April 1996

Creating Efficient Fail-Stop Cryptographic Protocols

Angelos D. Keromytis
University of Pennsylvania

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-96-32.

Creating Efficient Fail-Stop Cryptographic Protocols

Abstract

Fail-stop cryptographic protocols are characterized by the property that they terminate when an active attack is detected, rather than releasing information valuable to the attacker. Since such a construction forces attacks (other than denial-of-service) to be passive, the protocol designer's concerns can be restricted to passive attacks and malicious insiders. A significant advantage of such protocols is that by stopping and not attempting to recover, proofs about protocol behavior and security properties are greatly simplified. This paper presents a generic method of converting *any* existing (cryptographic) protocol into a fail-stop one, or designing new protocols to be fail-stop. Our technique uses cryptographic hashes to validate sequences of messages by reflecting message dependencies in the hash values. An informal proof of correctness is given. We apply it to an early version of Netscape's Secure Socket Layer (SSL) cryptographic protocol. We also suggest a possible application to TCP streams as a high-performance alternative to the per-packet authentication of IPSEC. The modified protocols require small increases in message size and the number of cryptographic operations relative to the initial non-fail-stop protocols.

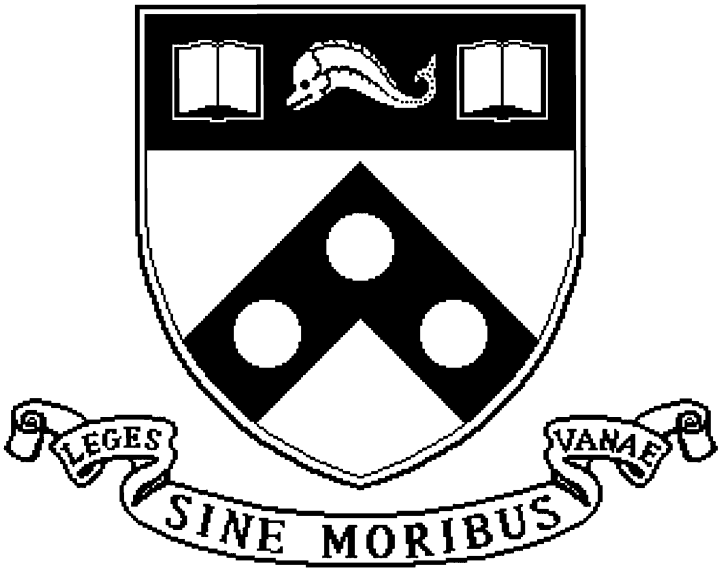
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-96-32.

Creating efficient fail-stop cryptographic protocols

MS-CIS-96-32

Angelos D. Keromytis
Jonathan M. Smith



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1996

Creating efficient fail-stop cryptographic protocols *
Technical Report MS-CIS-96-32

Angelos D. Keromytis
University of Pennsylvania
Jonathan M. Smith
University of Pennsylvania

April 4, 1996

Abstract

Fail-stop cryptographic protocols are characterized by the property that they terminate when an active attack is detected, rather than releasing information valuable to the attacker. Since such a construction forces attacks (other than denial-of-service) to be passive, the protocol designer's concerns can be restricted to passive attacks and malicious insiders. A significant advantage of such protocols is that by stopping and not attempting to recover, proofs about protocol behavior and security properties are greatly simplified.

This paper presents a generic method of converting *any* existing (cryptographic) protocol into a fail-stop one, or designing new protocols to be fail-stop. Our technique uses cryptographic hashes to validate sequences of messages by reflecting message dependencies in the hash values. An informal proof of correctness is given. We apply it to an early version of Netscape's Secure Socket Layer (SSL) cryptographic protocol. We also suggest a possible application to TCP streams as a high-performance alternative to the per-packet authentication of IPSEC.

The modified protocols require small increases in message size and the number of cryptographic operations relative to the initial non-fail-stop protocols.

 Authors' email addresses are angelos@dsl.cis.upenn.edu and jms@central.cis.upenn.edu. This research was supported by DARPA under contract #N66001-96-C-852.

1 Fail-stop protocols

Cryptographic protocols are widely used in many advanced applications, such as electronic banking, networked software distribution, and wireless personal communications systems. Due to the complexity of conditions they may encounter, careful reasoning and formal means such as proofs are used to validate the design of a cryptographic protocol. Such validation is easier if the set of threat conditions is reduced. If this reduction is via assumptions which ignore reality, the validation becomes worthless when the assumptions are falsified. Techniques resulting in the construction of protocols which *by design* reduce the complexity of threat conditions are thus extremely attractive.

One such idea is a *fail-stop* cryptographic protocol, recently introduced by Gong and Syverson [FS]:

> A protocol is *fail-stop* if any attack interfering with a message sent in one step will cause all causally-after messages in the next step or later not to be sent.

As Gong and Syverson show, fail-stop protocols possess a very useful security property, namely:

> active attacks cannot cause the release of secrets within the run of a fail-stop protocol

The fail-stop property lets a protocol designer restrict his or her concerns to passive (eavesdropping) attacks, a significant reduction in the class of threats to the protocol's security. There is, of course, no free lunch: the protocol must terminate when active attacks occur, rather than attempting to continue. However, when embedded in a larger system, this termination can be handled by higher-level detection and resolution mechanisms. We believe that reliable

termination is greatly preferred to unknown and insecure behavior in the face of active attacks on security.

1.1 Specifying fail-stop behavior

Syverson and Gong state the following specifications for a fail-stop protocol:

1. The content of each message has a header containing the identity of its sender, the identity of its intended recipient, the protocol identifier and its version number, a message sequence number, and a freshness identifier.
2. Each message is encrypted under the key shared between its sender and intended recipient.
3. An honest process follows the protocol and ignores all unexpected messages.
4. A process halts any protocol run in which an expected message does not arrive within a specified timeout period.

The above specifications assume that the two communicating parties share a secret encryption key used with a symmetric key cryptosystem (such as DES [FIPS46]).

The freshness identifier can be a nonce issued by the intended recipient or a time stamp (if the clocks are assumed to be securely and reliably synchronized—but see [LG92]).

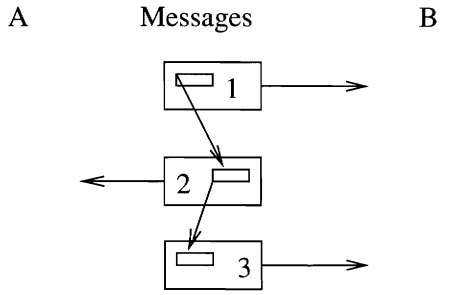
1.2 Outline of this paper

Section 2 presents our method for chaining the messages of a protocol run. This makes the messages sequenced and non-reusable outside the context of this protocol run, thereby making message tampering and replay attacks impossible [PS]. Section 3 gives a detailed example of the methods applied to the SSL cryptographic protocol. Section 4 proposes applications to a large class of protocols, those which provide reliable message streams. Section 5 makes some observations about the method, and addresses some potential criticisms. Section 6 concludes the paper and summarizes its contributions.

2 Message chaining

Fail-stop cryptographic protocols, as specified by Gong and Syverson, require repeating considerable information in each message of the protocol. This has several weaknesses. First, the size of the messages increases. Second, the amount of encryption/decryption required also increases [PET]. Third, and perhaps most consequential, there is more plaintext for an attacker to mount known plaintext attacks [LG90] [PPC].

Instead of including all the information of their specification in each protocol message, we can rely on one-way hash functions, such as MD5 [MD5] or SHA [SHA] to preserve the sequencing of the protocol messages.



The initiator of the protocol run starts by sending the first message including all the aforementioned information (some of which might already be included in the original protocol). Additionally, a hash of the entire message (including the new fields) is sent. That value is also kept as local state.

The responder verifies these values and then proceeds with the normal flow of the protocol. From there on, each message exchanged contains the one-way hash result of the new message and the state of the sender of that particular message. The receiver of each message calculates the hash of the message and his state and compares it with the hash value he received. If they're not the same, then some active attack is assumed to be taking place.

Naturally, the hash values cannot be sent along with the message in the clear; an attacker could tamper with those. There are three alternatives to securing them:

1. Sign the hash values with one's private key, if the peer is known to have one's public key, when using a public key cryptosystem (such as RSA [RSA] [RSA1]).
2. Encrypt the hash value with a shared secret key using a symmetric cryptosystem.
3. Use keyed hashes (where the key has to be used in each hash function application). Care must

be taken to use a strong way of keyed-hashing [BCK].

2.1 The hash function

Here we assume that the hash function used has four desirable properties:

- Collision resistance: an attacker cannot create another message with the same hash value.
- Entropy preservation: the effect that some value has in the hash result does not disappear, at least not before a reasonably large number of applications of the hash function.
- Non commutativity: reordering the messages will make the results invalid.
- Given subset of the input to the hash function and the result, an attacker can not find what the missing input bits are.

Depending on the particular protocol, additional properties might be required from the hash function [RJA].

2.2 Correctness of our method

Keeping in mind that the verification at each step of a protocol is of the form:

- H(*Key, State, Message*) $\stackrel{?}{=}$ Message.*Hash*

There are three methods of active attacks a malicious entity can attempt:

1. Find the key. We have already made the (weak, as cryptographic hash functions go) assumption that this is not possible.
2. Inject a new message such that the verification succeeds. This means that the attacker is capable of creating collisions on the hash function even when a secret quantity (the *Key*) is involved. Again, this is a relatively weak assumption we have already made.
3. Affect the *State* kept by either of the legitimate protocol parties, so that a captured message can be replayed. Manipulating the *State* to some desired value - even if it were possible - would not allow the attacker to introduce a new message of his own, since the *Key* used is unknown to him. Also, the chance of the *State* chancing to be the same as the one at a previous step of the protocol is negligible (approximately $(1/2)^{n/2}$, where **n** is the length in bits of the output of the hash function).

However, manipulating the *State* to some particular value means that the previous round of the protocol has been tampered with successfully [1]. This in turn (because of the reasons given above) means that the round before that has been successfully attacked.

Following this argument, the attacker would have to affect the first message in the protocol. More specifically, he would have to substitute it with another message from a previous or parallel run of the protocol which uses the same *Key*. The reasons this is not possible are:

- the first message can not be tampered with, since it involves usage of a secret key and the hash function is collision free
- it contains enough information to distinguish it from any other first-message of the same protocol

The above are not intended as a formal proof of correctness of our method, but rather as a line of reasoning which we believe is sufficiently convincing.

2.3 Other attacks

There is one final point of concern: although it is impossible for an attacker to inject a message in the middle of a protocol run under our scheme, it is still possible to use a captured message as the first message in a protocol. Of course, the protocol must allow this attack by message insertion in the first place, and such an attack is precluded if it is impossible for an attacker to remove the hash value from a message (he can discard that field if it's a keyed hash for example, but not if it's encrypted in CBC mode [CBC] with all other message fields).

If the protocol designer is also worried about insider attacks, he or she should ensure that any valuable pieces of information (such as a digital signature) is inseparable from the hash value. For example, in the case of a digital signature, one would use the hash value in the signature computation.

[1] We assume the attacker does not have access to the internals of either of the legitimate protocol parties.

3 An example: Netscape's SSL

An early version of the SSL protocol [SSL] included the following messages:

1. A $\rightarrow$ B: $\{K_{ab}\}_{K_b}$
2. B $\rightarrow$ A: $\{N_b\}_{K_{ab}}$
3. A $\rightarrow$ B: $\{CA, \{N_b\}_{K_a^{-1}}\}_{K_{ab}}$

Here, A and B are a client and a server respectively, K_b is B's public key, K_a is A's public key, CA is a certificate of A's public key, K_a^{-1} denotes signing with A's secret key, K_{ab} is a session secret key and N_b is a nonce/challenge. There are more messages in the protocol, but as they are irrelevant to client authentication we will ignore them.

This version of the SSL protocol has a flaw, noted in [PET]. If C is a malicious server and A tries to communicate with it, then C can impersonate A to another server B:

1. A $\rightarrow$ C: $\{K_{ac}\}_{K_c}$
2. C $\rightarrow$ B: $\{K_{ac}\}_{K_b}$
3. B $\rightarrow$ C: $\{N_b\}_{K_{ac}}$
4. C $\rightarrow$ A: $\{N_b\}_{K_{ac}}$
5. A $\rightarrow$ C: $\{CA, \{N_b\}_{K_a^{-1}}\}_{K_{ac}}$
6. C $\rightarrow$ B: $\{CA, \{N_b\}_{K_a^{-1}}\}_{K_{ac}}$

at which point C has convinced B that he is A. Applying our method would not be sufficient, since C is an internal attacker (someone A wants to communicate with directly). A slight change is required: in message 3, A will sign not only the nonce but also the history of the protocol up to that point. This will ensure that the signature and the protocol history are inseparable to an internal attacker as well as to an external one.

1. A $\rightarrow$ B: $\{K_{ab}, A, B, P, H1(K_{ab}, A, B, P)\}_{K_b}$
2. B $\rightarrow$ A: $\{N_b, H2(N_b, H1)\}_{K_{ab}}$
3. A $\rightarrow$ B: $\{CA, \{N_b, H3(CA, H2)\}_{K_a^{-1}}\}_{K_{ab}}$

where P in message 1 is the protocol identifier number and other protocol-run identifying information (such as network addresses/ports). Also notice that in the same message, K_{ab} acts as a nonce, in addition to being the session key.

This solution is similar in concept to the one proposed in [PET]. The current version of SSL does not have this flaw.

4 Application to stream protocols

Our technique can be applied to stream protocols (such as TCP) [TCP] over packet switched networks such as the Internet. Instead of authenticating each individual packet separately [IPSEC], we can authenticate the whole data stream in the same way. However, there are some issues that need to be addressed:

- The hash value should be calculated only over the invariant (between possible retransmissions) portions of the packet. Consequently, the underlying network layers and infrastructure should not modify the packet (or if they do, the remote end should reconstruct it in exactly the same form as it was transmitted). Under this rule, IP [IP] packet fragmentation is permitted.
- User-application delivery of *out of sequence packets* is not permitted, since they cannot be authenticated before all previous packets have been received. This ordering requirement does not pose any problem, as it reflects the usual semantics of stream protocols.
- Acknowledgments do not mean that a packet was accepted as authenticated, but rather that it was received without any transmission errors.
- Only those portions of the packet that are included in the hash value computation are considered trustable.

We believe that the best place do do this in the IP protocol stack is at the TCP layer. We would then additionally include in the hash computation the source and destination IP addresses (which violates the layering model) and create good random initial sequence numbers (which can be considered nonces). If used in other layers of a protocol stack (such as the network or the application), it might be necessary to include additional information in the first message to make the protocol distinguishable from any other, or make sure that it does not interfere with the normal operation of the network (e.g. in the case of packet retransmissions).

5 Observations and Discussion

The method we showed in the previous section is a superset of message authentication. In the case of one message sent, the hash value is equivalent to a

MAC. For each successive message sent (in either direction), the check made by the receiver is equivalent to checking a MAC over the particular message and the receiver's state. So, for message X, the receiver checks the validity of message X and his local state. Since we assume that an attacker cannot modify the hash value in an undetectable way, changes made in messages X and X-1 (the previous message in the protocol run) will be detected in this check. Furthermore, the state cannot be influenced by an attacker since an attempt to do so would have been detected in a previous step of the protocol.

At any time, the local state depends on the messages sent and received at that point, the specific order they were sent in, the initiator's and responder's identities, a freshness identifier, a protocol identifier and a protocol version number (all these were included in the first message only). Because of the required properties of the hash function, duplication of all these fields in subsequent messages is not necessary. Additionally, message sequence numbers are also not necessary, because inclusion of the state in the hash value computation ensures data sequencing.

6 Conclusion

We have presented a method of designing fail-stop protocols based on message-chaining. The method relies on a representation of the state of a cryptographic protocol in a secure hash value, and was demonstrated on a familiar cryptographic protocol, SSL, as a proof-of-concept example. The message-chaining idea of method is sufficiently general to be applicable to non-cryptographic protocols, such as the Internet protocol for reliable datastreams, TCP.

Ensuring that a protocol is fail-stop allows the designer to restrict his or her concerns to malicious insiders and passive (eavesdropping) attacks. Our method can be used to construct efficient implementations of such fail-stop protocols, as it neither greatly increases the size of the messages nor the number of cryptographic operations required.

7 Acknowledgments

The authors would like to thanks Dave Farber, Li Gong and Paul Syverson whose ideas and previous work influenced this paper. Thanks also go to Bill Arbaugh, Alex Garthwaite, Scott Alexander and Jonathan Shapiro for reviewing earlier versions of this paper.

References

[FS] "Fail-Stop Protocols: An Approach to Designing Secure Protocols", *Gong, Li and Syverson, Paul, Proceedings of IFIP DCCA-5, September 1995*

[PET] "Prudent Engineering Practice for Cryptographic Protocols", *Abadi, Martin and Needham, Roger, IEEE Computer Society Symposium on Research in Security and Privacy, 1994*

[FIPS46] "NBS FIPS PUB 46 - Data Encryption Standard", *National Bureau of Standards, U.S. Department of Commerce, January 1977*

[MD5] "The MD5 Message Digest Algorithm", *R.L. Rivest, RFC 1321, April 1992*

[SHA] "NIST FIPS PUB 180 - Secure Hash Standard", *National Institute of Standards and Technology, U.S. Department of Commerce, May 1993*

[BCK] "Keying Hash Functions for Message Authentication", *Mihir Bellare, Ran Canetti and Hugo Krawczyk, Advances of Cryptology, Crypto '96 Proceedings*

[RJA] "The Classification of Hash Functions", *Ross Anderson, Proceedings of IMA Conference on Cryptography and Coding, 1993*

[LG90] "Verifiable-Text Attacks in Cryptographic Protocols", *Li Gong, Proceedings of the IEEE INFOCOM '90, June 1990*

[LG92] "A Security Risk of Depending on Synchronized Clocks", *Li Gong, ACM Operating Systems Review, v26n1, January 1992*

[PPC] "Protecting Poorly Chosen Secrets from Guessing Attacks", *Li Gong, T. Mark A. Lomas, Roger M. Needham and Jerome H. Saltzer, IEEE Journal on Selected Areas in Communications, v11n5, June 1993*

[RSA] "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *R.L. Rivest, A. Shamir and L.M. Adleman, Communications of the ACM, v21n2, February 1978*

[RSA1] "On Digital Signatures and Public Key Cryptosystems", *R.L. Rivest, A. Shamir and L.M. Adleman, MIT/LCS/TR-212, January 1979*

[PS] "A Taxonomy of Replay Attacks", *Paul Syverson, Proceedings of the Computer Security Foundations Workshop VII (CSFW7), June 1994*

[TCP] "Transmission Control Protocol", *Postel, J.B., RFC 793, September 1981*

[IP] "Internet Protocol", *Postel, J.B., RFC 791, September 1981*

[IPSEC] "Security Architecture for the Internet", *R. Atkinson, RFC 1825, July 1995*

[WMF] "A Logic of Authentication", *M. Burrows, M. Abadi and R. Needham, ACM Transactions on Computer Systems, v8n1, February 1990*

[CBC] "Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd edition", *Bruce Schneier, John Wiley & Sons Inc., NY 1996*

[SSL] "The SSL Protocol", *K.E.B. Hickman, RFC, Netscape Communications Corp., October 1994*