Fail-Stop Protocols: An Approach to Designing Secure Protocols

Li Gong[*]   Paul Syverson[†]

Abstract

We present a methodology to facilitate the design and analysis of secure cryptographic protocols. We advocate the general approach, and a new avenue for research, of restricting protocol designs to well-defined practices, instead of ever increasing the complexity of protocol security analysis mechanisms to deal with every newly discovered attack and the endless variations in protocol construction. In particular, we propose a novel notion of a fail-stop protocol, which automatically halts in response to any active attack that interferes with protocol execution, thus reducing protocol security analysis to that of passive attacks only. We suggest types of protocols that are fail-stop, outline some proof techniques for them, and use 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 available 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., [22, 25]) 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. We can reduce this problem to the Turing machine halting problem by simply defining a protocol that broadcasts all its secrets if the analyzer finds it secure and does nothing otherwise.

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., [9, 6, 19]). Although these results are significant, they also have limitations.

Automated and semi-automated methods based on algebra and state transitions model protocols at a fairly detailed level [19]. This can make them difficult to apply and their application computationally intensive and intensive in human labor. Some of these methods are designed for searching out vulnerabilities rather than proving protocols secure. The 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.

Methods based on modal logic [6, 2], 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., [14]).

The single most difficult assumption is that a secret remains secret during an execution of the protocol.[1] When Nessett raised the difficulty with this assumption of secrecy [24], no satisfactory answer could be provided, although it is probably unfair to say that the logic of Burrows, Abadi, and Needham (the BAN logic) is flawed because the logic's scope is explicitly defined not to handle the issue of secrecy [7]. Nevertheless, this secrecy assumption is paradoxical in that 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 justify this assumption. None of the later published extensions of this logic resolves this difficulty. Some of these logics have a model-theoretic semantics [2, 33, 34] and can be

[*]SRI International, Computer Science Laboratory, Menlo Park, California 94025, U.S.A. Email at gong@csl.sri.com.

[†]Center for High Assurance Computer Systems, Naval Research Laboratory, Washington, DC 20375, U.S.A. Email at syverson@itd.nrl.navy.mil.

---

[1]Abadi and Tuttle [2] in their new semantics of the logic of Burrows, Abadi, and Needham [6] relaxed this assumption by assuming instead that a secret can be leaked but whoever possesses it (maybe illegally) will not misuse it. This is logically sound even though it does not reflect what usually happens in the real world.

Report Documentation Page

1. REPORT DATE **1995**	2. REPORT TYPE	3. DATES COVERED **00-00-1995 to 00-00-1995**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Fail-Stop Protocols: An Approach to Designing Secure Protocols	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Center for High Assurance Computer Systems,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**12**	

used to directly examine the truth of such assumptions [32, 33]. However, such analysis is not formal, and to guarantee that it is comprehensive would seem to involve methods as intensive as those mentioned above [19]. An earlier work by Dolev and Yao [9] proved that protocols using public-key cryptosystems [8] 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 novel notion of *fail-stop protocols*. This notion is partly inspired by the work on fail-stop processors by Schlichting and Schneider [27]. 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 [28]. 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.

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 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 logical analysis using the BAN method and its variations (usually referred to as BAN-like logics) will be much more convincing.

From another angle, just as algorithms or programs should be designed for their correctness to be easily proven [17, 26], 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.

To summarize, the advantage of our new approach can be seen in the following light. Most current research in cryptographic protocol analysis is focused on extending and enhancing the existing methods in order to analyze ever more complicated features and unexpected attacks. However, the logical methods are generally easy to apply but rely on assumptions that are hard to justify, while search-based methods are difficult and computationally intensive. Therefore, we advocate an alternative approach where protocol designs should be made more restricted, disciplined, and well structured. We expect that such well-designed protocols can be analyzed by existing (or even simpler) methods and hope that this approach opens a new avenue for research.

As a starting point within this general approach, we propose a novel notion of fail-stop protocols, which has the advantage of 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. We note that the analogy with fail-stop processors [27] is more in spirit than in practice. For example, in fault tolerance, it has been relatively successful to classify faults into a handful of ordered (in terms of difficulty) categories. These faults are generally well understood and there are well-defined methods to protect against them. In computer security, however, and in protocol security in particular, characterizing all types of threats completely and usefully is non-trivial. This is because new types of attacks are still being discovered, the relationships between attacks and the combined effects of them are unclear, and there is no consensus that all known classes of attacks have satisfactory solutions. Therefore, when a protocol is claimed to be secure, the claim is implicitly against an assumed set of threats and is often dependent on implicit assumptions about the nature of protocols or messages that may be violated in the presence of an unforeseen type of attack [18].

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 to construct practical fail-stop protocols. Finally, we discuss some extensions and directions for future work.

2 Fail-Stop Protocols and 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 [20], 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. In a fail-stop protocol, if a message actually sent is in any way 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 interfering with a message sent in one step will cause all causally-after messages in the next step or later not to be sent.*

We sometimes also conveniently call such a message fail-stop. Note that the definition is stated in a rather informal language. This is intentional so that 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.

Before describing and justifying the following claim, we must first limit what we mean by an active attack. These are attacks that involve the capturing and/or inserting of messages (including plaintext, whole encrypted, signed, and pieces of messages). Specifically, we assume that message integrity is protected within an encryption, signature, or some other mechanism. Thus, for example, attacks utilizing both the protocol structure and the cryptographic algorithm to allow message splicing, and integrity attacks based on the mode of DES operations in encrypting certain protocol messages, are assumed to have been independently addressed [30, 31].

Claim 1 *Active attacks cannot cause the release of secrets within the run of a fail-stop protocol.*

Claim 1 then follows immediately from the definition of a fail-stop protocol, because active attacks do not cause more (or different) messages to be sent; so an attacker using active attacks cannot obtain more secrets than one using passive eavesdropping. More generally, since an active attack will cause a fail-stop protocol to halt, in a fail-stop protocol no principal will ever produce encryptions or any other computations on data from a message that was not entirely legitimate. Therefore, we need to consider only passive attacks in which an adversary records messages and tries to compute secrets from them. Such passive attacks (and protection measures against them) are much better understood than active attacks, and we come back to this subject in the next section.

Note that a legitimate principal might mount an attack using a protocol to cause another to generate ciphertext based on known, chosen, or recognizable plaintext. For example, in a three party protocol, the attacker could cause an indefinite number of ciphertexts to be sent from one of the other parties to the third by legitimately initiating the protocol. These texts might then be useful for breaking a key based on ciphertext-plaintext pairs. Fail-stop protocols by themselves prevent only the generation of messages due to deviations from a protocol's intended execution. They cannot prevent attacks based on proper executions of the protocol. We assume that other mechanisms have been employed to prevent these attacks and/or that they have been independently shown to be of no use.

Given Claim 1 and its implications, we suggest the following proof methodology for a fail-stop protocol, as shown in Table 1. This particular methodology aims to put model-logic based methods on a stronger footing in the sense that we have removed the paradoxical secrecy assumption in BAN-like logics by independently validating it. Other proof methodologies may become available later, and they may aspire to proving different or additional kinds of security properties that are deemed important in cryptographic protocol analysis and verification.

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

Table 1: A proof methodology for fail-stop protocols

In the following sections, we discuss these three phases of protocol analysis in more detail. Checking the validity of the secrecy assumption sometimes can be used to disprove the security of a protocol by show-

ing that the assumption does not hold. This usage is independent from the other two phases. Also, even for fail-stop protocols that sustain the secrecy assumption, the application of BAN-like logics is still useful in finding protocol design errors. We will give an example to demonstrate this presently, after Claim 2.

2.1 Practical Fail-Stop Protocols

One 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 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.*

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, 14] 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 [14]. Note that a protocol consisting of two or more messages based entirely on nonces cannot be fail-stop because the first message cannot be fresh. We can extend the concept of fail-stop protocols to fail-safe protocols to better treat nonce-based protocols, as we discuss in Section 4.

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, it is possible to guarantee that, by proper encoding of the header information under the encryption algorithm, the message header provides sufficient redundancy so that any random modification of the message can be detected with an extremely high probability. We assume that this correct encoding can and will be independently verified. Recently, Abadi and Needham collected a number of prudent engineering principles for designing authentication protocols [1]. The above specification of fail-stop protocols satisfies some of these principles. Requirements of *robustness* and *explicitness* [3] are related to similar ideas.

We now give an example protocol that satisfies Claim 2 and is thus fail-stop. Nonetheless, it still has design flaws that can be revealed by a BAN-like logic. As typical in the literature, we use $A \to B : x$ 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[2] of x with key k.

In the following protocol as shown in Table 2, 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, a protocol identifier and its version number, P, a message sequence number, and session key k, and is encrypted with the key already shared between the server and the recipient.

1.	S → A:	$\{S, A, T_s, P, N, k, B\}_{k_{as}}$
2.	S → B:	$\{S, B, T_s, P, N+1, k\}_{k_{bs}}$

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

This protocol clearly satisfies Claim 2 for processes that satisfy clauses 3 and 4 therein. (Note that process behavior pertaining to clauses 3 and 4 cannot be specified in this type of protocol description, which is why we assume them separately.) The protocol is thus fail-stop. 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

---

[2]We are assuming the use of a public key system in which decryption and signing amount to the same operation, e.g., RSA. Other systems might require these to be notationally distinguished.

the key k. Such design errors, which can have serious security implications, have been found elsewhere by using BAN-like logics [14].

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.

Here we assume that the sender's private key and the recipient's public key do not cancel each other and thus reveal plaintext. In the case that the sender and the recipient are the same party, we assume that the message is sent via its local circuit and is not revealed to the outside. The order of signing and encrypting can be reversed in some situations, but in all cases, it is prudent not to encrypt or sign purely random messages. We will come back to this point later.

Also, it is not difficult to combine the above two formulations for fail-stop protocols using both types of cryptosystems, but we do not further discuss this. When one can assume that encryption is sufficient proof of the sender's identity (e.g., when the encryption key uniquely identifies the sender), we can relax the requirement that the sender identification be included in the message plaintext. Checking if a protocol conforms to Claim 2 (thus being fail-stop) is easy.

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 the late Christopher Strachey once remarked (retold by Roger Needham in his invited talk at ACM CCS-1, November 1993), one cannot foresee the consequences of being too clever. 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. The presence of such data does 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.

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, given that one can record all messages exchanged during a protocol execution, our task is to determine whether any party can 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" [21] in that we need to find out what information an attacker has gathered by recording the execution of a protocol.[3] The notion and rules of "possession" proposed by Gong, Needham, and Yahalom (the GNY logic [14]) can be applied directly for this purpose. Briefly, given a set of formulas (or data items) that an attacker is thought to possess (through 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 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. We simply 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

---

[3] 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.

are suitable guidelines as to when to terminate investigating 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 suggested 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 GNY logic 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 with this type that our validation process deals in phase 2.

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

Table 3: Analysis in stages

we can first analyze this protocol without the handshake messages, i.e., the protocol in Table 3 (with data item A now included in message 2) but including 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, one vital advantage of designing fail-stop protocols is that we can use the "possession" rules in the GNY logic (among or in addition to other mechanisms) to verify that the secrecy assumption holds for a protocol. This phase adds considerable credibility to the claim that a protocol has been analyzed to be secure by the application of BAN-like logics.

2.5 Complex Protocols

Security protocols in distributed systems often necessarily interact with each other directly or indirectly. For example, a complex protocol may use simpler 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 [15]. Moreover, a protocol is usually designed and implemented, and its security analyzed, independently of other protocols. Therefore, another crucial question is whether the deployment of one protocol invalidates the security conditions and claims of another, possibly in such a way that both protocols need to be modified in order to coexist securely [11].

Unfortunately being a fail-stop protocol is not trivially composable. To see this, observe that any one

message protocol is automatically fail-stop regardless of the message structure. If being fail-stop were a composable property, then a simple induction would show that all protocols of any length composed of any messages from anyone to anyone are fail-stop. Fortunately it is easy to define an extensible fail-stop protocol property that is sequentially and parallelly composable [16].

Definition 2 (Extensible Fail-Stop Protocol)
In a given protocol, we call a message "last" if no message in the protocol is causally after that message. A protocol is extensible fail-stop if adding any last message to the protocol results in a fail-stop protocol.

A protocol can contain more than one "last" message. For example, the protocol of Table 3 contains two messages both of which are last.

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

When we compose protocols, assumptions (e.g., those about the operating environment) of the component protocols must all be satisfied. If these assumptions conflict with each other, then protocol composition does not make sense because the composed protocol may not be secure or even feasible to implement.

We justify Claim 3 with two observations. First, any one or more messages placed causally before any message in a fail-stop protocol will either be ignored by the protocol or cause it to halt. Thus, the result is still a fail-stop protocol. Second, suppose that sequentially composing part or all of an extensible fail-stop protocol, $P1$, after a last message in an extensible fail-stop protocol, $P2$, were to yield a protocol that was not fail-stop. By definition, this could not be due to a message immediately after a last message of $P2$. And, if it were because of any later message, then, by our first observation, $P1$ would not have been fail-stop as was assumed.

If all individual protocols or building blocks are extensible fail-stop, then the analysis of an overall complex protocol can indeed be built on analysis of the individual protocols. For such sequential or parallel protocol compositions, 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.[4] While many fail-stop protocols are not

extensible fail-stop, extensible fail-stop protocols are not significantly harder to design than fail-stop protocols. For example, any protocol meeting the requirements of Claim 2 is extensible fail-stop.

3 Applying Our Methodology

In this section, we demonstrate the use of the proof methodology proposed in the previous section. We examine three published protocols, one being shown not to satisfy the secrecy assumption, and the other two being shown secure.

3.1 Example 1: the Nessett Protocol

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

1.	A → B:	$\{n_a, k_{ab}\}_{k'_a}^{-}$
2.	B → 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,[5] and is signed with A's private key k'_a. Message 2 is a handshake message which includes a nonce n_b that A regards as fresh.

This protocol is not fail-stop in its published form. For example, if the first message were diverted by an intruder to C, C would produce his own nonce, and encrypt it with k_{ab} (assuming C to be an honest functioning principal for the protocol). To make this protocol fail-stop, suppose "B" is added to the plaintext signed by A. Assuming that no other protocol these principals might run has a message (or a piece of a message) of the same form as our modified first message, the new protocol would be fail-stop. (If this assumption cannot be sustained, protocol identifiers and version numbers may be added as well.)

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 usually assumed to be public knowledge). Now we can use the possession rules [14] as follows. Here poss(x) denotes the possession of formula x.

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

---

[4]BAN analysis is very sensitive to message ordering, and Snekkenes first made explicit the significance of this requirement using counter-examples [29].

[5]We are aware that a nonce, as opposed to a timestamp, is typically regarded as fresh only by the principal who generates it. Nonetheless, we follow the original description [24].

and then

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

In other words, an attacker can obtain k_{ab}, and thus this protocol violates the secrecy assumption. Such a proof (of insecurity) means that the protocol does not satisfy an important assumption in BAN-like logics, thus the subsequent analysis using BAN is meaningless. 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. Model theoretic semantics (e.g. [2]) was presented as an approach to examining protocol assumptions, including the secrecy assumption in this protocol [32]. Yet another attempt uses a notion called terminating protocols [29]. Of these approaches to analyzing the Nessett protocol, using the possession rules of GNY appears to yield desired results with the simplest analysis.

We reemphasize 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. Thus, the procedure worked on this protocol and would have worked whether or not the changes necessary to make it fail-stop had been made. However, to be used 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 Wide-mouthed-frog Protocol

Our second example is the wide-mouthed-frog protocol [5]. It proceeds as shown in Table 5.

| 1. | A → S: | $A, \{T_a, B, k_{ab}\}_{k_{as}}$ |
| 2. | S → B: | $\{T_s, A, k_{ab}\}_{k_{bs}}$ |

Table 5: Wide-mouthed-frog protocol

In message 1, A sends a timestamp, B's name, and a session key to an authentication server S, all encrypted with a key they share. In message 2, S sends the session key to B along with a timestamp of his own, and A's name.

Like the Nessett protocol, this protocol is almost, but not quite, fail-stop as published. If an attacker were to prepend a plaintext B to the second message and send it immediately back to the server, the server would treat this as a first message in a new protocol run. In the case of this protocol, the inclusion of just one more bit within the encrypted portions to differentiate the messages is sufficient to insure being fail-stop. The only further assumption we need in this regard is that only in runs of this protocol are messages with the form of message 1 received by the server. (Thus protocol identifiers and version numbers are unnecessary.) Even a dishonest principal can only initiate a legitimate key exchange if this assumption is true. Again, if there is danger of this assumption failing, a protocol identifier and its version number can be added.

It is easy to see that this protocol satisfies the secrecy assumption since eavesdropping yields no secrets unless one possesses the keys shared between principals and server. A BAN analysis of this protocol showed that the protocol is secure [5, p. 26]. Our validation of the secrecy assumption puts that analysis on a much stronger footing.

3.3 Example 3: the Needham-Schroeder Public-Key Protocol

Our third example is the Needham-Schroeder public-key protocol [22], and it works as shown in Table 6. Like the last two, it is not fail-stop as given. However, we will not suggest ways to make it fail-stop; further below we will show that it can be made to satisfy another property, being fail-safe, that justifies the application of our methodology.

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

Table 6: 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.

This protocol is clearly not fail-stop; the first and fourth messages could be sent by anyone to anyone at any time. As we noted above, protocols that are not fail-stop are subject to active attacks, in which case validating the secrecy assumption is not helpful. Nonetheless, we will show below that it satisfies another weaker property that makes the validation procedure useful. For now we simply assume this usefulness and proceed 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:

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

and then

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

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 (assumed to be) 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. This protocol has been analyzed using the BAN logic [5, p.33].

4 Generalization to Fail-Safe Protocols

Many protocols, such as the one just discussed, contain messages composed entirely of cleartext. Without changing this feature it is impossible to make such a protocol fail-stop. Other protocols may even have essential features that are incompatible with being fail-stop. Fortunately, many protocols that are not fail-stop can apparently be analyzed in the same way as fail-stop protocols. While these protocols may contain messages that will preclude their being fail-stop, it is possible to design them so that the response to any such message is always safe.

We explain what it means for a message to be safe via an example. Suppose a principal in a protocol receives a message in which the identifiers of the sender, intended recipient, protocol run, freshness, or protocol message number are not clearly given. The principal may respond to the sender with a test message

that consists of his name, intended recipient's name, a timestamp, protocol run number, an indicator that this is a test response message, a protocol message sequence number of the message to which it is a response, and possibly a nonce. This is all encrypted with a shared key or signed with his public key and encrypted with that of the intended recipient (the principal to whom he is responding). The test message could include (in place of the timestamp) a nonce he received previously from the intended recipient. Assuming that the intended recipient of this test message did send the preceeding message, he can now resend the previous message using the correct identifiers (e.g., the correct nonce), and the protocol can now continue just the same as a fail-stop protocol.

This test response message is safe in two ways. First, it is safe to send in that, even if it were prompted by an active attacker, it yields nothing that the attacker can use (that was not assumed to be available to him already). Second, it is safe to receive and respond to in that it allows the protocol to proceed in a fail-stop manner – replaying an old test response message can at most cause a resend of the original message (perhaps with different and useless data filled in). Other active attacks would be immediately detectable and the protocol effectively halted. We will call messages that are safe to send (in the sense just explained) simply "safe". We will call those that are safe to receive and respond to "fail-stop" since a protocol composed entirely from such messages is fail-stop (actually extensible fail-stop).

Other types of safe messages include messages consisting entirely of cleartext that was already public or not meant to be kept secret (such as nonces that are not used as secrets). Also acceptable are signed messages containing the cleartext just mentioned provided that the signer knows what he is signing or clearly labels those fields he does not know, e.g., pieces of previously received messages. Public key encryption of any message that one is willing to share with the corresponding principal is safe. As with signatures, anything whose significance is not known should be labelled, and it must be assumed that the correct key for the correct principal is used.

Therefore, the proof methodology given above still applies. We will call such protocols *fail-safe*.

Definition 3 (Fail-Safe Protocol) *A protocol is fail-safe if the response to any message that makes the protocol not fail-stop is safe.*

As was mentioned above, the Needham-Schroeder public key protocol is not fail-stop. We will now show

that it can be rendered fail-safe with minor modifications. The key certificates sent by the server contain nothing to indicate that they have been sent recently or as part of the protocol run. Thus an attacker is not prevented from substituting outdated certificates, which possibly containing outdated keys. When this was first observed [5], it was suggested that the certificates in messages 2 and 5 include timestamps. But, if timestamps are available, an online server is unnecessary in most instances: principals can simply hold their own certificates, and the timestamp can be used to determine if the certificate is still valid. There is another option that works even if adequately synchronized clocks are not available and that will produce a fail-safe protocol. Instead of using timestamps, nonces can be included in messages 1 and 4 and then returned in the certificates of messages 2 and 5. We will now examine the Needham-Schroeder public key protocol so modified and show that it is fail-safe.

The server can be sure of nothing based on the first message since it is composed entirely of cleartext. Therefore, we need to verify that message 2 is safe. The field that contains the nonce must be labelled as a received nonce since S cannot know what he is signing when the nonce is included in the certificate.[6] If this is done, the message is safe since it contains only publicly available information and everything signed is either known to S or properly labelled.

Upon receipt of message 2, because of the nonce A can tell that it is a response to message 1, and because of the signature A can tell that it is from S. An active attack will alter this message in a detectable way. Thus, message 2 is also fail-stop, and A can safely proceed to send message 3. B cannot be sure who sent message 3, but this is all right since message 4 is safe in the same way as message 1. Message 5 is both safe and fail-stop in the same way as message 2. Message 6 is safe provided that n_a is labelled since it is an unfamiliar number received in a message that was not fail-stop. Message 6 is also fail-stop since any attack on it will be detectable by A. We need not worry about message 7, since it is causally last and we are not worried about composability.

We have shown that the Needham-Schroeder public key protocol can be made fail-safe with a few minor modifications. This means that active attacks do not yield messages usable to the attacker, and we were thus justified in applying our methodology, i.e., in doing our secrecy assumption analysis in Section 3.3.

(Actually that analysis should be conducted on the modified protocol; however, it is immediate that this analysis is essentially the same.)

5 Formalization Using the Notion of Causal Consistency

We can give a criterion for protocols that is equivalent to being extensible fail-stop and that is formally expressible in a BAN-like logic. This means that we can rigorously, indeed formally, specify what it means for a given protocol to be fail-stop. Formal specification can be a valuable ingredient in protocol analysis, sometimes simply because of the way it forces us to represent protocol properties.

This also means that, if we so desire, we can express all phases of our proof methodology in a single formal logic, i.e., SVO [34] with temporal additions [33].[7] We cannot, however, prove that this criterion holds within the logic, and must instead turn to an examination of the accompanying semantics. A proof within the semantics may be no easier than verifying that a protocol is fail-stop. Thus, it remains to be seen if formalization within SVO is of more than theoretical interest with respect to formal verification.

The criterion is called, as was first described [33], the *causal consistency criterion* (CCC) because in protocols that satisfy it principals have matching histories of the protocol (up to the last received message). CCC is expressed in terms of a *faithfulness assumption* (FA) and a *causality requirement* (CR). The faithfulness assumption basically says that principals can be assumed to follow the protocol, i.e., they will not send a message unless they have sent or received all causally prior messages involving them. (This corresponds to clause 3 of Claim 2 above.) CR says that any message received by a principal as part of a protocol was previously sent by the appropriate principal. (In Claim 2 this is what is achieved by clauses 1 and 2.) CCC says that If FA holds, then CR does too. How to produce a formal expression of CCC from a protocol description was discussed elsewhere [33]. Here, we will persist in the informal style of presentation that we have used throughout the paper.

Claim 4 *A protocol is extensible fail-stop if and only if it satisfies CCC.*

---

[6]The label might not be necessary if we can assume that nonces are adequately typed or that public key certificates have a structure that is always identifiable in the system.

[7]These temporal additions [33] were made to AT [2]. They are equally compatible with SVO, which has the added feature of distinguishing the available messages from the received messages (following GNY). This makes for a more natural expression of the criterion, to be set out presently.

A protocol is fail-stop if and only if an active attack on a protocol message causes any causally after messages not to be sent. And, this is true if and only if any active attack on a message is detectable by the message recipient. This is so if and only if the protocol is designed so that, assuming all legitimate principals follow the protocol faithfully, any message received is exactly as sent and is sent by the appropriate sender at the appropriate time – except possibly causally last messages. In extensible fail-stop protocols this exception is removed, which is to say that CCC is satisfied.

Note that Claim 4 applies only to extensible fail-stop protocols. This is because of the last messages, which may have any structure in an ordinary fail-stop protocol. If we weaken CCC so that it no longer applies to causally last messages, then we can give a corresponding claim for fail-stop protocols in general.

6   Summary and Future Work

We have presented a methodology to facilitate the design and analysis of secure cryptographic protocols. It 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. Similarly, we advocate the general approach of restricting a protocol design to well-defined practices so that its conformation to certain guidelines ensures that certain security properties are (automatically) satisfied and its proof of security is likely to be easier and simpler.

In particular, inspired by the work on fail-stop processors [27, 28], we have defined the novel notion of a fail-stop protocol, which automatically halts in response to any active attack that interferes with protocol execution. By using this new notion, we can reduce protocol security analysis to that of passive attacks (i.e., eavesdropping) only. Furthermore, by validating that an eavesdropper cannot obtain secret keys, we can remove the crucial but paradoxical secrecy assumption that is necessary for applying BAN-like logics. Such validations can be easily performed using the possession rules in the GNY logic, and they add significant credibility to the (positive) outcome of an analysis using BAN-like logics. Our proposed methodology must validate the secrecy assumption and also apply BAN-like logics because even for a fail-stop protocol, the residue from its execution may be useful to an attacker (e.g., as in the Nessett protocol).

We have extended the basic notion to that of fail-safe protocols, and to extensible fail-stop protocols for protocol composition. Our investigation shows that many existing protocols are fail-stop or fail-safe in spirit so that our new notions are not too limiting.

Our emphasis has been on protocol designs that are easily determined to be fail-stop or fail-safe. However, some protocols, such as those to protect poorly-chosen passwords from guessing attacks [13], may have other requirements that conflict with some of the particular fail-stop requirements, e.g., those described in Claim 2. To check whether such protocols are fail-stop, one possibility is to apply tools like the Interrogator or the NRL Protocol Analyzer [19] that can search backwards for the prerequisites of an action (e.g., sending a message); thus one direction for research is to investigate the integration of such methods within the methodology set out herein.

Another research direction is to extend the classes of fail-stop and fail-safe protocols. In particular, it will be fruitful to look at other generalizations of the fail-stop concept, and at new concepts within our general approach. Finally, it will be very interesting to see if it is possible, and how, to define a hierarchy of attack models so that a protocol in one model can be easily converted into another. This is analogous to converting protocols between various fault models [23].

Acknowledgements

References

[1] 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, California, May 1994.

[2] 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.

[3] R.J. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11):32–40, November, 1994.

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

[9] 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.

[10] L. Gong. Thoughts on Cryptographic Protocols. Talk at the *Cambridge Workshop on the Design, Verification, and Implementation of Security Protocols*, Cambridge, England, April 1993.

[11] L. Gong. Initial Thought on Secure Protocols Interaction. Talk at the *Cambridge Workshop on the Design, Verification, and Implementation of Security Protocols*, Cambridge, England, April 1994.

[12] L. Gong. Fail-Stop Protocols: An Approach to Designing Secure Protocols. Technical Report SRI-CSL-94-14, Computer Science Laboratory, SRI International, Menlo Park, California, October 1994.

[13] 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.

[14] 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, California, May 1990.

[15] N. Heintze and J.D. Tygar. A Model for Secure Protocols and Their Compositions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 2–13, Oakland, California, May 1994.

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

[17] 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.

[18] R. Kailar, V.D. Gligor, and L. Gong. On the Security Effectiveness of Cryptographic Protocols. In *Proceedings of the 4th IFIP Working Conference on Dependable Computing for Critical Applications*, volume 9 of *Dependable Computing and Fault-Tolerant Systems*, pages 139–157, San Diego, California, January 1994.

[19] R. Kemmerer, C. Meadows, and J. Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2):79–130, Spring 1994.

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

[21] M. Merritt and P. Wolper. States of Knowledge in Cryptographic Protocols. Unpublished manuscript, 1985. An earlier version 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.

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

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

[26] 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.

[27] 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.

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

[29] E. Snekkenes. Exploring the BAN Approach to Protocol Analysis. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 171–181, Oakland, California, May 1991.

[30] S.G. Stubblebine and V.D. Gligor. On Message Integrity in Cryptographic Protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 85–104, California, May 1992.

[31] S.G. Stubblebine and V.D. Gligor. Protecting the Integrity of Privacy-enhanced Electronic Mail with DES-based Authentication Codes. In *Proceedings of the PSRG Workshop on Network and Distributed Systems Security*, San Diego, California, February 1993.

[32] P. Syverson. Knowledge, Belief and Semantics in the Analysis of Cryptographic Protocols. *Journal of Computer Security*, 1(4):317–334, 1992.

[33] P. Syverson. Adding Time to a Logic of Authentication. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 97–101, Fairfax, Virginia, November 1993.

[34] P. Syverson and P.C. van Oorschot. On Unifying Some Cryptographic Protocol Logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 14–28, Oakland, California, May 1994.