Fail-Stop Protocols: An Approach to Designing Secure Protocols

Li Gong

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, California 94025 U.S.A.
(gong@csl.sri.com)

October 20, 1994

Abstract. This paper presents a methodology to facilitate the design and analysis of secure cryptographic protocols. This work is based on a novel notion of a fail-stop protocol, which automatically halts in response to any active attack. This paper suggests types of protocols that are fail-stop, outlines some proof techniques for them, and uses examples to illustrate how the notion of a fail-stop protocol can make protocol design easier and can provide a more solid basis for some proposed protocol analysis methods.

1 Background and Motivation

In a distributed system, security depends heavily on the use of secure protocols such as authentication protocols (e.g., [21, 26]) and secure communication protocols (e.g., [4]). It is well known that such protocols can fail even if the underlying cryptosystems are sound and can have very subtle security flaws that are quite difficult to debug [6]. In fact, the protocol security problem is undecidable in that, given any protocol analyzer, there are protocols whose security the analyzer cannot decide.[1]

Recent years have seen notable efforts devoted to developing methods – theories, logics, formal methods, and tools – to facilitate the analysis of the security of cryptographic protocols (e.g., [11, 6, 17]). Although these results are significant, they are not yet satisfactory due to the following reasons. Methods based on searching, as the authors themselves pointed out [17], can find protocol design vulnerabilities to only those attacks that are explicitly modelled. Thus, failure to find a vulnerability by such a method does not mean that the protocol is secure, but merely that certain lines of attack are less likely to succeed. In addition, like software testing, searching is computation

[1] To relate this problem to the Turing machine halting problem, simply define a protocol that broadcasts all its secrets if the analyzer finds it secure and does nothing otherwise.

intensive.

Methods based on modal logic [6, 3], on the other hand, seem more conclusive in that their aim is to produce a proof of protocol security by deducing that certain protocol goals are achieved. However, such methods generally make a number of assumptions some of which cannot be justified by the methods themselves. Most of these, such as the assumption that one can identify one's own messages or that one can distinguish between a random string and a properly encrypted message, can be guaranteed by additional constraints on protocol specification or implementation. They can also be dealt with by logics with more complicated constructs (e.g., [13]).

The single most difficult assumption is that a secret remains secret during an execution of the protocol.[2] This assumption is paradoxical – whether a secret can remain secret may depend crucially on whether the protocol is secure, thus the assumption cannot be used to derive the security of the protocol unless a separate mechanism can be used to justify this assumption. One would hope that the searching approach can be used to complement the proof-based approach, but because of the unlimited ways in which an attack can take place, it is impossible to model all possible attacks (both smart and stupid attacks) that must include all types of attacks as well as all possible instances of these attacks.[3] Indeed, when Nessett [24] raised the difficulty with this assumption of secrecy, no satisfactory answer could be provided. However, it is probably unfair to say that the logic of Burrows, Abadi, and Needham is flawed – the logic's scope is explicitly defined not to cover the issue of secrecy [7]. None of the later extensions of this logic resolves this difficulty.

An earlier work by Dolev and Yao [11] proved that protocols using public-key cryptosystems [10] and having certain very rigid structures can automatically satisfy the secrecy assumption. However, the restrictions on the protocols are so strict – for instance, one can only append to a message – that the results are not widely applicable.

Given the above observations, we propose a new approach to designing secure protocols that is centered on a notion of *fail-stop protocols*. This notion is partly inspired by the work on fail-stop processors by Schlichting and Schneider [30]. They proposed the concept of a fail-stop processor, which, when failing, stops completely before any effect is visible to the outside world. Schneider also showed how to construct a fail-stop processor using Byzantine agreement [31]. A desirable result of this fail-stop behavior is that it is much easier to reason about fault-tolerant systems built with fail-stop processors, compared with processors that may have omission or Byzantine failures. Just as the notion of fail-stop processors helped to simplify the design and analysis of fault-tolerant systems, we show that the notion of fail-stop protocols helps to simplify the design and analysis of secure protocols.

More specifically, a fail-stop protocol automatically halts when there is any derivation from the designed protocol execution path. Consequently, the only difference between effects of passive attacks and active attacks is that the latter can cause early termination of a protocol execution. Thus, we need to analyze only the effect of passive attacks, and in particular, it is now much easier

[2]Abadi and Tuttle [3] in their new semantics of the BAN logic [6] tried to relax this assumption by assuming instead that a secret can be leaked but whoever processes it (maybe illegally) will not misuse it. This is logically sound, but does not reflect what happens in the real world.

[3]Some searching-based method, such as the NRL protocol analyzer [17], are said to have the potential to provide some proving capability, but no general proof methodology is currently available.

to conclude whether the secrecy assumption can be violated. One obvious benefit is that once we show that the secrecy assumption for a protocol holds, a proof of the protocol's security using the logical analysis method of Burrows, Abadi, and Needham (also called the BAN logic) will be much more convincing.

From another angle, just as algorithms or programs should be designed for their correctness to be easily proven [15, 28], security protocols should be designed so that their security can be proven with relative ease. The difficulties encountered by previous efforts of protocol analysis, in our view, can be to some extent attributed to the undisciplined ways in which a protocol can be designed (and then submitted for analysis). By imposing a few restrictions on the format the messages of a protocol can take, we can greatly reduce the types of protocols we have to deal with. The specific construction of fail-stop protocols presented in this paper can result in practical and usable protocols, and therefore the restrictions are not as limiting as one might have first thought.

In short, the advantage of our new approach can be seen in the following light. Proof-based methods are more favorable than searching-based methods in that the former need to capture only all types of attacks whereas the latter must in addition capture all (and possibly an infinite number of) instances of all types of attacks. Our approach's advantage is in excluding the feasibility of active attacks so that protocol analysis can focus on the remaining case – namely, passive attacks that may lead to information leakage – and provide a more solid basis and some simplifications for proofs of security.

In the rest of this paper, we first define (albeit informally) fail-stop protocols. Then, we discuss how to analyze the security of such protocols. After that, we describe how practical fail-stop protocols can be constructed and give examples. Finally, we discuss possible extensions and some directions for future work.

2 Fail-Stop Protocols and Their Analysis

We model a distributed system as a collection of processes which are spatially separated. They communicate with each other by exchanging messages. A protocol is a specification for the format and relative timing of the messages exchanged. A *cryptographic protocol* uses cryptographic mechanisms such as encryption and decryption algorithms to guarantee the integrity, the secrecy, the origin, the destination, the order, the timeliness, and ultimately the meaning of the messages. We assume that a protocol executes in steps or rounds.

Using Lamport's definition of causality [18], we can organize the messages of a protocol into an acyclic directed graph where each arc represents a message and each directed path represents a sequence of messages. If a message is altered in any way that is inconsistent with the protocol specification, then all those messages that are behind this altered message on some path in the graph (i.e., they are causally after the altered message) will not be sent. We must assume that the attacking party does not possess the encryption key with which the target message is encrypted; otherwise, a forgery may not be detectable.

Definition 1 (Fail-Stop Protocol) *A protocol is fail-stop if any attack on a message sent in one* *step will cause all causally-after messages in the next step or later not to be sent.*

Note that the definition is stated in a rather informal language. *This is intentional, as the rest of the discussion in this paper is also informal.* This way, the basic idea and the intuition can be more easily presented. It should not be a difficult matter to formalize the idea once it is accepted as appealing and useful. The following claim follows immediately from the above definition.

Claim 1 *Active attacks do no harm to a fail-stop protocol other than causing early termination.*

Therefore, we need to consider only passive attacks in which an adversary records messages and tries to compute secrets from them. Such attacks are well understood and we come back to this subject in the next section. Given Claim 1 and its implications, we can use the following proof methodology for a fail-stop protocol:

Phase 1.	Verify that the protocol is fail-stop.
Phase 2.	Validate that the secrecy assumption holds.
Phase 3.	Apply BAN-like logics.

Table 1: A proof methodology

In the following sections, we discuss these 3 phases of protocol analysis in more detail. We note here that the validation of the secrecy assumption sometimes can be used to disprove the security of a protocol by showing that the assumption does not hold. This usage is independent from the other two phases.

We first give an example protocol to show that even for fail-stop protocols that hold the secrecy assumption, the application of the BAN logic is still useful in finding design errors. As typical in the literature, we use $A \rightarrow B : x$ to denote that A sends message x to B, (x, y) to denote concatenation of x and y, $\{x\}_k$ to denote encryption of x with key k, and $\{x\}_k^-$ to denote decryption of x with key k.

In the following protocol, server S distributes a session key k to be shared between clients A and B. Each message includes the identities for the message sender, the recipient, a timestamp, and key k, and is encrypted with the key already shared between the server and the recipient.

1.	S $\rightarrow$ A:	$\{S, A, T_s, k, B\}_{k_{as}}$
2.	S $\rightarrow$ B:	$\{S, B, T_s, k\}_{k_{bs}}$

Table 2: BAN-like logics still useful for fail-stop protocols

It is easy to check that this protocol is fail-stop because any attack on either message will make the message undecipherable. However, the second message differs from the first in that the session key is not clearly associated with the identity of another party, thus B after receiving message 2 will not know with whom he shares the key k. Such design errors, which can have serious security implications, have been found elsewhere by using BAN-like logics [13].

2.1 Practical Fail-Stop Protocols

A way to verify that a protocol is fail-stop is to show that the protocol conforms to one of the known specifications of fail-stop protocols. To build up such a "library" of protocol specifications, we first give one of the simplest specifications of fail-stop protocols. For simplicity, we assume for the moment that only symmetric key cryptosystems (such as DES) are used, and every pair of communicating processes share a secret encryption key.

Claim 2 *A protocol is fail-stop if:*

1. *The content of each message has a header containing the identity of its sender, the identity of its intended recipient, the protocol 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. *A process discards all unexpected messages.*
4. *A process halts if an expected message does not arrive within a specified timeout period.*

Here a freshness identifier can be a timestamp (if clocks are assumed to be securely and reliably synchronized) or a nonce issued by the intended recipient. When a freshness identifier takes on a more complicated form, the rules for reasoning about freshness [6, 13] can be used to determine if the identifier is fresh with regard to the recipient. Basically, if x is deemed fresh and y cannot be computed (in a computationally feasible way) by someone without the knowledge of x, then y is also deemed fresh [13].

To see that Claim 2 is valid, we note that a message's header uniquely identifies the position of the message (e.g., within which protocol execution and which message of this execution). It is not possible to use this message elsewhere without modifying the message. However, since the message is encrypted with the key shared between its sender and recipient, no one else can make undetectable modifications without obtaining the key first. In particular, the message header provides sufficient redundancy so that any random modification of the message can be detected with an extremely high probability. Recently, Abadi and Needham collected a number of prudent engineering principles for designing authentication protocols [2]. The above specification of fail-stop protocols satisfies some of these principles.

Fail-stop protocols using public-key systems can be similarly formulated. The only difference is in the encryption of the message.

> *2. Each message is signed by the sender's private key. The message can then be optionally encrypted under the public key of the recipient.*

The order of signing and encrypting can be reversed in some situations. Also, it is not difficult to combine the above two formulations for fail-stop protocols using both types of cryptosystems, but we do not discuss this in further detail. Since the encryption key may uniquely identify the sender, thus, we can relax the requirement in that the sender identification need not be included in the message plaintext if the encryption itself is sufficient proof of the sender's identity.

Checking whether a protocol conforms to Claim 2 (thus being fail-stop) is easy. For example, the Nessett protocol [24] and a few smart-card protocols [1] are fail-stop.

Many published protocols are not fail-stop as we defined. One reason is that many designers try to be economical – they would want to send plaintext messages whenever they think it safe to do so. However, this kind of unguided (and rather ad hoc) "optimization" is easily one of the rich sources of security bugs. As Roger Needham remarked, one cannot foresee the consequences of being clever [20]. Nevertheless, examples have shown that guided optimizations can indeed identify and remove redundant data from messages [5].

Moreover, sometimes auxiliary data are sent in the clear. For example, the identity of the sender of an encrypted message is sent along when it helps the recipient to choose the correct key for decryption. Although protocols can be designed so that the amount of such auxiliary data is kept to a minimum, the presence of such data do not change the nature of a protocol being fail-stop. For example, if the sender identification is modified, then the recipient will choose an incorrect key and fail to decrypt the message. This modification will thus cause the recipient to halt, so the protocol is still fail-stop. Suppose we allow the existence of such auxiliary data, more published protocols can be viewed as fail-stop. They include the Denning-Sacco protocol [9], the Needham-Schroeder Public-key protocol [21], the Demonstration Protocol and the Enhanced Kerberos Protocol [12], the Wide-mouthed-frog protocol and the CCITT X.509 protocol [5], the Andrew Secure RPC [29], and the Private-key Certificate protocol [8]. The fact that there are many existing protocols that are fail-stop suggest that our formulations are not too limiting for practical applications.

2.2 Validating the Secrecy Assumption

In BAN-like logics, a secrecy assumption is that a data item is known only to a set of parties. Since active attacks simply halt the execution of a fail-stop protocol, an attacker is better off waiting for the protocol to complete and gathering as many messages as possible (to use for deducing secrets). Therefore, to check that no other party can obtain the item through attacks, our task is to decide that, given that one can record all messages exchanged during a protocol execution, can any party learn a particular data item by manipulating the messages (together with those data items the party already has in possession)?

This question is related to that of the "state of knowledge" [19] in that we need to find out what information an attacker has gathered by recording the execution of a protocol.[4] The notion and rules of "possession" proposed in the GNY logic [13] can be applied directly for this purpose. Briefly, given a set of formulas (or data items) that an attacker is thought to possess (through

[4] We can use the same method to check how much information can be gathered by recording multiple executions of a protocol or of a number of protocols. We do not further discuss this issue here.

recording or other means), possession rules can be used to derive all formulas the attacker can possess. Suppose a formula can be the concatenation of sub-formulas, for example, message *hello, I am alive* is consists of sub-formulas *hello* and *I am alive*. The possession rules can be summarized as the following:

- possession of a formula implies possession of every sub-formulas contained in the formula.
- possession of all sub-formulas contained in a formula implies possession of the formula.
- possession of a data item implies possession of a function of the data where the function is computationally feasible to compute.
- possession of a formula and an encryption implies the possession of the formula encrypted with the key.
- possession of an encrypted formula and the appropriate encryption key implies the possession of the formula after decryption.

We later see how these rules can be applied to real examples. It is suffice to note here that although it seems that these rules can be applied indefinitely to add new formulas to an attacker's possession, in practice there are suitable guidelines as to when to terminate when we want to find out if *a particular secret* can be in the attacker's possession. For example, if all relevant formulas have at most 2 levels of nested encryption, then it is futile to try to encrypt beyond two levels. Also, if all formulas exchanged in the protocol are either plaintext or encrypted, then it is useless to apply decryption to plaintext formulas.

2.3 Applying BAN-like Logics

The last phase in the proof methodology of fail-stop protocols is to apply the BAN-like logics. As we have shown in the beginning of this section, fail-stop protocol can still have subtle errors that can be captured using BAN-like logics.

It is important to note that a fail-stop protocol of the form defined in Claim 2 automatically satisfies most of the assumptions necessary to apply the BAN logic. For example, the sender information ensures that one can identify one's own messages. Also, there is sufficient redundancy in an encrypted message. Moreover, all messages are fresh. Thus, some postulates – especially those on freshness and recognizability – are no longer needed here (though they may be needed to verify that a protocol is fail-stop, as we pointed out earlier), and other postulates can be simplified. This advantage can be seen from another point of view. The logic of Gong, Needham, and Yahalom (GNY) [13] has a number of extensions to BAN so that most of the assumptions in BAN are handled explicitly in GNY. If we use the GNY logic to analyze a fail-stop protocol, we no longer need constructs such as "not-originated-here" and "recognizability", and thus the complexity of the GNY logic can be greatly reduced.

The definition of fail-stop protocols does not satisfy the crucial assumption necessary for BAN-like logics – that all secrets remain secret during protocol execution. Thus we need phase 2 in the proof methodology. To validate the secrecy assumption, we can divide secrets into two types. The first type of secrets include those that are used as keys to encrypt messages but are not sent as message content. Clearly these keys cannot be compromised, on the assumption that cryptosystems are strong and cryptanalysis is infeasible. (And this is the basis for requirement 2 in Claim 2.) The other type includes secrets that are sent as message contents, and it is this type that our validation process deals with.

2.4 Protocol Analysis in Stages

In authentication protocols, it is common that a secret, a session key, is first sent by the server to clients, who later use it in handshake messages. It appears that we cannot cast such protocols as fail-stop because, paradoxically, in phase 1 we need to assume that encryption keys are secret while only phase 2 can we validate this assumption (about the session key). One way to overcome this difficulty is to analyze such a protocol in stages. For example, given the following protocol (our earlier example, with the design error corrected, plus two handshake messages):

1.	S $\rightarrow$ A:	$\{S, A, T_s, k, B\}_{k_{as}}$
2.	S $\rightarrow$ B:	$\{S, B, T_s, k, A\}_{k_{bs}}$
3.	A $\rightarrow$ B:	$\{A, B, T_a\}_k$
4.	B $\rightarrow$ A:	$\{B, A, T_b\}_k$

Table 3: Analysis in stages

we can first analyze the protocol without the handshake messages, i.e., the protocol in Table 2 (with an additional data item B in message 2), which includes only messages 1 and 2. After the 3 phases of proof methodology, if the verdict is that the protocol is secure, then we have proven that in the second stage of the full protocol (in Table 3), i.e., messages 3 and 4, the encryption keys are securely shared between the message sender and recipient. Thus we can continue and apply the whole cycle of methodology to the full protocol. Obviously, a more complicated protocol can be analyzed in more than two stages. In the extreme case, we can have step-wise or message-wise verification.

To summarize, the vital advantage of designing fail-stop protocols is that we can use the "possession" rules to verify that the secrecy assumption holds for a protocol. This phase adds considerable credibility to the claim that a protocol has been analyzed by BAN like logics to be secure.

2.5 Complex Protocols

Security protocols in a distributed systems often necessarily interact with each other directly or indirectly. For example, a complex protocol may use simplier protocols as building blocks, in which case an important question is whether we can reduce the analysis of the complex protocol to that of the building blocks in isolation. Moreover, a protocol is usually designed and implemented, and its security analyzed, independently of other protocols. Therefore, a crucial question is whether the deployment of one protocol invalidates the security claims of another, possibly in such a way that both protocols need to be modified in order to coexist securely.

If all the individual protocols or building blocks are fail-stop, then the analysis of the overall complex protocol can indeed be built on analysis of the individual protocol. For example, clearly a protocol consisting of two fail-stop protocols executed one after another is fail-stop. Also, a protocol consisting of two fail-stop protocols running in parallel is fail-stop, even the two protocols share variables.

Claim 3 *The sequential and parallel composition of fail-stop protocols is also fail-stop.*

For such sequential or parallel protocol compositions [14], the analysis of secrecy (see Section 2.2) is insensitive to the order of the interleaving messages. However, the correctness of the overall protocol as analyzed by BAN-like logics may depend on a particular interleaving order, in which case this global ordering should be part of the specification of the overall protocol.

3 Examples and Discussion

In this section, we give examples to demonstrate how the proof methodology proposed in the previous section can be used in practice. We examine two published protocols, one being shown not to satisfy the secrecy assumption, and the other being shown secure. We also outline some generalizations of fail-stop protocols, such as fail-safe protocols, which help to expand the scope of protocols that can be analyzed using our new approach.

3.1 Example 1: the Nessett Protocol

Our first example is the Nessett protocol [24], as follows.

1.	A $\rightarrow$ B:	$\{n_a, k_{ab}\}^{-}_{k'_a}$
2.	B $\rightarrow$ A:	$\{n_b\}_{k_{ab}}$

Table 4: Nessett protocol

In message 1, A "distributes" a key k_{ab} to be shared between A and B. The message includes a nonce n_a which B regards as fresh, and is signed with A's private key k'_a. Message 2 is a handshake message which includes a nonce n_b which A regards as fresh.

First, we can temporarily ignore the required inclusion of sender and recipient identification in the messages since it is unrelated to our discussion here. One can easily see that this protocol conforms to all other aspects of our definition in Claim 2, so the protocol is basically fail-stop.

The implicit secrecy assumption is that only A and B can obtain key k_{ab}. To check this assumption, we assume that the attacker can record both messages and can have access to A's public key k_a (which is generally known to the public). Now we can use the possession rules [13] as follows:

$$\frac{\text{poss}(\{n_a, k_{ab}\}^{-}_{k'_a}) \text{ AND poss}(k_a)}{\text{poss}((n_a, k_{ab}))}$$

and then

$$\frac{\text{poss}((n_a, k_{ab}))}{\text{poss}(k_{ab})}$$

In other words, an attacker can obtain k_{ab}. Therefore, we can prove that this protocol violates the secrecy assumption. Such a proof means that the protocol does not satisfy one of the most important assumptions in BAN like logics, thus the subsequent analysis using BAN is meaningless and cannot be taken as a counter example to show that BAN is flawed. Nevertheless, this protocol did raise the legitimate question of how to validate the secrecy assumption for any given protocol. Our concept of fail-stop protocol aims precisely to fill in this gap.

We emphasize that this validation procedure (using the possession rules) can be used to disprove the security of a given protocol (due to leaking of secrets) whether the protocol is fail-stop or not. However, as a crucial stepping stone in proving the security of a protocol, this procedure can be applied only to fail-stop protocols, because in protocols that are not fail-stop, active attacks can be successful so that secrets can be leaked in ways not detectable by this procedure.

3.2 Example 2: the Needham-Schroeder Public-Key Protocol

Our second example is the Needham-Schroeder public-key protocol [21], and it works as follows.

1.	A $\rightarrow$ S:	A, B
2.	S $\rightarrow$ A:	$\{k_b, B\}^{-}_{k'_s}$
3.	A $\rightarrow$ B:	$\{n_a, A\}_{k_b}$
4.	B $\rightarrow$ S:	B, A
5.	S $\rightarrow$ B:	$\{k_a, A\}^{-}_{k'_s}$
6.	B $\rightarrow$ A:	$\{n_a, n_b\}_{k_a}$
7.	A $\rightarrow$ B:	$\{n_b\}_{k_b}$

Table 5: Needham-Schroeder public-key protocol

In message 2, S signs a message certifying that B's public key is k_b. In message 3, A sends a nonce to B encrypted with B's public key. In message 5, S signs a message certifying that A's public key is k_a. A and B then complete handshake.

We contrast this example with the previous example and show how to prove that the secrecy assumption holds. Assume that an attacker can record all messages. Moreover, assume that the attacker possesses all the public keys, especially the public key of S, k_s. Using the possession rules, the attacker can only do encryption on the messages. It is obvious (and can be easily made mechanically decidable) that it is fruitless to encrypt messages other than 2 and 5, because further encrypting these messages (i.e., messages 1, 3, 4, 6, and 7) cannot help decryption in the future. By encrypting messages 2 with S's public key, or in the language of the possession rules, we have:

$$\frac{\text{poss}(\{k_b, B\}^{-}_{k'_s}) \text{ AND poss}(k_s)}{\text{poss}((k_b, B))}$$

and then

$$\frac{\text{poss}(k_b, B))}{\text{poss}(k_b)}$$

The attacker has learned nothing new – B's public key is already public information. Similarly, encrypting message 5 does not lead to new information. Since no other encryption of a signature or decryption of an encrypted message is possible, the validation procedure terminates, and we are convinced that the secrecy assumption holds. Note that the fact that such an argument is possible is because the protocol in question is fail-stop, so that no active attacks can be successful and we can study the protocol in its fixed form.

Even if we assume that the attacker is an insider, say A, we can still show that A cannot gain possession of B's private key. An analysis of this protocol using the BAN logic is contained in [5, p.33].

3.3 Fail-Safe Protocols and Other Generalizations

Some protocols are not fail-stop but can apparently be analyzed in the same way as fail-stop protocols. For example, when a message arrives whose freshness the recipient cannot decide, it is safe to let the recipient respond with a nonce. We can call such protocols *fail-safe*. Many existing protocols are fail-safe in nature, including the revised Needham-Schroeder protocol [22], the Otway-Rees protocol [27], the GLNS nonce protocol [12], the Neuman-Stubblebine protocol [25], and the protocol for multiple authentication of Kehne et al. [16].

Some may argue that the rigid structural requirement in Claim 2 is still too limiting, and indeed it is possible that some such defined protocols may transmit unnecessary data. *After* we are satisfied that a protocol is secure, we can optimize the protocol while maintaining its security. For example, for a message that is encrypted with a public key, the recipient identity in the message content can be safely omitted, as long as the remaining message still contains enough redundancy, because no one else can correctly decrypt this message. Similarly, a message that is signed with a private key, the sender identification in the message content can be omitted because no one else is able to sign the message (again, as long as the rest of the message contains enough redundancy). Beside such simple rules of optimization, the BAN logic is supposedly to be quite good at finding redundancies in messages [6], so it can also be used to optimize a protocol. Note that such optimizations are guided by security analysis so that they do not weaken protocol security. This is different from ad hoc economical measures present in many published protocols.

4 Summary and Future Work

The general idea of our work reported in this paper is based on the well known observation that if a program is well structured then its proof of correctness is likely to be easier and simpler. Therefore, similarly, if a protocol is well designed so that its conformation to certain guidelines ensures that certain security properties are (automatically) satisfied, then its proof of security is likely to be easier and simpler. In particular, inspired by the work on fail-stop processors [30, 31], we defined the notion of a fail-stop protocol and illustrated how the important secrecy assumption necessary for applying the BAN logic can be easily validated with the possession rules in the GNY logic for such a restricted class of protocols. The availability of this validation phase adds significant credibility to the (positive) outcome of an analysis using the BAN logic. Our discussion also shows that many existing protocols are fail-stop in spirit so that our restriction is not too limiting, especially given the possible extensions and optimizations.

For future work, obviously one direction is to formalize this work, possibly embedding it in some protocol specification and theory-proving environment. Another direction is to investigate other possible constructions of fail-stop protocols and generalizations of the concept of a fail-stop protocol that can facilitate the design and analysis of cryptographic protocols. Moreover, by imposing a well defined structure, we have removed the threat of active attacks and reduced our tasks to examining the security of a protocol under passive wiretaps. It will be very interesting to see how to define a hierarchy of attacker models and show how to convert a protocol for one model into another. This is analogous to the work on converting protocols between various fault models [23].

Acknowledgement

Fred Schneider of Cornell University, in a private correspondence of March 20, 1992, gave me early encouragement to clarify and develop my initial ideas of fail-stop protocols. These ideas and some later development were informally presented at the First and Second Cambridge Workshops on Security Protocols, University of Cambridge, England, April 1993 and April 1994.

References

[1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and Delegation with Smart-cards. Technical Report 67, DEC System Research Center, Palo Alto, California, October 1990.

[2] M. Abadi and R.M. Needham. Prudent Engineering Practice for Cryptographic Protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–136, Oakland, California, May 1994.

[3] M. Abadi and M. Tuttle. A Semantics for a logic for Authentication (Extended Abstract). In *Proceedings of the ACM Symposium of Principles of Distributed Computing*, pages 201–216, January 1991.

[4] A.D. Birrell. Secure Communications Using Remote Procedure Calls. *ACM Transactions on Computer Systems*, 3(1):1–14, February 1985.

[5] M. Burrows, M. Abadi, and R.M. Needham. A Logic for Authentication. Technical Report 39, DEC System Research Center, Palo Alto, California, February 1989. Revised version of February 22, 1990.

[6] M. Burrows, M. Abadi, and R.M. Needham. A Logic for Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[7] M. Burrows, M. Abadi, and R.M. Needham. Rejoinder to Nessett. *ACM Operating Systems Review*, 24(2):39–40, April 1990.

[8] D. Davis and R. Swick. Network Security via Private-Key Certificates. *ACM Operating Systems Review*, 24(4):64–67, October 1990.

[9] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[10] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–65, November 1976.

[11] D. Dolev and A.C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.

[12] L. Gong, T.M.A. Lomas, R.M. Needham, and J.H. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.

[13] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 234–248, Oakland, California, May 1990.

[14] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1985.

[15] C.B. Jones. The Search for Tractable Ways of Reasoning about Programs. Technical Report UMCS-92-4-4, Department of Computer Science, University of Manchester, England, March 1992.

[16] A. Kehne, J. Schonwalder, and H. Langendorfer. A Nonce-Based Protocol for Multiple Authentications. *ACM Operating Systems Review*, 26(4):84–89, October 1992.

[17] R.A. Kemmerer, C. Meadows, and J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, October 1993. To appear.

[18] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[19] M. Merritt and P. Wolper. States of Knowledge in Cryptographic Protocols. Unpublished manuscript, 1985. An earlier version of this work appeared as R. DeMillo, N. Lynch, and M. Merritt. Cryptographic Protocols. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, May 1982, pages 383–400.

[20] R.M. Needham. Denial of Service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153, Fairfax, Virginia, November 1993.

[21] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[22] R.M. Needham and M.D. Schroeder. Authentication Revisited. *ACM Operating Systems Review*, 21(1):7, January 1987.

[23] G. Neiger and S. Toueg. Automatically Increasing the Fault-Tolerance of Distributed Systems. Technical Report GIT-ICS-89/01, Georgia Institute of Technology, Atlanta, Georgia, January 1989.

[24] D.M. Nessett. A Critique of the Burrows, Abadi, and Needham Logic. *ACM Operating Systems Review*, 24(2):35–38, April 1990.

[25] B.C. Neuman and S.G. Stubblebine. A Note on the Use of Timestamps as Nonces. *ACM Operating Systems Review*, 27(2):10–14, April 1993.

[26] B.C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.

[27] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *ACM Operating Systems Review*, 21(1):8–10, January 1987.

[28] J. Rushby. Formal Methods and the Certification of Critical Systems. Technical Report SRI-CSL-93-07, Computer Science Laboratory, SRI International, Menlo Park, California, November 1993.

[29] M. Satyanarayanan. Integrating Security in a Large Distributed System. *ACM Transactions on Computer System*, 7(3):247–280, August 1989.

[30] R.D. Schlichting and F.B. Schneider. Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *ACM Transactions on Computing Systems*, 1(3):222–238, August 1983.

[31] F.B. Schneider. Byzantine Generals in Action: Implementing Fail-Stop Processors. *ACM Transactions on Computing Systems*, 2(2):145–154, May 1984.