it into the integrity check. I.e., if you have a per-connection identifier and a fixed key, you can do:

```
* K_conn = HMAC(K_fixed, conn_id); Packet_MAC = HMAC(K_conn, packet)
```

or

```
* Packet_MAC = HMAC(K_fixed, conn_id || packet)
```

This doesn't work for encryption as well because of accidental plaintext collisions and/or two-time pad issues, but it's not a problem for MACs.

In TCP, cut-and-paste attacks are also possible within a connection due to sequence number rollover. This can be fixed, however, by extending the sequence numbers virtually, as done with ESP/AH.

[3.2.](#) Rekeying for Key Compromise

If the long-term key used to authenticate an endpoint becomes known to an attacker, then the attacker can impersonate that endpoint until the key is revoked (i.e., until the counterparties know that key is no longer valid). If the endpoint wishes to continue communicating, then a new key will need to be established as well. Note that this also applies to traffic keys: in fact, since they are generally used with symmetric rather than asymmetric modes, the attacker can typically impersonate both A to B and B to A.

In the BGP case that everyone is concerned about, this all needs to be done without breaking the connection, so this implies the need to

impose entirely new cryptographic state on the connection without breaking it.

Note that this is an orthogonal issue to the question of whether connections use the same key: even if all your associations simultaneously used the same keys, you might still want to rekey them all to recover from a compromise.

[3.3.](#) Limiting Plaintext/MAC Pairs

A classical concern of COMSEC designers has been to limit the amount of plaintext/ciphertext material available to attackers for a single key:

There are two primary rationales for this practice:

- o Many older algorithms are weak if large numbers of plaintext/ciphertext pairs are available. In some cases this means analytic attacks. In others it means some form of table-driven dictionary attack such as CBC IV rollover. The latter form of attack typically isn't that powerful, with the exception of counter-mode ciphers which have hard limits on counter use.
- o To increase the attacker's workload: if the attacker needs to break a new instance of the cipher for each association, then it doesn't pay off to capture six months work of traffic and break it at leisure.

The first concern is a lot less applicable with modern crypto algorithms such as AES or HMAC. No known good analytic attacks are known on these systems no matter how many plaintext/ciphertext pairs you have, and collision limits due to blocksize are fairly far out: 2^{64} blocks for AES, $2^{b/2}$ blocks for HMAC, where b is the size of the authentication tag used. This isn't to say that they're totally irrelevant, but regular rekeying simply is not necessary.

The second concern is rather less relevant for integrity checks, systems, since a successful attack on an integrity system doesn't compromise old traffic; you can just attack newer traffic. This means the attacker has to break the system during the life of a single association, which is much more limiting.

[4.](#) Credentials Interface

As described at the beginning of this document, essentially all the currently deployed systems use a single pre-configured pairwise shared key. This key is directly configured on the router interface. For instance, here's the example from Cisco's IOS manuals [REF: http://www.cisco.com/en/US/docs/ios/12_0/np1/configuration/guide/1cbgp.html#wp5978]

```
router bgp 109 neighbor 145.2.2.2 password v61ne0qkel33&
```

It's extremely desirable to be able to retain the existing interfaces. Any solution which requires a significant change to those interfaces and especially to the interaction model creates a substantial additional burden on operators, which creates a major barrier to deployment.

One important implication of this constraint is that a workable system must either allow a single key (or perhaps set of keys) to be used as traffic keys for multiple connections/association, or it must allow for some method of automatic key establishment based on a single small set of initial keys. A system which requires a fresh key for each connection and does not provide any kind of automatic key establishment will break once the set of preconfigured keys is exhausted, which is unacceptable in a production system.

[5.](#) Handshake and Capabilities Discovery

The ability to automatically discover a peer's capabilities is a common feature of modern cryptographic protocols. For instance, SSL/TLS, IPsec (IKE [[RFC4306](#)]), and SSH all offer some form of algorithm negotiation, in which the sides mutually determine some common set of

acceptable algorithms.

This capability has obvious advantages in systems like SSL/TLS and IKE where endpoints may be communicating with other endpoints with which they share no preexisting state: it allows the implementations to discover each others capabilities without having them preconfigured. However, even in systems where the keying material is statically and manually configured, an automatic discovery system

removes the need for a configuration knob (the algorithms for this remote system), which potentially reduces errors. In addition, it allows for the possibility of automatic upgrade between algorithms (and other functionality) without reconfiguring each peer. For instance, if peer A shares keys with 10 other peers and wants to roll out new algorithm X, it can simply upgrade its implementation and X will be used with the other peers automatically as they upgrade without requiring manual discovery.

[6.](#) Potential Architectures

This section describes a number of potential architectures for key management for TCP. These descriptions are sketches and are certainly not complete, nor do we claim to describe all possible architectures.

[6.1.](#) Architectures Without Handshakes

We first consider architectures which do not involve any explicit handshake. These systems have the advantage of being simple to design and implement but lack the flexibility benefits of a handshake scheme, as described in [Section 5](#)

[6.1.1.](#) Single Static Key

The simplest design is simply to use the same architecture as TCP MD5: a single pairwise symmetric key used for all connections between two endpoints, but updating it to meet the requirements of [Section 1](#). This would imply the following changes:

- o A description of how to add new MACs
- o Support for a strong MAC (e.g., HMAC-SHA1)
- o Some description of how to do key rollover (this can either be done with explicit identifiers or a trial verification hack like that described in [\[RFC4808\]](#)).

This mechanism would provide roughly similar architectural security properties to TCP MD5, except with stronger crypto, including an almost identical user interface and a very similar implementation.

paste attacks in between connections if they happen to have the same connection parameters (host/port quartet).

The trial verification technique involves trying to verify segments with both key/algorithm pairs. Once the newer key/algorithm pair verifies, that can be treated as the maximum sequence number at which the switchover occurred (it may have occurred earlier if packets were lost). Note that in principle with this technique you could even use the same TCP MD5 option code point and bits on the wire with a stronger MAC. Since TCP MD5 requires pairwise keys which must be manually configured, all that would be required would be to configure a pairwise algorithm. This is probably unwise because it reinterprets the meaning of the TCP MD5 option, but it shows how similar the two techniques are.

6.1.2. Implicit Diversification

A substantial amount of resistance to inter-connection cut-and-paste attacks can be achieved by exploiting connection information other than the host/port quartet. For instance, one could start with a static key and use the TCP ISNs to produce a unique per-connection key:

```
K_connection = HMAC(K_static, ISN_client || ISN_server)
```

If we treat ISNs as random variates (note: this does not require cryptographic randomness, just low probability of collisions) then there are 2^{64} equally probable ISN combinations between any pair of hosts, which implies 2^{64} potential keys (even if we ignore the host/port quartet), so the chance of two connection keys colliding becomes acceptably low: on average there will be a collision after 2^{32} connections. If the host/port quartet is included, then collisions become even less probable.

Note: the ISNs are being used here serve the same role as nonces in standard cryptographic protocols (e.g., the ClientRandom and ServerRandom in TLS). Therefore, the security model explicitly assumes that they are public information. Thus, the sequence number predictability issues described on [[RFC1948](#)] do not impact the security of the system. Only nonce uniqueness is required.

This scheme has very similar security properties to the scheme described in [Section 6.1.1](#), including having a compatible key management interface. Unlike static keys, however, there is a significant amount of resistance to inter-connection cut-and-paste attacks because each connection with high probability uses a unique key.

As with any system that requires per-connection keys, this scheme would require some amount of per-connection state to store that key. This, and the hooks to feed the ISNs into the key computation, would presumably require some changes to implementations.

[6.2.](#) Architectures with Handshakes

We now consider architectures which involve explicit protocol handshakes between the two endpoints. As noted in [Section 5](#), this provides more flexibility and upgradeability but at significantly more protocol and implementation cost. In addition, like all handshake methods, we would expect it to provide unique per-connection keys. The difference between these architectures is in how the key management protocol messages are carried.

[6.2.1.](#) Internal Key Management Protocol

The traditional approach to this problem would be to build a KMP into TCP. This would presumably entail designing a new crypto protocol (no small job) which is then carried in a TCP option.

The primary advantage of this sort of design is that it allows for tight integration with TCP. In general, tighter protocol integration allows for the provision of security services which are better tuned to the target protocol (cf. SRTP versus RTP over TLS).

The primary disadvantage of this design is the large amount of effort involved, including:

- The cost of designing an entirely new protocol.
- The cost of integrating the crypto protocol into the TCP stack (which is more difficult than a normal crypto protocol because it is typically in the kernel.)

In addition, this mechanism may be less flexible than other handshake-based systems. First, many handshakes involve multiple round-trips between the client and server. This is not a natural fit for the TCP handshake, which only involves three messages. Other messages can potentially be piggybacked on empty TCP segments (bare ACKs), but their delivery properties would need careful study. [Note: this also applies to the final handshake message, the ACK, so we may be reduced to a two message handshake.]

Second, if the implementation is in the kernel, then this reduces one's ability to deploy public key-based mechanisms in the future (one of the major value propositions of a handshake mechanism),

because the public key operations are too slow to run in the kernel while stalling all other operations. This either requires a

multithreaded kernel or a piecewise/asynchronous public key computation (something that takes careful programming and that current crypto implementations don't do).

[6.2.2.](#) External KMP

The approach used by IPsec is to split the system into two pieces:

A KMP establishes cryptographic associations (e.g., IKE).

A standalone cryptographic transport protocol which relies on the associations created by the KMP (e.g., AH/ESP).

This architecture has the advantage of decoupling (at least partly) the key management system from the protected transport protocol. At least in principle this allows the KMP to be developed independently of the transport protocol, encouraging modular design and (potentially) mix-and-match between key management protocols and transport protocols. The experience with IPsec suggests that both of these benefits may exist more in principle than in practice. In particular, IKE has a fair amount of embedded knowledge about AH/ESP and as far as I know is not used to key any other transport protocol. That said, IKE is probably the only plausible candidate to be used in this fashion with TCP.

A second advantage is that because the KMP is In separate, it is not tied to the dataflow of the transport protocol. Thus, even though IP has no notion of reliable transport or even of associations, IKE is able to have reliable multi-message exchanges because it is transported separately. In the case of TCP, this would mean that it would be straightforward to have an arbitrary number of key management messages without having to worry about TCP handshake dynamics.

An additional advantage of having an external key management protocol is that it can be implemented separately from the transport protocol. For instance, conventional IPsec implementations have AH/ESP implemented in the kernel with IKE implemented in a userland daemon. This has obvious advantages in terms of the relative congeniality of the two environments, but also note that it allows the expensive PK

operations to be done in the background without stalling the kernel.

The major disadvantage of this design is the extremely loose coupling between the KMP and the transport protocol. This manifests in at least three ways:

- o The need for an association database to map keying material to the traffic it protects.

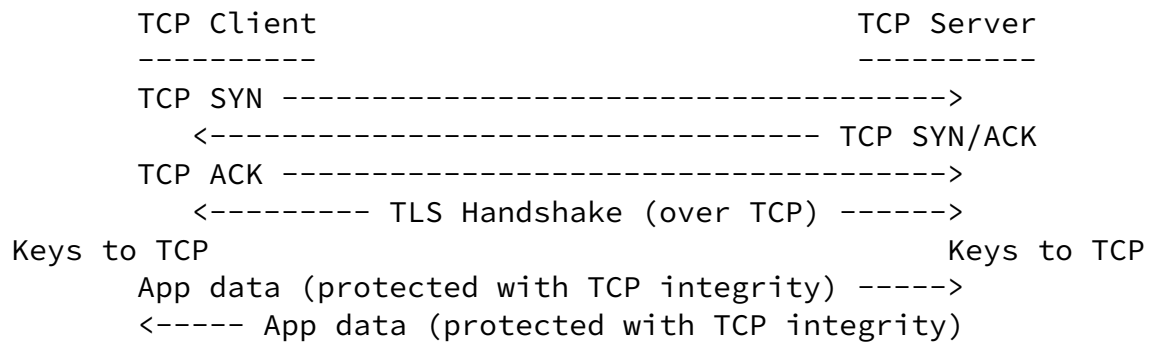
- o The difficulty of configuring settings and giving applications information about association properties.
- o The need for interfaces to allow the transport protocol to request key establishment from the KMP and (potentially) to stall the transport protocol while keys are established.

The last of these is probably more serious for TCP than for IPsec because an IPsec SA can be configured to connect a large number of different transport-level associations. There may be a new key management protocol run for each TCP connection. This may make stalling the transport protocol impractical, but then this exacerbates the problem of application visibility--how does the application know if the security association was set up or what its properties are?

Another disadvantage is that if the transport of the key management protocol is decoupled from that of the transport protocol it is establishing keys for, then there may be ways for the key management protocol to fail (e.g., firewalls) but the transport protocol succeeds.

6.2.3. Layered KMP

Another way to provide a separate KMP is to bind it more tightly to the transport protocol by running it over (or next to, as in DTLS-SRTP [[I-D.ietf-avt-dtls-srtp](#)]) the transport protocol. At the time that the association is created, the application initiating the association also initiates a KMP exchange over (next to) the transport protocol. When the KMP terminates, it outputs keying and parameter information and imposes them on the association. In the case of TLS over TCP, this would look something like:



The primary advantage of this type of system is that it has a tight binding to both the transport and to the application. Because the KMP has the same transport parameters as the application layer protocol which will run over the transport and conceptually as the transport itself, they have a simple one-to-one relationship.

The application can control the cryptographic parameters directly. The application gets visibility into what parameters were established.

The KMP fate-shares with the transport and application layer protocols.

This architecture has two major disadvantages. First, unlike all the other architectures discussed, it requires the cooperation of the application layer protocol and implementation. This is the price that you pay for the tight application coordination. Second, it creates a window of vulnerability during the period after the transport protocol connection has been established and before the KMP has run. This is not an issue for injection attacks, provided that the application layer protocol state machine keeps track of which state it is in and refuses to accept un-integrity protected data (a standard issue in any application layer protocol when used with a channel security mechanism.) However, DoS attacks on the connection are possible during this initial period, which provides a window for attack which does not exist with these other modes.

[6.2.3.1.](#) TLS as a Layered KMP

In the specific case of TLS, another advantage is that this can be treated by the application as if it were doing TLS over TCP. The application can simply do TLS as it normally would provided that the

API stack provides a mechanism for indicating that it wishes to also negotiate keys for TCP authentication. Two models are actually possible here:

In the first, TLS is used purely as a KMP and then it gets out of the way and data is sent "in the clear" over TCP with an integrity check. This has the advantage of efficiency, at a slight cost to the cleanness of the TLS model.

In the second, the data is protected with TLS before being passed to TCP, where it is integrity checked again. This is less efficient (because two MACs are performed) but slightly more compatible with a "just use TLS" model. It also provides an opportunity for confidentiality if an appropriate cipher suite is used.

7. Security Considerations

This entire document is about security.

8. IANA Considerations

This document has no IANA considerations.

9. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", [RFC 2385](#), August 1998.
- [RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", [RFC 1948](#), May 1996.
- [RFC4808] Bellovin, S., "Key Change Strategies for TCP-MD5", [RFC 4808](#), March 2007.

- [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", [RFC 4346](#), April 2006.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", [RFC 4301](#), December 2005.
- [RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", [RFC 4306](#), December 2005.
- [I-D.ietf-tcpm-tcp-auth-opt]
Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", [draft-ietf-tcpm-tcp-auth-opt-00](#) (work in progress), November 2007.
- [I-D.ietf-avt-dtls-srtp]
McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP)", [draft-ietf-avt-dtls-srtp-02](#) (work in progress), February 2008.
- [I-D.bellovin-tcpsec]
Edddy, W., Bellovin, S., and R. Bonica, "Problem Statement and Requirements for a TCP Authentication Option", [draft-bellovin-tcpsec-01](#).
- [WH05] Wang, X. and H. Yu, "How to Break MD5 and Other Hash Functions", EUROCRYPT 2005.

Rescorla

Expires September 10, 2008

[Page 13]

Internet-Draft

TCP Auth. Arch

March 2008

Author's Address

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Email: ekr@rtfm.com

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.