

Cryptographic Protocol Engineering: Building Security from the Ground Up*

Jim Alves-Foss
Center for Secure and Dependable Software
University of Idaho
Moscow, ID, U.S.A.

Abstract *There are several types of attacks, including multi-protocol attacks, that occur when incorrect assumptions are made about the operating environment of a cryptographic protocol. This paper introduces the concept of a protocol service layer providing separation between cryptographic services and their uses. This layer implements the necessary security functions needed to implement the secure environment assumed by many protocol designers.*

Keywords: cryptographic protocols, network security

1 Introduction

There have been numerous attempts to define, analyze and validate the security of cryptographic protocols. No easy solution has been found. One of the problems encountered in protocol analysis and design is the implicit assumptions made by designers. Sneekenes [1] emphasizes this in his discussion of BAN-style “security proofs” of protocols; protocols which he successfully attacks. Researchers, realizing the fragility of cryptographic protocols have attempted to develop engineering approaches to cryptographic protocol development.

Abadi and Needham [2] presented a set of design principles, amassed over the years, to aid protocol designers in creating more robust protocols. These principles, neither necessary

nor sufficient, are a good foundation for cryptographic protocol engineering.

Kelsey, Schneier, and Wagner [3] determined that secure protocols could be broken when the adversary had access to multiple protocols. In their paper they suggest a list of design principles that would prevent this type of attack. Alves-Foss discusses the limitations of these design principles in the context of a real-world environment [4, 5], which still leaves the protocols susceptible to multi-protocol attacks.

Gong and Syverson [6] proposed the concept of Fail-Stop protocols in an attempt to define a class of protocols that will operate correctly even in the presence of an adversary. In conjunction with these protocols they developed a three-step protocol design and analysis approach whereby the protocol is first classified as fail-stop; second the protocol is validated for security/privacy concerns and third the protocol is checked using a formal logic such as BAN or GNY. Although Gong and Syverson allude to the concept of a multi-protocol attack; they provide no analysis of the security of the protocols in the presence of a multi-protocol attack.

Bellare and Rogaway [7] have taken a different approach. They presented a technique for protocol design and analysis that models a powerful adversary to demonstrate that breaking the cryptographic protocol would be as difficult as breaking the underlying cryptographic primitives. Authors have extended this work for public key use [8], and in the context of an even more powerful adversary [5] who is able to launch a multi-protocol attack.

*This work was partially supported under DARPA grant MDA972-00-1-0001. The contents of this paper do not necessarily reflect the position or policy of the US Government and no official endorsement should be inferred.

The Open Group has taken yet another approach with the Common Data Security Architecture (CDSA) [9]; which imposes a strict structure on the installation and use of cryptographic services, but does not sufficiently address the security of applications that use these services.

In this paper, we have borrowed concepts from several of these approaches to address the concept of engineering secure cryptographic protocols. The important issue to understand when developing a secure protocol is that the environment in which the protocol is executing is very complex. All assumptions about the environment must be explicitly defined. For example, Bellare and Rogaway explicitly define an environment where the adversary has access to unlimited sessions of the same protocol [7]. The design solutions presented by Kelsey et. al. [Kelsey97a], and by Gong and Syverson [6] work if all installed protocols follow these rules. However, not all vendors will necessarily follow these design rules, and worse yet, malicious netizens likely will not follow these rules. Therefore the question we seek to answer in our research is:

Given a general purpose computer, running a commercial operating system, connected to the Internet, is there an architectural model that will enable us to engineer secure protocols?

For the most general system, we would like the system to maintain security even if the following assumptions hold:

- the adversary has complete control over all communication between players (users), can initiate and interact with any number of simultaneous protocols
- the adversary can request that a participant divulge any private information

In this environment, a protocol is said to be secure if, when the private information of a protocol has not been divulged, the players will

only enter an accepting state when the adversary is functionally benign. This is very similar to the concept of a fail-stop protocol [6] and the adversary behavior of the Bellare and Rogaway [7]. However, we have extended these assumptions to include the possibility that the adversary could use different protocols within a single attack [5].

If we require a software only solution, we must assume that the following conditions hold to prevent a player from accidentally divulging private information. These assumptions are generally required to be true for secure network protocols in open systems.

1. The underlying cryptographic algorithm is computationally infeasible to break; otherwise the whole protocol is insecure.
2. The player's private keys are not stolen or revealed; otherwise an adversary can masquerade as the player.
3. There exist secure cryptographic modules that perform as specified and do not contain any Trojan horses and there exists adequate operating system security measures such that the stored copies of user's keys can not be compromised; otherwise flaws in these systems can be used to subvert the protocol.

It is well known that keys (especially public keys) should not be used for more than one purpose (see remark 13.32 in [10]), such as a digital signature key being used to decrypt an encrypted message; or an adversary using a signature of a nonce to force a signature of a blank check. We will assume that there exist key management functions that prevent a key from being used for different purposes. What we address in this paper is how to limit attacks that occur even when a key is used only for its defined purpose.

2 Environmental Model

Consider a layered approach to cryptographic protocol design, similar to that offered by the

cryptographic service providers (or depicted in the CDSA [9]), but with more depth and functionality (Figure 1).

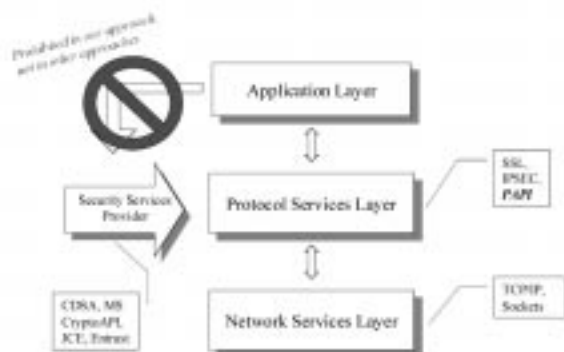


Figure 1: Proposed Architecture.

Operating system layer. It is assumed that this layer provides the basic system security and functionality of commercially available systems. Current popular operating systems provide some of these features but are prone to security design errors and flaws. Although a large portion of system security relies on correct operating system behavior, we will not discuss it further in this paper.

Cryptographic service provider layer. A great deal of effort has gone into developing cryptographic layers for popular operating systems. These layers, interfaced through a cryptographic API, attempt to provide the end user with several important security features:

Cryptographic services. The basic functionality of these layers is to provide implementations of common cryptographic algorithms (e.g., DES, RSA, DSA) to provide encryption and digital signature capabilities.

Key management. In addition to cryptographic algorithms, these layers often provide capabilities for generation and management of keying material, including certificate generation and validation. This may also include a password protected repository of the keying material to prevent the unauthorized disclosure of this material.

Trust. To enhance the security of the sys-

tem, most cryptographic service layers require digital signatures on new modules that implement cryptographic algorithms. This permits the user to install new algorithms into the layer as long as they have been generated by a trusted entity. The cryptographic layer can keep track of the appropriate use of the keying material and control that use in the algorithms. Beyond these controls, the applications are free to use these algorithms as they see fit (as long as the user authorizes the use of the keying material, when authorization is required.)

Protocol service layer. It has been shown [3, 4, 5, 11] that a protocol using a key restricted by the cryptographic layer to a single use (such as digital signature) can still be broken, even if the protocol adheres to all of the design rules proscribed in the literature [12, 2, 3]. The protocol service layer, introduced in this paper, provides the extra functionality needed to prevent these attacks. There are two approaches to this type of layer:

Secure session approach. The SSL and IPSEC communities have advocated the concept that a lower layer should provide a host to host (or application to application) secure communication channel. By secure, they mean encrypted with a recently generated session key; where the session key generation involved anti-spoofing techniques such as nonces, session ids, and authentication. This approach will work as long as the supported algorithms and protocols provide the functionality that the user desires, that they are designed securely (e.g., following fail-stop protocol rules) so that they are not subject to attack, and that the keying material used for encryption and authentication is not available to applications for use outside the cryptographic layer.

Protocol API approach. We advocate a slightly different approach from that discussed above (although it may be implemented on top of a secure session if desired); one that allows the user application to specify any protocol and still be assured of much of the security of the protocol. The impetus for this approach is to permit the application layer as much flexibil-

ity in cryptographic services as possible while still ensuring that the protocols are engineered to be secure. We could require that all protocols go through a rigorous evaluation process to ensure their security (a time consuming and costly requirement) or we can have the layer provide much of this functionality automatically, resulting in a simpler security analysis. We chose to explore the later mechanism in our research.

3 Protocol Application Programmer Interface (PAPI)

The purpose of the protocol services layer, as interfaced by the protocol application programmer interface (PAPI), is to provide a clean separation between cryptographic services and their uses. As discussed in the literature concerning multi-protocol attacks [3, 4, 5, 11], these attacks can occur even when a cryptographic key is used in precisely one manner, but in different protocols, bypassing controls provided by the cryptographic services provider.

The PAPI layer will provide a secure communication session similar to the SSL or IPSEC sessions but with some additional flexibility. The layer will automatically add session ids, participant ids, and checksums to all messages as required by the fail-stop protocol design [6], and in general will apply the appropriate security measures to all protocols.

The PAPI interface requires that the user initialize a protocol session. This initialization includes information regarding the session participants, keys to be used and most importantly, the protocol being used. All subsequent communication for this session must occur through the PAPI layer which will add the necessary security measures. Even if a user wishes a simple digital signature or encryption, the PAPI layer will add the checksum and security parameters to ensure that the resulting cryptographic data could not be used in any other context; avoiding any possible interleaving attack

3.1 Operation of the PAPI Layer

It is our belief that the PAPI interface is best used in two separate contexts. In the first context it supports higher level cryptographic protocols. In the second context it is used as an interface to the lower level cryptographic services. The focus of this paper is on the first context. We can treat the second context as a degenerate case where the protocol consists of users sending themselves a single message in the appropriate format. The operation of a PAPI layer in the first context occurs in three simple phases. The first phase involves session communication initialization which provides a generic form for initializing a cryptographic protocol run with a peer entity. The second phase involves message transport where the PAPI layer supports sending and receiving all messages between the participants. The third phase involves termination of the protocol run.

PAPI Initialization Phase. PAPI initialization requires the establishment of an instance of a PAPI client that will act as the user's intermediary for the life of the protocol run; enhancing the protocol's execution environment. To provide the greatest level of flexibility, the PAPI initialization allows the user to specify which cryptographic protocol they wish to use by passing in a protocol specification written in some precise format. This specification, along with other parameters, will identify all players, keys, cryptographic algorithms, message formats, nonces, and any other values needed by the protocol run.

The PAPI layer will then create a specific PAPI client that will provide the mechanisms needed for the user to execute the protocol in phase two. After creation of the PAPI client, the client will communicate with the PAPI client of the other user(s) to establish a PAPI communication link over the user specified communication channel (e.g., socket). This establishment will provide some handshaking to create a new session id for the protocol run and to verify that both players established the

same protocol. The clients will also create a template for message headers that contains the new session id, player ids, and a protocol id. The protocol id is automatically generated from the protocol specification (e.g., a secure hash of the specification text). In addition, the PAPI clients could create an SSL-like secure link if that feature is desired.

PAPI Communication Phase. During the second phase, the user invokes the PAPI client to send and receive specified messages of the protocol. To send a message, the user informs the PAPI client of the message number (as defined in the specification) and provides the contents of message fields. It is possible that some of the message fields, such as nonces or player ids, should be automatically inserted by the PAPI client and do not have to come from the user. The PAPI client then creates a message header from the header template by adding the message number and a checksum to the header. The message is then created, calling the specified cryptographic functions for cryptographic fields of the message. These functions may, for example, encrypt, sign or hash the data. Any data sent through these functions will have the header prepended to them to signify the context in which these operations occurred.

To receive a message, the user informs the PAPI client of the message number. The PAPI client receives the message from the other player, performs the necessary cryptographic functions on the data, verifies the message and send the results to the user. The PAPI client may have to verify signatures or hashes or decrypt portions of the message. The headers included in the cryptographic fields are compared with expected values and the user is informed if discrepancies occur. In addition, if the sender's PAPI client automatically inserted some fields into the message, the receiver's client may be able to automatically verify if the contents of those fields are correct (e.g., player ids are as expected, or returned nonces have the correct value).

If the protocol specification is rich enough, the PAPI client can also enforce specific mes-

sage dependencies, ensuring that no message is sent or received until all of the specified predecessors have been processed. In any case where the PAPI client detects a problem the user will be notified and the layer will take appropriate action, which may involve deletion of the offending message or even termination of the protocol run.

PAPI Termination Phase. The final phase of the PAPI layer is to provide termination of the protocol run. This termination may occur abnormally if the client error handling mechanism deemed that it was required or normally if the user requests a protocol termination.

3.2 Security of PAPI

The types of attacks that can occur against a protocol run can be classified by the behavior of the adversary and players. In this section we discuss these attacks and describe how the PAPI layer can assist in defending against these attacks. There are different aspects to the environment in which a cryptographic protocol executes; the behavior of the players and adversary play an important part in this environment. A taxonomy of this behavior is presented below. In this discussion we define the adversary as the collection of all entities working together to attack the protocol.

1. *Passive or Active Adversary.* A *passive adversary* allows the protocol run to proceed uninhibited but may store information for subsequent use. An *active adversary* may create, modify or remove contents of messages, or redirect them to different protocols runs. The active adversary can masquerade as one or more players in a protocol run.
2. *Honest or dishonest players.* The communicating parties in the protocol run may both be *honest*: executing valid instances of PAPI and using good security measures. Otherwise, some of the players may be *dishonest*: deliberately using messages of the current protocol run to attack another

protocol run. We assume the existence of at least one honest player in each protocol run and only analyze attacks that occur against the honest player(s). The differentiation between a dishonest player and the typical adversary is that a dishonest player is an adversary that can successfully complete some protocol without attacking or subverting a protocol run.

3.2.1 Attacking Only Honest Players

We will first look at the set of attacks that can occur when we have only honest players in the protocol run.

Passive Security Attack. This type of attack occurs when the adversary is passively monitoring traffic between the active participants and tries to break the protocol through cryptanalytical attacks on the message stream. The PAPI layer assumes the security of the underlying cryptographic services which are used to prevent this type of attack and hence does not address this type of attack.

Replay Attacks. We define a replay attack as one where the adversary uses messages from a prior run of a protocol to spoof a new protocol run. The adversary acts as a passive entity during the run of the prior protocol and only becomes active during the run of the attacked protocol. It has been documented since the early 1980's [13] that these types of attacks can occur and that timestamps, session ids or other "recency" indicators can be used to defeat these attacks. All cryptographic content in protocol messages run through PAPI automatically contain headers with a session id that prevents this type of attack.

Interleaving Attacks. We define an interleaving attack as one where the adversary actively uses messages from more than one concurrent protocol run to spoof at least one of the protocol runs. The adversary is active and may be actively participating in the protocol runs. These attacks can not succeed in PAPI

supported protocols due to the headers that contain protocol, message, player and session identifiers as suggested in [3, 6].

3.2.2 Attacks with Dishonest Players

A dishonest player can act as the adversary in any of the situations specified above. In addition, a dishonest player has the added advantage of gaining some level of trust with an honest user. A perfect example of this is Lowe's attack on the Needham-Schoreder protocol [14]. In this case, the dishonest user behaved honestly in the execution of one protocol run, but acted as an active adversary in another protocol run using messages from the first protocol. Other examples can be found in the literature on multi-protocols attacks [3, 4, 5, 11].

Although the dishonest player is able to control message content they send by emulating portions of the PAPI layer, they can not control the content of messages sent by the other players. Assume players A and E are executing protocol P1 and players E and B are executing protocol P2. Lets also assume that A and B are honest and that E is dishonest and that E will behave honestly in P1 and will try to attack P2. Successfully interleaving attacks discussed in the literature require that E use some message from P1 to spoof P2. However, since A is honest, all messages created by A for P1 will have headers denoting that fact. B, also being honest, will check these headers and detect the discrepancy, rejecting the message from E. This occurs even if A and B are actually the same user and P1 and P2 are the same protocol since PAPI includes session ids and message numbers in all headers.

4 Conclusion

In this paper we have discussed some of the concerns with cryptographic protocol design and implementation and some proposed solutions. What we have found in our examination of the literature is that there still exists

no solid engineering foundation for the development of secure cryptographic protocols. It is possible that the reason involves the complexity of the operating environment in which the protocol executes. We introduced a new protocol services layer that will act as an intermediary between the user and all cryptographic services and cryptographic protocol runs. The existence of this intermediary restricts the operating environment of the protocol and will permit the engineering and verification of secure cryptographic protocols. Our layer, implemented through the PAPI interface, provides an environment where the cryptographic content of the protocols will be immune to replay and interleaving attacks for both single and multi-protocol based attacks. Using this immunity as a base assumption, the design of these protocols become much more tractable. The engineer still has to design a protocol that is secure, but in the context of the automatic operations provided by the PAPI client.

References

- [1] E. Sneekenes. Exploring the BAN approach to protocol analysis. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 171–181, 1991.
- [2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 122–136, May 1994.
- [3] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In *Proc. 1997 Security Protocols Workshop*, pages 91–104, 1997.
- [4] J. Alves-Foss. Multi-protocol attacks and the public key infrastructure. In *Proc. National Information System Security Conference*, pages 566–576, October 1998.
- [5] J. Alves-Foss. Provably insecure mutual authentication protocols: The two-party symmetric encryption case. In *Proc. National Information System Security Conference*, October 1999.
- [6] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *International Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.
- [7] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – Crypto 93 Proceedings*, pages 232–249, 1993.
- [8] S. Blake-Wilson and A. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proc. 1997 Security Protocols Workshop*, pages 137–153, 1997.
- [9] Common security: CDSA and CSSM version 2. Technical Report C902, The Open Group, November 1999.
- [10] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [11] W. Tzeng and C. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Information Processing Letters*, 69(6):297–302, March 1999.
- [12] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 234–248, 1990.
- [13] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
- [14] G. Lowe. Breaking and fixing the Needham-Shroeder public-key protocol using FDR. In *Proc. TACAS*, pages 147–166. Springer-Verlag, 1996.