Identifying Potential Type Confusion in Authenticated Messages

Catherine Meadows
Code 5543
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract

A *type confusion attack* is one in which a principal accepts data of one type as data of another. Although it has been shown by Heather et al. that there are simple formatting conventions that will guarantee that protocols are free from simple type confusions in which fields of one type are substituted for fields of another, it is not clear how well they defend against more complex attacks, or against attacks arising from interaction with protocols that are formatted according to different conventions. In this paper we show how type confusion attacks can arise in realistic situations even when the types are explicitly defined in at least some of the messages, using examples from our recent analysis of the Group Domain of Interpretation Protocol. We then develop a formal model of types that can capture potential ambiguity of type notation, and outline a procedure for determining whether or not the types of two messages can be confused. We also discuss some open issues.

1 Introduction

Type confusion attacks arise when it is possible to confuse a message containing data of one type with a message containing data of another. The most simple type confusion attacks are ones in which fields of one type are confused with fields of another type, such as is described in [7], but it is also possible to imagine attacks in which fields of one type are confused with a concatenation of fields of another type, as is described by Snekkenes in [8], or even attacks in which pieces of fields of one type are confused with pieces of fields of other types.

Simple type confusion attacks, in which a field of one type is confused with a field of another type, are easy to prevent by including type labels (tags) for all data and authenticating labels as well as data. This has been shown by Heather et al. [4], in which it is proved that, assuming a Dolev-Yao-type model of a cryptographic protocol and intruder, it is possible to prevent such simple type confusion attacks by the use of this technique. However, it is not been shown that this technique will work for more complex type confusion attacks, in which tags may be confused with data, and terms or pieces of terms of one type may be confused with concatenations of terms of several other types.[1] More importantly, though, although a tagging technique may work within a single protocol in which the technique is followed for all authenticated messages, it does not prevent type confusion of a protocol that uses the technique with a protocol that does not use the technique, but that does use the same authentication keys. Since it is not uncommon for master keys (especially public keys) to be used with more than one protocol, it may be necessary to develop other means for determining whether or not type confusion is possible. In this paper we explore these issues further, and describe a procedure for detecting the possibility of the more complex varieties of type confusion.

The remainder of this paper is organized as follows. In order to motivate our work, in Section Two, we give a brief account of a complex type confusion flaw that was recently found during an analysis of the Group Domain of Authentication Protocol, a secure multicast protocol being developed by the Internet Engineering Task Force. In Section Three we give a formal model for the use of types in protocols that takes into account possible type ambiguity. In Section Four we describe various techniques for constructing the artifacts that will be used in our procedure. In Section Five we give a procedure for determining whether it is possible to confuse the type of two messages. In Section Six we illustrate our procedure by showing how it could be applied to a simplified version of GDOI. In Section Seven we conclude the pa-

[1] We believe that it could, however, if the type tags were augmented with tags giving the length of the tagged field, as is done in many implementations of cryptographic protocols.

per and give suggestions for further research.

2 The GDOI Attack

In this section we describe a type flaw attack that was found on an early version of the GDOI protocol.

The Group Domain of Interpretation protocol (GDOI) [2], is a group key distribution protocol that is undergoing the IETF standardization process. It is built on top of the ISAKMP [6] and IKE [3] protocols for key management, which imposes some constraints on the way in which it is formatted. GDOI consists of two parts. In the first part, called the Groupkey Pull Protocol, a principal joins the group and gets a group key-encryption-key from the Group Controller/Key Distributor (GCKS) in a handshake protocol protected by a pairwise key that was originally exchanged using IKE. In the second part, called the Groupkey Push Message, the GCKS sends out new traffic encryption keys protected by the GCKS's digital signature and the key encryption key.

Both pieces of the protocol can make use of digital signatures. The Groupkey Pull Protocol offers the option of including a Proof-of-Possession field, in which either or both parties can prove possession of a public key by signing the concatenation of a nonce NA generated by the group member and a nonce NB generated by the GCKS. This can be used to show linkage with a certificate containing the public key, and hence the possession of any identity or privileges stored in that certificate.

As for the Groupkey Push Message, it is first signed by the GCKS's private key, and then encrypted with the key encryption key. The signed information includes a header HDR, (which is sent in the clear), and contains, besides the header, the following information:

1. a sequence number SEQ (to guard against replay attacks);
2. a security association SA;
3. a Key Download payload KD, and;
4. an optional certificate, CERT.

According to the conventions of ISAKMP, HDR must begin with a random or pseudo-random number. In pairwise protocols, this is jointly generated by both parties, but in GDOI, since the message must go from one to many, this is not practical. Thus, the number is generated by the GCKS. Similarly, it is likely that the Key Download message will end in a random number: a key. Thus, it is reasonable to assume that the signed part of a Groupkey Push Message both begins and ends in a random number.

We found two type confusion attacks. In both, we assume that the same private key is used by the GCKS to sign POPs and Groupkey Push Messages. In the first of these, we assume a dishonest group member who wants to pass off a signed POP from the GCKS as a Groupkey Push Message. To do this, we assume that she creates a fake plaintext Groupkey Push Message GPM, which is missing only the last (random) part of the Key Download Payload. She then initiates an instance of the Groupkey Pull Protocol with the GCKS, but in place of her nonce, she sends GPM. The GCKS responds by appending its nonce NB and signing it, to create a signed (GPM,NB). If NB is of the right size, this will look like a signed Groupkey Push Message. The group member can then encrypt it with the key encryption key (which she will know, being a group member) and send it out to the entire group.

The second attack requires a few more assumptions. We assume that there is a group member A who can also act as a GCKS, and that the pairwise key between A and another GCKS, B, is stolen, but that B's private key is still secure. Suppose that A, acting as a group member, initiates a Groupkey Pull Protocol with B. Since their pairwise key is stolen, it is possible for an intruder I to insert a fake nonce for B's nonce NB. The nonce he inserts is a fake Groupkey Push Message GPM' that it is complete except for a prefix of the header consisting of all or part of the random number beginning the header. A then signs (NA,GPM'), which, if NA is of the right length, will look like the signed part of a Groupkey Push Message. The intruder can then find out the key encryption key from the completed Groupkey Pull Protocol and use it to encrypt the resulting (NA,GPM') to create a convincing fake Groupkey Push Message.

Fortunately, the fix was simple. Although GDOI was constrained by the formatting required by ISAKMP, this was not the case for the information that was signed within GDOI. Thus, the protocol was modified so that, whenever a message was signed within GDOI, information was prepended saying what the purpose was (e.g. a member's POP, or a Groupkey Push Message). This eliminated the type confusion attacks.

There are several things to note here. The first is that existing protocol analysis tools are not very good at finding these types of attacks. Most assume that some sort of strong typing is already implemented. Even when this is not the case, the ability to handle the various combinations that arise is somewhat limited. For example, we found the second, less feasible, attack automatically with the NRL Protocol Analyzer, but the tool

could not have found the first attack, since the ability to model it requires the ability to model the associativity of concatenation, which the NRL Protocol Analyzer lacks. Moreover, type confusion attacks do not require a perfect matching between fields of different types. For example, in order for the second attack to succeed, it is not necessary for NA to be the same size as the random number beginning the header, only that it be no longer than that number. Again, this is something that is not within the capacity of most crypto protocol analysis tools. Finally, most crypto protocol analysis tools are not equipped for probabilistic analysis, so they would not be able to find cases in which, although type confusion would not be possible every time, it would occur with a high enough probability to be a concern.

The other thing to note is that, as we said before, even though it is possible to construct techniques that can be used to guarantee that protocols will not interact insecurely with other protocols that are formatted using the same technique, it does not mean that they will not interact insecurely with protocols that were formatted using different techniques, especially if, in the case of GDOI's use of ISAKMP, the protocol wound up being used differently than it was originally intended (for one-to-many instead of pairwise communication). Indeed, this is the result one would expect given previous results on protocol interaction [5, 1]. Since it is to be expected that different protocols will often use the same keys, it seems prudent to investigate to what extent an authenticated message from one protocol could be confused with an authenticated message from another, and to what extent this could be exploited by a hostile intruder. The rest of this paper will be devoted to the discussion of a procedure for doing so.

3 The Model

In this section we will describe the model that underlies our procedure. It is motivated by the fact that different principals may have different capacities for checking types of messages and fields in messages. Some information, like the length of the field, may be checkable by anybody. Other information, like whether or not a field is a random number generated by a principal, or a secret key belonging to a principal, will only be checkable by the principal who generated the random number in the first case, and by the possessor(s) of the secret key in the second place. In order to do this, we need to develop a theory of types that take differing capacities for checking types into account.

We assume an environment consisting of principals who possess information and can check properties of data based on that information. Some information is public and is shared by all principals. Other information may belong to only one or a few principals.

Definition 3.1 *A* field *is a sequence of bits. We let ι denote the empty field. If x and y are two fields, we let $x||y$ denote the concatenation of x and y. If $\bar{x}$ and $\bar{y}$ are two lists of fields, then we let $append(\bar{x}, \bar{y})$ denote the list obtained by appending $\bar{y}$ to $\bar{x}$.*

Definition 3.2 *A* type *is a set of fields, which may or may not have a probability distribution attached. If P is a principal, then a* type local to P *is a type such that membership in that type is checkable by P. A* public type *is one whose membership is checkable by all principals. If G is a group of principals, then a* type private to G *is a type such that membership in that type is checkable by the members of G and only the members of G.*

Examples of a public type would be all strings of length 256, the string "key," or well-formed IP addresses. Examples of private types would be a random nonce generated by a principal (private to that principal) a principal's private signature key (private to that principal), and a secret key shared by Alice and Bob (private to Alice and Bob, and perhaps the server that generated the key, if one exists). Note that a private type is not necessarily secret; all that is required is that only members of the group to whom the type is private have a guaranteed means of checking whether or not a field belongs to that type. As in the case of the random number generated by a principal, other principals may have been told that a field belongs to the type, but they do not have a reliable means of verifying this.

The decision as to whether or not a type is private or public may also depend upon the protocol in which it is used and the properties that are being proved about the protocol. For example, to verify the security of a protocol that uses public keys to distribute master keys, we may want to assume that a principal's public key is a public type, while if the purpose of the protocol is to validate a principal's public key, we may want to assume that the type is private to that principal and some certification authority. If the purpose of the protocol is to distribute the public key to the principal, we may want to assume that the type is private to the certification authority alone.

Our use of public and local types is motivated as follows. Suppose that an intruder wants to fool Bob into accepting an authenticated message M from a principal Alice as an authenticated message N from Alice. Since M is generated by Alice, it will consist of types local to

her. Thus, for example, if M is supposed to contain a field generated by Alice it will be a field generated by her, but if it is supposed to contain a field generated by another party, Alice may only be able to check the publically available information such as the formatting of that field before deciding to include it in the message. Likewise, if Bob is verifying a message purporting to be N, he will only be able to check for the types local to himself. Thus, our goal is to be able to check whether or not a message built from types local to Alice can be confused with another message built from types local to Bob, and from there, to determine if an intruder is able to take advantage of this to fool Bob into producing a message that can masquerade as one from Alice.

We do not attempt to give a complete model of an intruder in this paper, but we do need to have at at least some idea of what types mean from the point of view of the intruder to help us in computing the probability of an intruder's producing type confusion attacks. In particular, we want to determine the probability that the intruder can produce (or force the protocol to produce) a field of one type that also belongs to another type. Essentially, there are two questions of interest to an intruder: given a type, can it control what field of that type is sent in a message, and given a type, will any arbitrary member of that type be accepted by a principal, or will a member be accepted only with a certain probability.

Definition 3.3 *We say that a type is* under the control of the intruder *if there is no probability distribution associated with it. We say that a type is* probabilistic *if there a a probability distribution associated with it. We say that a probabilistic type local to a principal A is* under the control of A *if the probability of A accepting a field as a member of X is given by the probability distribution associated with X.*

The idea behind probabilistic types and types under control of the intruder is that the intruder can choose what member of a type can be used in a message if it is under its control, but for probabilistic types the field used will be chosen according to the probability distribution associated with the type. On the other hand, if a type is not under the control of a principal A, then A will accept any member of that type, while if the type is under the control of A, she will only accept an element as being a member of that type according to the probability associated with that type.

An example of a type under the control of an intruder would be a nonce generated by the intruder, perhaps while impersonating someone else. An example of a probabilistic type that is not under the control of A would be a nonce generated by another principal B and sent to A in a message. An example of a probabilistic type that is also under the control of A would be a nonce generated by A and sent by A in a message, or received by A in some later message.

Definition 3.4 *Let X and Y be two types. We say that $X \sqcap Y$ holds if an intruder can force a protocol to produce an element x of X that is also an element of Y.*

Of course, we are actually interested in the probability that $X \sqcap Y$ holds. Although the means for calculating $P(X \sqcap Y)$ may vary, we note that the following holds if there are no other constraints on X and Y:

1. If X and Y are both under the control of the intruder, then $P(X \sqcap Y)$ is 1 if $X \cap Y \neq \phi$ and is zero otherwise;
2. If X is under the control of the intruder, and Y is a type under the control of A, and the intruder knows the value of the member of Y before choosing the member of X, then $P(Y \sqcap X) = P(\hat{x} \in X \cap Y)$, where $\hat{x}$ is the random variable associated with X;
3. If X a type under the control of A, and Y is a type local to B but not under the control of B, then $P(X \sqcap Y) = P(\hat{x} \in X \cap Y)$;
4. If X is under the control of A and Y is under the control of some other (non-intruder) B, then $P(Y \sqcap X) = P(\hat{x} = \hat{y})$ where $\hat{x}$ is the random variable associated with X, and $\hat{y}$ is the random variable associated with Y.

Now that we have a notion of type for fields, we extend it to a notion of type for messages.

Definition 3.5 *A* message *is a concatenation of one or more fields.*

Definition 3.6 *A* message type *is a function $\mathcal{R}$ from lists of fields to types, such that:*

1. *The empty list is in $\mathrm{Dom}(\mathcal{R})$;*
2. *$\langle x_1, ..., x_k \rangle \in \mathrm{Dom}(\mathcal{R})$ if and only if $\langle x_1, ..., x_{k-1} \rangle \in \mathrm{Dom}(\mathcal{R})$ and $x_k \in \mathcal{R}(\langle x_1, ..., x_{k-1} \rangle)$;*
3. *If $\langle x_1, ..., x_k \rangle \in \mathrm{Dom}(\mathcal{R})$, and $x_k = \iota$, then $\mathcal{R}(\langle x_1, ..., x_k \rangle) = \{\iota\}$, and ;*
4. *For any infinite sequence $S = \langle ..., x_i, ... \rangle$ such that all prefixes of S are in $Dom(\mathcal{R})$, there exists an n such that, for all $i > n$, $x_i = \iota$.*

The second part of the definition shows how, once the first $k-1$ fields of a message are known, then $\mathcal{R}$ can be used to predict the type of the k'th field. The third and fourth parts describe the use of the empty list ι in indicating message termination. The third part says that, if the message terminates, then it can't start up again. The fourth part says that all messages must be finite. Note, however, that it does not require that messages be of bounded length. Thus, for example, it would be possible to specify, say, a message type that consists of an unbounded list of keys.

The idea behind this definition is that the type of the n'th field of a message may depend on information that has gone before, but exactly where this information goes may depend upon the exact encoding system used. For example, in the tagging system in [4], the type is given by a tag that precedes the field. In many implementations, the tag will consist of two terms, one giving the general type (e.g. "nonce"), and the other giving the length of the field. Other implementations may use this same two-part tag, but it may not appear right before the field; for example in ISAKMP, and hence in GDOI, the tag refers, not to the field immediately following it, but the field immediately after that. However, no matter how tagging is implemented, we believe that it is safe to assume that any information about the type of a field will come somewhere before the field, since otherwise it might require knowledge about the field that only the tag can supply (such as where the field ends) in order to find the tag.

Definition 3.7 *The* support *of a message type $\mathcal{R}$ is the set of all messages of the form $x_1||...||x_n$ such that $\langle x_1, ..., x_n\rangle \in \mathrm{Dom}(\mathcal{R})$.*

For an example of a message type, we consider a message of the form

$\text{“}nonce'', N_1, NONCE_1, \text{“}nonce'', N_2, NONCE_2$

where $NONCE_1$ is a random number of length N_1 generated by the creator of the message, N_1 is a 16-bit integer, and $NONCE_2$ is a random number of length N_2, where both $NONCE_2$ and N_2 are generated by the intended receiver, and N_2 is another 16-bit integer. From the point of view of the generator of the message, the message type is as follows:

1. $\mathcal{R}(\langle\rangle) = \text{“}nonce''$.
2. $\mathcal{R}(\langle\text{“}nonce''\rangle) = \{X|length(X) = 16\}$. Since N_1 is generated by the sender, it is a type under the control of the sender consisting of the set of 16-bit integers, with a certain probability attached.
3. $\mathcal{R}(\langle\text{“}nonce'', N_1\rangle) = \{X|length(X) = N_1\}$. Again, this is a private type consisting of the set of fields of length N_1. In this case, we can choose the probability distribution to be the uniform one.
4. $\mathcal{R}(\langle\text{“}nonce'', N_1, NONCE_1\rangle) = \{\text{“}nonce''\}$.
5. $\mathcal{R}(\langle\text{“}nonce'', N_1, NONCE_1, \text{“}nonce''\rangle) = \{X|length(X) = 16\}$. Since the sender did not actually generate N_2, all he can do is check that it is of the proper length, 16. Thus, this type is not under the control of the sender. If N_2 was not authenticated, then it is under the control of the intruder.
6. $\mathcal{R}(\langle\text{“}nonce'', N_1, NONCE_1, \text{“}nonce'', N_2\rangle) = \{Y|length(Y) = N_2\}$. Again, this value is not under the control of the sender, all the principal can do is check that what purports to be a nonce is indeed of the appropriate length.
7. $\mathcal{R}(\langle\text{“}nonce'', N_1, NONCE_1, \text{“}nonce'', N_2, NONCE_1, \rangle) = \{\iota\}$. This last tells us that the message ends here.

From the point of view of the receiver of the message, the message type will be somewhat different. The last two fields, N_2 and $NONCE_2$ will be types under the control of the receiver, while N_1 and $NONCE_1$ will be types not under its control, and perhaps under the control of the intruder, whose only checkable property is their length. This motivates the following definition:

Definition 3.8 *A* message type local to a principal P *is a message type $\mathcal{R}$ whose range is made up of types local to P.*

We are now in a position to define type confusion.

Definition 3.9 *Let $\mathcal{R}$ and $\mathcal{S}$ be two message types. We say that a pair of sequences $\langle x_1, ..., x_n\rangle \in Dom(\mathcal{R})$ and $\langle y_1, ..., y_n\rangle \in Dom(\mathcal{S})$ is a* type confusion between $\mathcal{R}$ and $\mathcal{S}$ *if:*

1. $\iota \in \mathcal{R}(\langle x_1, ..., x_n\rangle)$*;*
2. $\iota \in \mathcal{S}(\langle y_1, ..., y_m\rangle)$*, and;*
3. $x_1||...||x_n = y_1||...||y_m$.

The first two conditions say that the sequences describe complete messages. That last conditions says that the messages, considered as bit-strings, are identical.

Definition 3.10 *Let $\mathcal{R}$ and $\mathcal{S}$ be two message types. We say that $\mathcal{R} \sqcap \mathcal{S}$ holds if an intruder is able to force a protocol to produce an $\bar{x}$ in $Dom(\mathcal{R})$ such that there exists $\bar{y}$ in $Dom(\mathcal{S})$ such that $(\bar{x}, \bar{y})$ is a type confusion..*

Again, what we are interested in is computing, or at least estimating, $P(\mathcal{R} \sqcap \mathcal{S})$. This will be done in Section 5.

4 Constructing and Rearranging Message Types

In order to perform our comparison procedure, we will need the ability to build up and tear down message types, and create new message types out of old. In this section we describe the various ways that we can do this.

We begin by defining functions that are restrictions of message types (in particular to prefixes and postfixes of tuples).

Definition 4.1 *An* n-postfix message type *is a function $\mathcal{R}$ from tuples of length n or greater to types such that:*

1. *For all $k > 0$, $\langle x_1, ..., x_{n+k} \rangle \in \mathrm{Dom}(\mathcal{R})$ if and only if $x_{n+k} \in \mathcal{R}(\langle x_1, ..., x_{n+k-1} \rangle)$;*
2. *If $\langle x_1, ..., x_{n+k} \rangle \in \mathrm{Dom}(\mathcal{R})$, and $x_{n+k} = \iota$, then $\mathcal{R}(\langle x_1, ..., x_{n+k+1} \rangle) = \{\iota\}$, and ;*
3. *For any infinite sequence $S = \langle ..., x_i, ... \rangle$ such that all prefixes of S of length n and greater are in $Dom(\mathcal{R})$, there exists an m such that, for all $i > m$, $x_i = \iota$.*

We note that the restriction of a message type $\mathcal{R}$ to sequences of length n or greater is an n-postfix message type, and that a message type is a 0-postfix message type.

Definition 4.2 *An n-prefix message type is a function $\mathcal{R}$ from tuples of length less than n to types such that:*

1. *$\mathcal{R}$ is defined over the empty list;*
2. *For all $k < n$, $\langle x_1, ..., x_k \rangle \in \mathrm{Dom}(\mathcal{R})$ if and only if $x_k \in \mathcal{R}(\langle x_1, ..., x_{k-1} \rangle)$, and;*
3. *If $k < n - 1$, and $\langle x_1, ..., x_k \rangle \in \mathrm{Dom}(\mathcal{R})$, and $x_k = \iota$, then $\mathcal{R}(\langle x_1, ..., x_{k+1} \rangle) = \{\iota\}$.*

We note that the restriction of a message type to sequences of length less than n is an n-prefix message type.

Definition 4.3 *We say that a message type or n-prefix message type $\mathcal{R}$ is* t-bounded *if $\mathcal{R}(x) = \iota$ for all tuples x of length t or greater.*

In particular, a message type that is both t-bounded and t-postfix will be a trivial message type.

Definition 4.4 *Let $\mathcal{R}$ be an n-postfix message type. Let X be a set of m-tuples in the pre-image of $\mathcal{R}$, where m $\geq$ n. Then $\mathcal{R}\lfloor X$ is defined to be the restriction of R to the set of all $\langle x_1, ..., x_m, ..., x_r \rangle$ in $Dom(\mathcal{R})$ such that $\langle x_1, ..., x_m \rangle \in X$.*

Definition 4.5 *Let $\mathcal{R}$ be an n-prefix message type. Let X be a set of n-1 tuples. Then $\mathcal{R}\lceil X$ is defined to be the restriction of $\mathcal{R}$ to the set of all tuples $\bar{x}$ such that $\bar{x} \in X$, or $\bar{x} = \langle x_1, ...x_i \rangle$ such that there exists $\langle y_{i+1}, ..., y_{n-1} \rangle$ such that $\langle x_1, ...x_i, y_{i+1}, ..., y_{n-1} \rangle \in X$.*

Definition 4.6 *Let $\mathcal{R}$ be an n-postfix message type. Then $Split(\mathcal{R})$ is the function whose domain is the set of all $\langle x_1, ..., x_n, y_1, y_2, x_{n+2}, ..., x_m \rangle$ of length n+1 or greater such that $\langle x_1, ..., x_n, y_1 || y_2, x_{n+2}, ..., x_m \rangle \in Dom(\mathcal{R})$ and such that*

a. *For the tuples of length $i > n$ +1, $Split(\mathcal{R})(\langle x_1, ..., x_n, y_1, y_2, x_{n+2}, ..., X_m \rangle) = \mathcal{R}(\langle x_1, ..., x_n, y_1 || y_2, x_{n+2}, ..., x_m \rangle)$, and;*

b. *For tuples of length n +1 , $Split(\mathcal{R})(\langle y_1, ..., y_{n+1} \rangle) = \{z \mid \langle y_1, ..., y_{n+1} || z \rangle \in Dom(\mathcal{R})$.*

Definition 4.7 *Let $\mathcal{R}$ be an n-prefix message type. Let F be a function from a set of n-tuples to types such that there is at least one tuple $\langle x_{i+1} ..., x_n \rangle$ in the domain of F such that $\langle x_{i+1} ..., x_{n-1} \rangle$ is in the domain of $\mathcal{R}$. Then $\mathcal{R} \sharp F$, the* extension of $\mathcal{R}$ by F, *is the function whose domain is*

a. *For $i < n$, the set of all $\langle x_1 ...., x_i \rangle$ such that $\langle x_1 ...., x_i \rangle \in Dom(\mathcal{R})$, and such that there exists $\langle x_{i+1} ..., x_n \rangle$ such that $\langle x_1 ...., x_i, x_{i+1} ..., x_n \rangle \in Dom(F)$;*

b *For i = n, the set of all $\langle x_1 ...., x_{n-1}, x_n \rangle$ such that $\langle x_1 ...., x_{n-1} \rangle \in Dom(\mathcal{R})$ and $\langle x_1 ...., x_{n-1}, x_n \rangle \in Dom(F)$;*

and whose restriction to tuples of length less than n is $\mathcal{R}$, and whose restriction to n-tuples is F.

Proposition 4.1 *If $\mathcal{R}$ is an n-postfix message type, then $\mathcal{R}\lfloor X$ is an m-postfix message type for any set of m-tuples X, and $Split(\mathcal{R})$ is an (n+1)-postfix message*

type. If $\mathcal{R}$ is t-bounded, then so is $\mathcal{R}\lfloor X$, while $Split(\mathcal{R})$ is (t+1)-bounded. Moreover, if S is an n-prefix message type, then so is $\mathcal{S}\lceil Y$ for any set of n-1 tuples Y, and $\mathcal{S}\sharp F$ is an (n+1)-prefix message type for any function F from n-tuples to types such at for at least one element $\langle x_{i+1}..., x_n\rangle$ in the domain of F, $\langle x_{i+1}..., x_{n-1}\rangle$ is in the domain of $\mathcal{S}$.

We close with one final definition.

Definition 4.8 *Let F be a function from k-tuples of fields to types. We define $Pre(F)$ to be the function from k-tuples of fields to types defined by $Pre(F)(x)$ is the set of all prefixes of all elements of $F(x)$.*

5 The Zipper: A Procedure for Comparing Message Types

We now can define our procedure for determining whether or not type confusion is possible between two message types $\mathcal{R}$ and $\mathcal{S}$, that is, whether it is possible for a verifier to mistake a message of type $\mathcal{R}$ generated by some principal for a message of type $\mathcal{S}$ generated by that same principal , where $\mathcal{R}$ is a message type local to the generator, and $\mathcal{S}$ is a message type local to the verifier. But, in order for this to occur, the probability of $\mathcal{R} \sqcap \mathcal{S}$ must be nontrivial. For example, consider a case in which $\mathcal{R}$ is a type local to and under the control of Alice consisting of a random variable 64 bits long, and $\mathcal{S}$ consists of another random 64-bit variable local to and under the control of Bob. It is possible that $\mathcal{R} \sqcap \mathcal{S}$ holds, but the probability that this is so is only $1/2^{64}$. On the other hand, if $\mathcal{R}$ is under the control of the intruder, then the probability that their support is non-empty is one. Thus, we need to choose a threshold probability, such that we consider a type confusion whose probability falls below the threshold to be of negligible consequence.

Once we have chosen a threshold probability, our strategy will be to construct a "zipper"between the two message types to determine their common support. We will begin by finding the first type of $\mathcal{R}$ and the first type of $\mathcal{S}$, and look for their intersection. Once we have done this, for each element in the common support, we will look for the intersection of the next two possible types of $\mathcal{R}$ and $\mathcal{S}$, respectively, and so on. Our search will be complicated, however, by the fact that the matchup may not be between types, but between pieces of types. Thus, for example, elements of the first type of $\mathcal{R}$ may be identical to the prefixes of elements of the first type of $\mathcal{S}$, while the remainders of these elements may be identical to elements of the second type of $\mathcal{R}$, and so forth. So we will need to take into account three cases: the first, where two types have a nonempty intersection, the second, where a type from $\mathcal{R}$ (or a set of remainders of types from $\mathcal{R}$) has a nonempty intersection with a set of prefixes from the second type of $\mathcal{S}$, and the third, where a type from $\mathcal{S}$ (or a set of remainders of types from $\mathcal{S}$) has a nonempty intersection with a set of prefixes from the second type of $\mathcal{R}$. All of these will impose a constraint on the relative lengths of the elements of the types from $\mathcal{S}$ and $\mathcal{R}$, which need to be taken into account, since some conditions on lengths may be more likely to be satisfied than others.

Our plan is to construct our zipper by use of a tree in which each node has up to three possible child nodes, corresponding to the three possibilities given above. Let $\mathcal{R}$ and $\mathcal{S}$ be two message types, and let p be a number between 1 and 0, such that we are attempting to determine whether the probability of constructing a type confusion between $\mathcal{R}$ and $\mathcal{S}$ is greater than p. We define a tertiary tree of sept-tuples as follows. The first entry of each sept-tuple is a set U of triples $\langle x, \bar{y}, \bar{z}\rangle$, where x is a bit-string and $\bar{y} = \langle y_1, ..., y_n\rangle$ and $\bar{z} = \langle z_1, ..., z_m\rangle$ such that $y1||...||y_n = z_1||...||z_m = x$. We will call U the *support* of the node. The second and third entries are n and m postfix message types, respectively. The fourth and fifth are message types or prefix message types. The sixth is a probability q. The seventh is a set of constraints on lengths of types. The root of the tree is of the form $\langle \phi, \mathcal{R}, \mathcal{S}, \langle\rangle, \langle\rangle, 1, D\rangle$, where D is the set of length constraints introduced by $\mathcal{R}$ and $\mathcal{S}$.

Given a node, $\langle U, \mathcal{H}, \mathcal{I}, \mathcal{J}, \mathcal{K}, q, C\rangle$, we construct up to three child nodes as follows:

1. The first node corresponds to the case in which a term from H can be confused with a term from I. Let T be the set of all $\langle x, \bar{y}, \bar{z}\rangle \in U$ such that $P(\mathcal{H}(\bar{y}) \cap \mathcal{I}(\bar{z}) \neq \phi) \cdot q > p$. Then, if T is nonempty, we construct a child node as follows:

   a. The first element of the new tuple is the set T' of all $\langle x', \bar{y}', \bar{z}'\rangle$ such that there exists $\langle x, \bar{y}, \bar{z}\rangle \in T$ such that $x' = x||y_1$, where $y_1 \in \mathcal{H}_n(\bar{y})$, $\bar{y}' = append(\bar{y}, \langle y_1\rangle)$, and $\bar{z}' = append(\bar{z}, \langle y_1\rangle)$;

   Note that, by definition y_1 is an element of $\mathcal{I}(\bar{z})$ as well as $\mathcal{H}(\bar{y})$.

   b. The second element is the (n+1)-postfix message type $\mathcal{H}\lfloor W_R$, where $W_R = \{\bar{y}' | \langle x', \bar{y}', \bar{z}'\rangle \in T'\}$;

   c. The third element is the (m+1)-postfix message type $\mathcal{I}\lfloor W_S$, where

$W_S = \{\bar{z}' | \langle x', \bar{y}', \bar{z}' \rangle \in T'\}$;

d. The fourth element is $(\mathcal{J} \sharp \mathcal{H}_n) \lceil V_R$, where $V_R = \{\bar{y} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

e. The fifth element is $(\mathcal{K} \sharp \mathcal{I}_m) \lceil V_S$, where $V_S = \{\bar{z} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

f. The sixth element is $max(\{P(\mathcal{H}_n(\bar{y}) \cap \mathcal{I}_m(\bar{z}) \neq \phi \mid \exists xs.t.(x, \bar{y}, \bar{z}) \in T)\}) \cdot q$, and;

g. The seventh element is $C \cup \{c_1\}$, where c_1 is the constraint $\text{length}(\mathcal{H}_n) = \text{length}(\mathcal{I}_n)$.

We call this first node the *node generated by the constraint* $\text{length}(\mathcal{H}_n) = \text{length}(\mathcal{I}_m)$.

2. The second node corresponds to the case in which a type from $\mathcal{H}$ can be confused wit prefix of a type from $\mathcal{I}$.

   Let T be the set of all $\langle x, \bar{y}, \bar{z} \rangle$ such that $P(\mathcal{H}_n(\bar{y}) \sqcap Pre(\mathcal{I}_m)(\bar{z})) \cdot q > p$. Then, if T is non-empty, we construct a child node as follows:

   a. The first element of the new tuple is the set T' of all $\langle x', \bar{y}', \bar{z}' \rangle$ such that there exists $\langle x, \bar{y}, \bar{z} \rangle \in T$ such that $x' = x || y_1$, where $y_1 \in \mathcal{H}_n(\bar{y})$, $\bar{y}' = append(\bar{y}.\langle y_1 \rangle)$, and $\bar{z}' = append(\bar{z}, \langle y_1 \rangle)$;

   Note that, in this case y_1 is an element of $Pre(\mathcal{I}_m)(\bar{z}))$ as well.

   b. The second element is the (n+1)-postfix message type $\mathcal{H} \lfloor W_R$, where $W_R = \{\bar{y}' | \langle x', \bar{y}', \bar{z}' \rangle \in T'\}$;

   c. The third element is the m-postfix message type $Split(\mathcal{I}) \lfloor W_S$, where $W_S = \{\bar{z}' | \langle x', \bar{y}', \bar{z}' \rangle \in T'\}$;

   d. The fourth element is $(\mathcal{J} \sharp \mathcal{H}_n) \lceil V_R$, where $V_R = \{\bar{y} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

   e. The fifth element is $(\mathcal{K} \sharp Pre(\mathcal{I}_m)) \lceil V_S$, where $V_S = \{\bar{z} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

   f. The sixth element of the tuple is $max(\{P(\mathcal{H}_n(\bar{y}) \sqcap Pre(\mathcal{I}_m)(\bar{z}) \mid \exists xs.t.(x, \bar{y}, \bar{z}) \in T))\}) \cdot q$, and;

   g. The seventh element is $C \cup \{c_1\}$, where c_1 is the constraint $\text{length}(\mathcal{H}_n) < \text{length}(\mathcal{I}_m)$.

   We call this node the *node generated by the constraint* $\text{length}(\mathcal{H}_n) < \text{length}(\mathcal{I}_m)$.

3. The third node corresponds to the case in which a prefix of a type from $\mathcal{H}$ can be confused with a type from $\mathcal{I}$.

   Let T be the set of all $\langle x, \bar{y}, \bar{z} \rangle$ in Usuch that $P(Pre(\mathcal{H}_n)(\bar{y}) \sqcap \mathcal{I}(\bar{z})) \cdot q > p$. Then, if T is nonempty, we construct a child node as follows:

   a. The first element of the new tuple is the set T' of all $\langle x', \bar{y}', \bar{z}' \rangle$ such that there exists $\langle x, \bar{y}, \bar{z} \rangle \in T$ such that $x' = x || y_1$, where $y_1 \in Pre(\mathcal{H}_n)(\bar{y})$, $\bar{y}' = append(\bar{y}, \langle y_1 \rangle)$, and $\bar{z}' = append(\bar{z}, \langle y_1 \rangle$;

   Note that, in this case y_1 is an element $\mathcal{I}_m(\bar{z})$ as well.

   b. The second element is the n-postfix message type $Split(\mathcal{H}) \lfloor W_R$, where $W_R = \{\bar{y}' | \langle x', \bar{y}', \bar{z}' \rangle \in T'\}$;

   c. The third element is the (m+1)-postfix message type $\mathcal{I} \lfloor W_S$, where $W_S = \{\bar{z}' | \langle x', \bar{y}', \bar{z}' \rangle \in T'\}$;

   d. The fourth element is $(\mathcal{J} \sharp Pre(\mathcal{H}_n))) \lceil V_R$, where $V_R = \{\bar{y} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

   e. The fifth element is $(\mathcal{K} \sharp \mathcal{I}_m) \lceil V_S$, where $V_S = \{\bar{z} | \langle x, \bar{y}, \bar{z} \rangle \in T\}$;

   f. The sixth element is $max(\{P(Pre(\mathcal{H}_n)(\bar{y}) \sqcap \mathcal{I}_m(\bar{z})) \mid \exists xs.t.(x, \bar{y}, \bar{z}) \in T)\}) \cdot q$, and;

   g. The seventh element is $C \cup \{c_1\}$, where c_1 is the constraint $\text{length}(\mathcal{H}_n) > \text{length}(\mathcal{I}_m)$.

   We call this node the *node generated by the constraint* $\text{length}(\mathcal{H}_n) > \text{length}(\mathcal{I}_m)$.

The idea behind the nodes in the tree is as follows. The first entry in the sept-tuple corresponds to the part of the zipper that we have found so far. The second and third corresponds to the portions of $\mathcal{R}$ and $\mathcal{S}$ that are still to be compared. The fourth and fifth correspond to the portions of $\mathcal{R}$ and $\mathcal{S}$ that we have compared so far. The sixth entry gives an upper bound on the probability that this portion of the zipper can be constructed by an attacker. The seventh entry gives the constraints on lengths of fields that are satisfied by this portion of the zipper.

Definition 5.1 *We say that a zipper* succeeds *if it contains a node* $\langle U, \langle\rangle, \langle\rangle, \mathcal{J}, \mathcal{K}, q, C \rangle$.

Theorem 5.1 *The zipper terminates for bounded message types, and, whether or not it terminates, it succeeds if there are any type confusions of probability greater than p. For bounded message types, the complexity is exponential in the number of message fields.*

6 An Example: An Analysis of GDOI

In this section we give a partial analysis of the signed messages of a simplified version of the GDOI protocol.

There are actually three such messages. They are: the POP signed by the group member, the POP signed by the GCKS, and the Groupkey Push Message signed by the GCKS. We will show how the POP signed by the GCKS can be confused with the Groupkey Push Message.

The POP is of the form $NONCE_A, NONCE_B$ where $NONCE_A$ is a random number generated by a group member, and $NONCE_B$ is a random number generated by the GCKS. The lengths of $NONCE_A$ and $NONCE_B$ are not constrained by the protocol. Since we are interested in the types local to the GCKS, we have $NONCE_A$ the type consisting of all numbers, and $NONCE_B$ the type local to the GCKS consisting of the the single nonce generated by the GCKS.

We can thus define the POP as a message type local to the GCKS as follows:

1. $\mathcal{R}(\langle\rangle) = NONCE_A$ where $NONCE_A$ is the type under the control of the intruder consisting of all numbers, and;
2. $\mathcal{R}(\langle y_1\rangle) = NONCE_B$ where $NONCE_B$ is a type under control of the GCKS.

We next give a simplified (for the purpose of exposition) Groupkey Push Message. We describe a version that consists only of the Header and the Key Download Payload:

$NONCE_H, \mathrm{kd}, MESSAGE\_LENGTH, \mathrm{sig},$
$KDLENGTH, KDHEADER, KEYS$

The $NONCE_H$ at the beginning of the header is of fixed length (16 bytes). The one-byte kd field gives the type of the first payload, while the 4-byte $MESSAGE\_LENGTH$ gives the length of the message in bytes. The one-byte sig field gives the type of the next payload (in this case the signature, which is not part of the signed message), while the 2-byte $KDLENGTH$ gives the length of the key download payload. We divide the key download data into two parts, a header which gives information about the keys, and the key data, which is random and controlled by the GCKS. (This last is greatly simplified from the actual GDOI specification).

We can thus define the Groupkey Push Message as the following message type local to the intended receiver:

1. $\mathcal{S}(\langle\rangle) = NONCE_H$ where $NONCE_H$ is the type consisting of all 16-byte numbers;
2. $\mathcal{S}(\langle x_1\rangle) = \{\mathrm{kd}\}$;
3. $\mathcal{S}(\langle x_1, x_2\rangle) = MESSAGE\_LENGTH$, where $MESSAGE\_LENGTH$ is the type consisting of all 4-byte numbers;
4. $\mathcal{S}(\langle x_1, x_2, x_3\rangle) = \{\mathrm{sig}\}$;
5. $\mathcal{S}(\langle x_1, x_2, x_3, x_4\rangle) = KDLENGTH$, where $KDLENGTH$ is the type consisting of all 2-byte numbers;
6. $\mathcal{S}(\langle x_1, x_2, x_3, x_4, x_5\rangle) = KDHEADER$, where the type $KDHEADER$ consists of all possible KD headers whose length is less than $x_3 - \mathrm{length}(x_1||x_2||x_3||x_4||x_5)$ and the value of x_5.
7. $\mathcal{S}(\langle x_1, x_2, x_3, x_4, x_5, x_6\rangle) = KEYS$, where $KEYS$ is the set of all numbers whose length is less than $x_3 - \mathrm{length}(x_1||x_2||x_3||x_4||x_5||x_6)$ and equal to $x_5 - \mathrm{length}(x_6)$. Note that the second constraint makes the first redundant.

All of the above types are local to the receiver, but under the control of the sender.

We begin by creating the first three child nodes. All three cases length(y_1) = length(x_1), length(y_1) < length(x_1), and length(y_1) > length(x_1), are non-trivial, since $x_1 \in NONCE_H$ is an arbitrary 16-byte number, and $y_1 \in NONCE_A$ is a completely arbitrary number. Hence the probability of $NONCE_A \sqcap NONCE_B$ is one in all cases. But let's look at the children of these nodes. For the node corresponding to length(y_1) = length(x_1), we need to compare x_2 and y_2. The term x_2 is the payload identifier corresponding to "kd". It is one byte long. The term y_2 is the random nonce $NONCE_B$ generated by the GCKS. Since y_2 is the last field in the POP, there is only one possibility; that is, length(x_2) < length(y_2). But this would require a member of $Pre(NONCE_B)$ to be equal to "kd". Since $NONCE_B$ is local to the GCKS and under its control, the chance of this is $1/2^8$. If this is not too small to worry about, we construct the child of this node. Again, there will be only one, and it will correspond to length(x_3) < length(y_2) - length(x_2). In this case, x_3 is the apparently arbitrary number $MESSAGE\_LENGTH$. But there is a nontrivial relationship between $MESSAGE\_LENGTH$ and $NONCE_B$, in that $MESSAGE\_LENGTH$ must describe a length equal to $M + N$, where M is the length of the part of $NONCE_B$ remaining after the point at which $MESSAGE\_LENGTH$ appears in it, and N describes the length of the signature payload. Since both of these lengths are outside of the intruder's control, the probability that the first part of $NONCE_B$ will have exactly this value is $1/2^{16}$. We are now up to a probability of $1/2^{24}$.

When we go to the next child node, again the only possibility is length(x_4) < length(y_2) - length(x_3) -

length(x_2), and the comparison in this case is with the 1-byte representation of "sig". The probability of type confusion now becomes $1/2^{32}$. If this is still a concern, we can continue in this fashion, comparing pieces of $NONCE_B$ with the components of the Groupkey Push Message until the risk has been reduced to an acceptable level. A similar line of reasoning works for the case length(y_1) < length(x_1).

We now look at the case length(y_1) > length(x_1), and show how it can be used to construct the attack we mentioned at the beginning of this paper. We concentrate on the child node generated by the constraint length(y_1) - length(x_1) > length(x_2). Since $y_1 \in NONCE_A$ is an arbitrary number, the probability that x_2 can be taken for a piece of y_1, given the length constraint, is one. We continue in this fashion, until we come to the node generated by the constraint length(x_7) <length(y_1) - $\sum_{i=1}^{5} x_i$. The remaining field of the Groupkey Pull Message, $x_7 \in KEYS$ is an arbitrary number, so the chance that the remaining field of the POP, y_2 together with what remains of y_1, can be mistaken for x_7, is one, since the concatenation of the remains of y_1 with y_2, by definition, will be a member of the abitrary set $KEYS$.

7 Conclusion and Discussion

We have developed a procedure for determining whether or not type confusions are possible in signed messages in a cryptographic protocol. Our approach has certain advantages over previous applications of formal methods to type confusion; we can take into account the possibility that an attacker could cause pieces of message fields to be confused with each other, as well as entire fields. It also takes into account the probability of an attack succeeding. Thus, for example, it would catch message type attacks in which typing tags, although present, are so short that it is possible to generate them randomly with a non-trivial probability.

Our greater generality comes at a cost, however. Our procedure is not guaranteed to terminate for unbounded message types, and even for bounded types it is exponential in the number of message fields. Thus, it would have not have terminated for the actual, unsimplified, GDOI protocol, which allows an arbitrary number of keys in the Key Download payload, although it still would have found the type confusion attacks that we described at the beginning of this paper.

Also, we have left open the problem of how the probabilities are actually computed, although in many cases, such as that of determining whether or not a random number can be mistaken for a formatted field, this is fairly straightforward. In other cases, as in the comparison between $NONCE_B$ and $MESSAGE\_LENGTH$ from above, things may be more tricky. This is because, even though the type of a field is a function of the fields that come before it in a message, the values of the fields that come after it may also act as a constraint, as the length of the part of the message appearing after $MESSAGE\_LENGTH$ does on the value of $MESSAGE\_LENGTH$.

Other subtleties may arise from the fact that other information that may or may not be available to the intruder may affect the probability of type confusion. For example, in the comparison between $MESSAGE\_LENGTH$ and $NONCE_B$, the intruder has to generate $NONCE_A$ before it sees $NONCE_B$. If it could generate $NONCE_A$ after it saw $NONCE_B$, this would give it some more control over the placement of $MESSAGE\_LENGTH$ with respectt to $NONCE_B$. This would increase the likelyhood that it would be able to force $MESSAGE\_LENGTH$ to have the appropriate value.

But, although we will need to deal with special cases like these, we believe that, in practice, the number of different types of such special cases will be small, and thus we believe that it should be possible to narrow the problem down so that a more efficient and easily automatable approach becomes possible. In particular, a study of the most popular approaches to formatting cryptographic protocols should yield some insights here.

8 Acknowledgements

We are greatful to MSec and SMuG Working Groups, and in particular to the authors for the GDOI protocol, for many helpful discussions on this topic. This work was supported by ONR.

References

[1] J. Alves-Foss. Provably insecure mutual authentication protocols: The two party symmetric encryption case. In *Proc. 22nd National Information Systems Security Conference.*, Arlington, VA, 1999.

[2] Mark Baugher, Thomas Hardjono, Hugh Harney, and Brian Weis. The Group Domain of Interpretation. Internet Draft draft-ietf-msec-gdoi-04.txt, Internet Engineering Task Force, February 26 2002. available at http://www.ietf.org/internet-drafts/draft-ietf-msec-gdoi-04.txt.

[3] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, Internet Engineering Task FOrce, November 1998. available at http://ietf.org/rfc/rfc2409.txt.

[4] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society Press, June 2000. A revised version is to appear in the *Journal of Computer Security*.

[5] John Kelsey and Bruce Schneier. Chosen interactions and the chosen protocol attack. In *Security Protocols, 5th International Workshop April 1997 Proceedings*, pages 91–104. Springer-Verlag, 1998.

[6] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). Request for Comments 2408, Network Working Group, November 1998. Available at http://ietf.org/rfc/rfc2408.txt.

[7] Catherine Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In *Proceedings of ESORICS '96*. Springer-Verlag, 1996.

[8] Einar Snekkenes. Roles in cryptographic protocols. In *Proceedings of the 1992 IEEE Computer Security Symposium on Research in Security and Privacy*, pages 105–119. IEEE Computer Society Press, May 4-6 1992.