Protocol Design for Integrity Protection

Stuart G. Stubblebine

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
stubblebine@isi.edu

Virgil D. Gligor

Electrical Engineering Department
University of Maryland
College Park, Maryland 20742
gligor@eng.umd.edu

Abstract

We present a design method for message integrity protection. We illustrate the use of the method by designing large classes of message types whose integrity is provably preserved and by applying the method to the symmetric key option of the Privacy-enhanced Electronic Mail protocol to help discover and eliminate an integrity vulnerability.

1. INTRODUCTION

The integrity of messages in cryptographic protocols has been one of the long-standing assumptions made by most protocol analysis and design methods. Recently, Stubblebine and Gligor [SG92, 93a] showed that a significant number of cryptographic protocols used in practice contain subtle flaws that can expose communications to integrity attacks by active intruders. To date, formal characterizations of message integrity for cryptographic protocols have not been available and, consequently, protocol design for integrity protection has been performed mostly on an ad-hoc basis.

We present a design method for integrity protection in cryptographic protocols that is based on a formal model [SG93b]. The method is independent of the specific encryption system (e.g., symmetric or asymmetric cryptosystem) and checksum/digest functions used. It expresses desirable requirements for message integrity protection in terms of abstract encryption and checksum/digest properties, and relates these properties to the message type, representation, and lifetime of the protocol run and keys used. The use of the method is illustrated by the design of a large class of message types whose integrity is provably preserved in face of active intruder attacks. In particular, the method is used to help discover and eliminate a vulnerability in the symmetric-key option of the Privacy-enhanced Electronic Mail (PEM) protocol for the Internet.

The paper is organized as follows. In section 2, we introduce the terminology used in the description of the design method, and present a message integrity condition. In Section 3, we present the design method, and in Section 4 we include an example of a significant class of message types that preserve message integrity in face of active intruder attacks. In Section 5, we demonstrate the application of the method to the correction of a PEM option vulnerability (i.e., to determine what properties of encryption and checksum/digest can be used to satisfy the integrity protection objectives).

2. DEFINITIONS AND MESSAGE INTEGRITY CONDITION

The analysis of message integrity in cryptographic protocols requires the specification of the behavior of both the communicating principals and the attacker and the message structure and representation [SG92, 93b]. These principals, and the attacker, communicate by constructing and sending, and by receiving and recognizing, messages (i.e., protocol data units). For each message type, a principal uses a Key to encrypt or decrypt selected components of the message and, possibly, to compute the redundancy (e.g., checksum, digest) function added to the message. The redundancy functions are generally intended to capture the membership (M), order (O), and cardinality (C) of the blocks within a message. For this reason, these functions are generically called the *MOC* functions [SG92]. If the result of the MOC computation over the (possibly decrypted) blocks of a message equals that MOC value included in that message by the sender, the recipient believes that the received message is genuine provided that only an entity initially possessing a secret Key could have created a message representation of that type.

Message Structure: The structure of a message consists of a Key, the plaintext and ciphertext blocks, which are created by using the encryption/decryption functions *ENC()/DEC()*. As shown in Figure 1, the Key attribute of a message structure may include multiple component attributes, such as a MOC component key, called the *moc-key*; an encryption/decryption component called the transformation key or, simply, *t-key*; and, depending upon the type of function, initialization vectors for the MOC computation and for the encryption itself, called the *moc-IV* and *t-IV*, respectively. Some components of the Key may be secret; e.g., the t- and moc-keys. Other components of the Key can be public as is the case with asymmetric cryptographic functions implementing encryption or digital signatures. (In the rest of this paper, whenever it is unambiguous which key and IV is used, or whenever the two keys and IVs are identical, the term key and IV are used without any qualification.)

In Proc. 1993 IEEE Symposium on Research in Security and Privacy,
IEEE Computer Society Press, Oakland, California, May 1993.

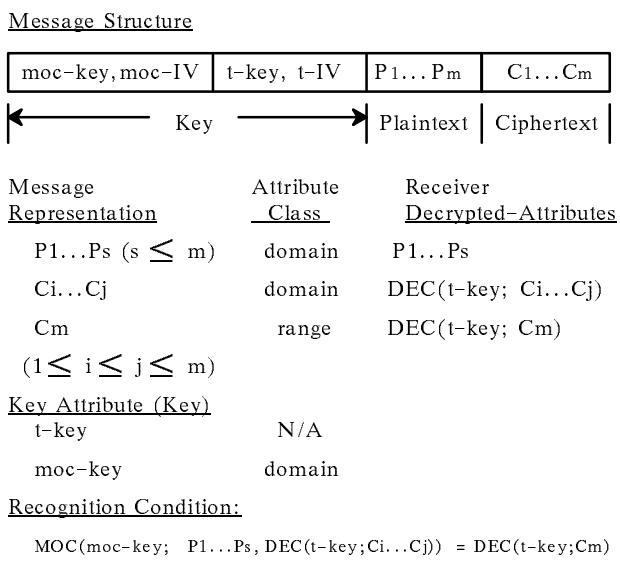

Figure 1. Message Structure, Representation, Decrypted-Attributes, and Recognition Condition

Message Representation: The representation of a message consists of a specified subset of the attributes of a message structure; i.e., those attributes that are sent from the sender to the receiver. Not all attributes of the message structure are included in the representation. For example, the secret components of the Key are never included as cleartext in the representation of any message. The representation of a message is defined by a message type. Multiple types of private (confidentiality and integrity protected) and safe (integrity protected) messages can be included in a protocol.

The message sender views the message structure as shown in Figure 1. The message receiver views the message structure as the union of the message representation, Key attribute, and decrypted attributes. The receiver determines values for decrypted attributes from the physical blocks of a message representation and from the Key (see Figure 1). For example, block DEC(t-key; Cm) represents a decrypted attribute that corresponds to the moc value.

The MOC() function determines the MOC value whose size is a constant, k. The MOC() function divides the message representation into the domain representation and range representation components. The *range representation* is the set of blocks of the message representation whose decrypted attributes correspond to the moc value. The *domain representation* is the set of blocks of the message representation whose decrypted attributes are in the domain of the MOC() function. (An attribute is in the domain of the MOC() function if computing the MOC value to recognize a message requires the attribute to be specified.) Values derived from the (possibly decrypted) blocks of the domain representation are used to decode and check the MOC value. We say that a range (domain) representation *matches* a domain (range) representation whenever the MOC value included in the range representation equals the MOC value computed over the values derived from the domain representation. The domain and range representations are defined with respect to physical blocks of the message representation that are seen ''on the wire" after the message representation has been constructed using the ENC()/DEC() and MOC() functions.

Message Integrity Condition: At any time, the protocol state contains the messages that have been constructed and sent, messages that have been received and recognized, and message types each principal can recognize. A protocol run is a finite sequence of states.The message integrity condition is defined as:

The probability that a bogus message representation is recognized by a principal in a state of a protocol run is no greater than a threshold specified by the protocol integrity policy.

The threshold specified by the protocol integrity policy can be different for each message type since messages sent by a principal can have different levels of integrity protection (e.g., weak and strong integrity in DCE [OSF91]). Attackers are assumed to have physical access to the communication media. Thus, the message integrity condition can be rewritten to capture the notion that an attacker can create a recognizable message representation. That is(*):

The probability that an attacker can create a bogus message representation of type T recognizable to a principal in a state of a protocol run, is no greater than a threshold specified by the protocol integrity policy for message type T.

Since every block of the message representation is part of either the range or domain representation, to satisfy the message integrity condition it is sufficient to limit the probability that the attacker can (1) choose a domain representation to match a range representation, or (2) choose a range representation to match a domain representation. That is, for every state of a protocol run, given a message type, T, defined using a Key, and policy threshold specified for T,

if *for every range representation, Prob[attacker chooses a domain representation such that the domain representation matches the range representation of type T]* ≤ *Threshold,*

(*) Communication errors are modeled by attacker actions. Thus, messages sent by the attacker can be considered to be error free.

and *for every domain representation, Prob[attacker chooses a range representation such that the domain representation matches the range representation of type T]* $\leq$ *Threshold*

then *Prob[attacker can create a bogus message representation of a type, T, recognizable to a principal]* $\leq$ *Threshold*.

This above statement of the message integrity condition is used throughout the balance of this paper. Since we refer repeatedly to the clauses of this condition, we denote these clauses by S1 and S2, where the condition is ''if S1 and S2 then the Message Integrity Condition is satisfied." The sufficiency of the above condition for message integrity is demonstrated in [SG93b].

To satisfy the message integrity condition in practice, properties of the ENC()/DEC() and MOC() functions and time constraints on the protocol run and keys a) must limit the probability that an attacker discovers the keys during the run (i.e., moc–key or t–key), and b) must limit the probability that the attacker can choose, without discovering the keys, a domain representation to match a range representation or choose a range representation to match a domain representation. (We assume that attackers may only discover keys by analyzing message representations, that principals can be trusted both to maintain the privacy of the key, and that the key is used only to send message representations of designated types.)

An attacker is prevented from discovering the keys if the designer chooses and applies MOC() and ENC()/DEC() functions in a manner that makes it difficult to analytically derive these keys. Furthermore, the designer can choose the MOC() and ENC()/DEC() functions, and apply them to the construction of a message representation, in a manner that makes it difficult for the attacker to analytically discover matching range or domain representations without knowing the key. Note that the MOC() function need not always be computationally complex itself, since the combined use of the MOC() and ENC()/DEC() functions may exhibit the desired complexity properties.

Since analytic attacks can reduce the attacker's uncertainty in discovering the value of a key, the designer can limit the attacker's chances of verifying a key by limiting the lifetime of the protocol run. The bound on the protocol run by the key lifetime is denoted by δ_k. Though an analytic attack can reduce the uncertainty of satisfying the recognition condition (viz., Figure 1), an attack can be further constrained (1) by limiting the time period, δ_d, during which the attacker can obtain a solution enabling him to choose a domain representation that matches a range representation, or (2) by limiting the time period, δ_r, during which the attacker can obtain a solution enabling him to choose a range representation that matches a domain representation.

Scope of the Method: The design method concerns the integrity protection of protocols as opposed to the design of individual message structures in isolation. This is important since messages of one type, although fully protected, can affect the integrity of other types [SG93a]. The designer can use the method to determine the properties and parameters of the ENC()/DEC() and MOC() functions that satisfy the integrity policy. For example, the method can be used to determine sufficient properties of the MOC() function and the relationship between function parameters (e.g., block size, MOC range size) and constraints on the lifetime of a protocol run and secret key.

Note that use of the method does not necessarily result in an unique message design and protocol. In fact, the method demonstrates that different message integrity designs and protocols are possible having the same level of integrity protection. The method can be applied to traditional messages types which are built using a single MOC() function and (optionally) encrypted. It does not specify either ENC()/DEC() or MOC() functions; instead, it enables the selection of function properties for particular message types.

We make no attempt to analyze whether specific protocol functions satisfy the MOC() and ENC()/DEC() properties selected by using the method. This analysis is dependent on the algebraic properties of the specific protocol functions. Although important, this analysis is beyond the scope of this method. The method does not attempt to provide protection against message–stream modification. Services that provide such protection can be built into protocols once integrity of individual messages is demonstrably preserved. The method does not attempt to capture information gained by actions of the principals other than sending and receiving messages (e.g., key leakage by un–trusted principals through covert channels).

3. DESIGN METHOD

The design method is based on a state–transition model [SG93b]. The general objective of the design method is to ensure that the selected properties of the ENC()/DEC() and MOC() functions, message structures, message representations and protocol assumptions are sufficient to achieve the protocol integrity objective; i.e., to satisfy the conditions defined by clauses S1 and S2 for each message type. The design method consists of the following four steps:

(1) the designer specifies the integrity policy; i.e., for each message type supported by the protocol, a

message integrity threshold, Threshold, is specified. The Threshold is an independent design parameter that states the maximum allowable probability that a bogus message can be accepted as legitimate.

(2) the designer specifies the message structure and representation for each type of message; in particular, the domain and range representations are defined, and the attribute constraints are identified. These constraints state assumptions about whether the value of a secret–domain attribute can be discovered, or limit the number of known values for attributes that may be chosen by a potential attacker. The protocol throughput constraint, or the maximum rate at which messages of each type are created and sent, is also specified.

(3) the designer specifies the properties of the ENC()/DEC() and MOC() functions for each message type. These properties are selected such that the conditions S1 and S2 are satisfied whenever the protocol throughput and run lifetime limit the probability, to the chosen Threshold, that (i) the secret components of the Key are discovered, and (ii) that the message representations generated by all the protocol message types enable the creation of bogus messages. The set of properties of the ENC()/DEC() and MOC() functions depend on the constraints placed on the message representation by the protocol (as specified in step (2)).

(4) the designer determines the lifetime constraints on the protocol run as required by each message type, and then specifies the maximum lifetime for the protocol run. The goal of this specification is to ensure that constraints of step (2) are satisfied. In other words, the lifetime is specified such that an attacker can neither guess the Key nor construct bogus messages that can satisfy the message integrity condition.

The protocol design relies on the following set of common protocol assumptions:

PA1. Principals are trusted to maintain the privacy of the secret components of the Key (i.e., the key, which could be the t–key, or moc–key).

PA2. A key used in the protocol run is generated at random from the set of keys that satisfy the computational needs of the algorithm. (The space of *key* is represented as |key|.)

PA3. The space and the computational power of the attacker are known. (Maximum pre–computation is also known.)

(1) Specify the Policy Threshold

The Threshold (i.e., maximum allowable probability that a bogus message is accepted as legitimate), specified by the integrity policy, is based on the degree of protection desired for the data attributes for the message type. The Threshold may not be smaller than 1/|key|.

(2) Specify the Message Structure, Representation, and Protocol Constraints

Message Structure and Representation: The protocol definition includes a specification of the message formats, including all the attributes of the message structure and representation. The message structure and representation for each protocol message type are ''idealizations" of the actual message formats. These idealizations are required for the uniform representation of all attributes of the message types used in a protocol.

Protocol constraints: For each message type, the protocol places constraints on the value of a message attribute (e.g., the attribute is encrypted), or limits the number of known values for attributes that can be chosen by a potential attacker. These constraints differ with the type of message because different types of ENC()/DEC() and MOC() functions are used for different types of messages and because these functions are used in different ways for different types of messages [SG92, SG93a]. Identifying the attribute constraints is necessary in message design because it provides the *prima facie* evidence that the attacker is prevented from successfully creating a recognizable bogus message. Identifying the attribute constraints placed by a protocol requires that the attributes of *all* message types be examined. This is necessary because determining whether an attribute should be constrained can depend on all uses of secret components of the Key.

The secret and decrypted attributes may be either constrained or unconstrained. A *secret attribute is constrained* whenever the probability that the attacker may discover the attribute value is less than the given Threshold (*). Otherwise, the secret component is unconstrained. Examples of constrained secret–attributes include the secret components of the Key; i.e., the secret transformation key (t–key) the and secret–domain attribute (moc–key). We call the moc–key attribute a *secret–domain attribute* because it is within the domain of the MOC() function.To ensure that a secret–attribute remains constrained, the attacker must be prevented from verifying the value of a chosen attribute. If an attacker is not (adequately) prevented from verifying a chosen attribute that is meant to remain secret, then that attribute is unconstrained.

A *decrypted attribute is constrained* if the number of known attribute values the attacker can choose is limited. Otherwise, the decrypted attribute is unconstrained.Examples of constrained decrypted attributes include any part of the ciphertext of the domain or range representation when the attacker is

(*) Note PA2 maximizes the attacker's uncertainty of the attribute's value.

prevented from discovering the maximum number of known plaintext–ciphertext pairs. It is important to note that decrypted attributes may be constrained even when the attributes were not encrypted. For example, the computation of a moc value using a keyed MOC() function may be included in the representation without an an additional layer of encryption. This moc value is constrained if the number of X,MOC(moc–key,X) pairs that can be obtained by the attacker is limited. Examples of unconstrained decrypted attributes also include higher layer attributes, such as user–data, which are included in the message representation in an unencrypted form.

(3) Specify the ENC()/DEC() and MOC() Function Properties

ENC()/DEC() and MOC() functions are used (1) to prevent the attacker from discovering a secret–attribute, and to limit the number of known values the attacker may choose for an attribute, and (2) to ensure that, given the designer's choice of attribute constraints, conditions of clauses S1 and S2 above are satisfied. We do not assume the actual ENC()/DEC() and MOC() functions are secret.

ENC()/DEC() Functions: Given that a protocol uses an ENC()/DEC() function satisfying definition EA1 below, properties EA2 and EA3, also defined below, ensure that any protocol can produce constrained, and trivially unconstrained attributes.

EA1. The encryption function of block ciphers break a text M into successive blocks P1,..,Pn, and encipher each block with the same key; i.e., ENC(key,M) = ENC(key,P1) ENC(key,P2) .... ENC(key,Pn) = C1,...,Cn. The decryption function, DEC, is the left inverse of ENC; i.e., DEC(key,ENC(key,M))=M. The block size of the cipher is b bits. (Pairs Pi, Ci are called the plaintext–ciphertext pairs. For example, in CBC encryption, the plaintext–ciphertext pairs are <ENC(key,Pi–1)$\oplus$Pi, Ci > where $\oplus$ stands for the "exclusive or (XOR)" operator, and i > 1.)

EA2. Given a random t–key, it is computationally infeasible to determine any bit of Pi from Ci, any bit of Ci from Pi, and any bit of the key from Pi and Ci.

EA3. The use of ENC()/DEC() restricts the generation of chosen plaintext–ciphertext pairs by an entity not possessing the t–key(*).

In what follows, EA is used to represent EA1, EA2, and EA3.

(*) For example, the ENC() function may XOR a random value (e.g., a confounder, or a random IV) and the first block of plaintext before generating the ciphertext. With CBC encryption, this makes both the first and subsequent plaintext blocks random. This property is important when a server uses the same key on behalf of multiple clients [SG92].

MOC() Functions: The properties of MOC() functions defined below are defined by assigning a maximum probability that, given the constraints placed on the decrypted–domain and secret–domain attributes, the conditions of clauses S1 and S2 are satisfied; i.e., an attacker may not (1) choose a value of a decrypted–domain attribute that matches a decrypted–range attribute, or (2) choose a value of a decrypted–range attribute matching a decrypted–domain attribute, with a chance of success higher than that determined by the Threshold.

Since message attributes of different message types can be subject to multiple constraints, the design method can be used to specify multiple MOC() properties, each tailored to a different message type. Our approach to tailoring MOC() function properties to message structures is important since it facilitates the optimal choice of MOC() function for the design application.

Enumerating the many possible MOC() properties is beyond the scope of this paper. However, it is useful to reduce the number of MOC() properties by using the "high–water–mark" property selection.That is, if a decrypted– or secret–domain attribute contains both constrained and unconstrained attributes, the selection of the property requiring that the attribute be constrained is sufficient to satisfy the message integrity condition.

Selecting properties of ENC()/DEC and MOC() functions consists of selecting a set of properties satisfying, at a minimum, the same attribute constraints as those specified for the decrypted message attributes. These properties are selected such that, given the designer–specified attribute constraints from step (2), the message types and representations can be defined to satisfy the message integrity condition. In Tables 1 and 2, we summarize a few of the possible MOC() and ENC()/DEC() properties for typical message structures and representations, which can be selected using designer–specified attribute constraints.

(1) Properties limiting the probability an attacker can choose a domain representation to match a range representation.

The properties of the MOC() function, defined below, can be used to limit the probability an attacker can choose a domain representation to match a range representation. In the properties that follow, a block is defined to be "unknown" to the attacker whenever the values of that block are uniformly distributed over all possible values of any block in the range of the MOC() function. Also, the size of each block in the range of the MOC() function, k, is assumed to be sufficiently large so that each probability below approximates 1/2^k.

Constraints on Domain Attributes	*Constraints on Decrypted-Range Attributes*: Unconstrained	Constrained
Both Unconstrained	N/A (vulnerable) [1]	**SA1** [2]
Secret-Domain Constrained	**MA1** (or SA1) [3]	**MA1** or **SA1** [4]
Decrypted-Domain Constrained	N/A [5]	**MA3** and **EA** (or SA1) [6]

Table 1.

Property Assumptions Protecting Against Choosing a Domain Representation to Match a Range Respresentation

Constraints on Domain Attributes	*Constraints on Decrypted-Range Attributes*: Unconstrained	Constrained
Both Unconstrained	N/A (vulnerable) [1]	**EA** [2]
Secret-Domain Constrained	**MA2** (or none | SA1) [3]	**MA2** and/or **EA** | **MA1** or **EA** | **SA1** [4]
Decrypted-Domain Constrained	N/A [5]	**EA** | **MA3** and **EA** or **EA** | **SA1** [6]

"|" reads Given

Table 2.

Property Assumptions Protecting Against Choosing a Range Representation to Match a Domain Representation.

MOC Function Properties:

MA1. Given X of length k, and (P1...Ps−1 Pt+1..Pu), that contains m>0 or more unknown blocks, Prob[choosing Ps...Pt such that MOC(P1...Ps−1 Ps...Pt Pt+1...Pu) = X] $\approx$ 1/2^k $\leq$ Threshold.

In practice, the value m of MA1 would typically be a constant corresponding to the length of a moc-key.

MA3. Given X of length k, and a set of p or fewer random but known blocks of length b, Prob[choosing P1...Ps−1 Ps...Pt Pt+1...Pu such that each block of Ps...Pt is a member of the set, and MOC(P1...Ps−1 Ps...Pt Pt+1...Pu) =