How to Prevent Type Flaw Attacks on Security Protocols

James Heather* Gavin Lowe† Steve Schneider*

*Department of Computer Science
Royal Holloway, University of London
Egham Hill, Egham, Surrey, TW20 0EX, UK
{jheather,steve}@dcs.rhbnc.ac.uk

†Department of Mathematics and Computer Science
University of Leicester
University Road, Leicester, LE1 7RH, UK
gavin.lowe@mcs.le.ac.uk

Abstract

A type flaw attack *on a security protocol is an attack where a field that was originally intended to have one type is subsequently interpreted as having another type. A number of type flaw attacks have appeared in the academic literature. In this paper we prove that type flaw attacks can be prevented using a simple technique of tagging each field with some information indicating its intended type.*

1 Introduction

A *type flaw attack* on a security protocol is an attack where a field that was originally intended to have one type is subsequently interpreted as having another type. For example, consider the seven message version of the adapted Needham-Schroeder Public-Key Protocol [2]:

$$
\begin{array}{lrcl}
\text{Msg 1.} & a \to s & : & b \\
\text{Msg 2.} & s \to a & : & \{PK(b), b\}_{SK(s)} \\
\text{Msg 3.} & a \to b & : & \{na, a\}_{PK(b)} \\
\text{Msg 4.} & b \to s & : & a \\
\text{Msg 5.} & s \to b & : & \{PK(a), a\}_{SK(s)} \\
\text{Msg 6.} & b \to a & : & \{na, nb, b\}_{PK(a)} \\
\text{Msg 7.} & a \to b & : & \{nb\}_{PK(b)}.
\end{array}
$$

We adopt standard notation for protocols: a and b are agents' identities; s is the identity of a trusted server; na and nb are nonces; the functions PK and SK return an agent's public key and secret key, respectively; $\{m\}_k$ denotes m encrypted by key k. We consider the above variables to be free variables that can be instantiated with different values in different runs.

Meadows, in [4], describes a type flaw attack on the unmodified Needham-Schroeder Public-Key Protocol [5]. The adapted version of the protocol is vulnerable to essentially the same attack:

$$
\begin{array}{lrcl}
\text{Msg } \alpha.3. & I_A \to B & : & \{N_I, A\}_{PK(B)} \\
\text{Msg } \alpha.4. & B \to S & : & A \\
\text{Msg } \alpha.5. & S \to B & : & \{PK(A), A\}_{SK(S)} \\
\text{Msg } \alpha.6. & B \to I_A & : & \{N_I, N_b, B\}_{PK(A)} \\
\text{Msg } \beta.3. & I_{(N_b,B)} \to A & : & \{N_I, (N_b, B)\}_{PK(A)} \\
\text{Msg } \beta.4. & A \to I_S & : & (N_b, B) \\
\text{Msg } \alpha.7. & I_A \to B & : & \{N_b\}_{PK(B)}.
\end{array}
$$

The attack uses two runs, whose messages are labelled α and β; the notation I_A denotes the penetrator I either faking a message, apparently from A, or intercepting a message intended for A. The penetrator seeks to impersonate A throughout run α. When B issues a nonce challenge at message $\alpha.6$, the penetrator replays this message at A, as message $\beta.3$; A interprets the field (N_b, B) as being an agent's identity, and so believes this message came from (N_b, B). A therefore tries to request (N_b, B)'s public key, by sending the "identity" (N_b, B) to the server; this allows the penetrator to learn N_b, and hence respond to the nonce challenge.

In this paper, we consider a system where fields are tagged with some extra information indicating their intended type. One can think of the tag as a few bits attached to the field, with different bit patterns allocated to different types. For example, we will write "(nonce, N)" to represent a value N tagged in such a way to indicate that it is intended as a nonce. We will similarly tag compound messages; for example, we will write "$(\mathsf{pair}, ((\mathsf{nonce}, N), (\mathsf{nonce}, N')))$" to represent a pair of values N and N' tagged as nonces; we assume that the tag for a pair contains enough information to allow an honest agent to decompose the message correctly (for example, the tag might contain a representation of the number of bits in each component). We will assume that the tag for an encryption contains the type of the encrypting key and the type of the body of the encryption; for example, we will write "$\{|\mathsf{nonce}, \mathsf{nonce}|\}_{\mathsf{pubkey}}$" to represent a tag indicating an encryption of a pair of nonces with a public key.

We assume that honest agents will tag messages that they create with their *true type*; for example, if they introduce a nonce, they will tag it as being a nonce. We assume that when an honest agent receives a message, it will check that all accessible tags (i.e. those tags not inside an encryption that this agent cannot decrypt) are as expected. On the other hand, we will not assume that the penetrator follows these tagging rules—we will allow the penetrator to place arbitrary tags upon accessible messages (i.e. those messages not protected by an encryption)—and we will not assume that honest agents can detect such dishonest tagging.

Suppose we were to use this tagging scheme in the attack described above. The text of message $\alpha.6$ would then become:

$$\begin{array}{l}(\{|\mathsf{nonce}, \mathsf{nonce}, \mathsf{agent}|\}_{\mathsf{pubkey}}, \\ \qquad \{(\mathsf{nonce}, N_I), (\mathsf{nonce}, N_b), (\mathsf{agent}, B)\}_{PK(A)}).\end{array}$$

A penetrator could replace the outermost tag with

$$\{|\mathsf{nonce}, \mathsf{agent}|\}_{\mathsf{pubkey}},$$

but the tag inside the encryption cannot be tampered with. Now A will not accept this message as an instance of a message 3, because in such messages the second field inside the encryption should be tagged as being an agent's identity. Hence this tagging scheme would prevent the above attack.

As another example, consider the Woo and Lam Protocol π_1 from [10]:

$$\begin{array}{lrcl}\text{Msg 1.} & a \to b & : & a \\ \text{Msg 2.} & b \to a & : & nb \\ \text{Msg 3.} & a \to b & : & \{a, b, nb\}_{shared(a,s)} \\ \text{Msg 4.} & b \to s & : & \{a, b, \{a, b, nb\}_{shared(a,s)}\}_{shared(b,s)} \\ \text{Msg 5.} & s \to b & : & \{a, b, nb\}_{shared(b,s)}.\end{array}$$

Here $shared(a, s)$ denotes a key shared between a and s. Note that b cannot decrypt the message he receives in message 3, but instead simply includes it inside message 4. The following type flaw attack exploits this:

$$\begin{array}{lrcl}\text{Msg 1.} & I_A \to B & : & A \\ \text{Msg 2.} & B \to I_A & : & N_b \\ \text{Msg 3.} & I_A \to B & : & N_b \\ \text{Msg 4.} & B \to I_S & : & \{A, B, N_b\}_{shared(B,S)} \\ \text{Msg 5.} & I_S \to B & : & \{A, B, N_b\}_{shared(B,S)}.\end{array}$$

The penetrator replays the nonce N_b at B in message 3, which B accepts as being of the form

$$\{A, B, N_b\}_{shared(A,S)}.$$

B therefore encrypts N_b within message 4. However, this is precisely the form of message that the penetrator requires to fake message 5.

If we were to again adopt the tagging scheme, the nonce in message 2 would become (nonce, N_b). The penetrator cannot replay the nonce in this form in message 3, but can retag the nonce with the tag that B is expecting $(\{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}}, N_b)$. Message 4 would then become (dropping the "pair" tag and the pairing parentheses for ease of notation):

$$\begin{array}{l}(\{|\mathsf{agent}, \mathsf{agent}, \{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}}|\}_{\mathsf{shared\text{-}key}}, \\ \qquad \{(\mathsf{agent}, A), (\mathsf{agent}, B), \\ \qquad\quad (\{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}}, N_b)\}_{shared(B,S)}).\end{array}$$

But this message could not now be replayed as an instance of message 5, because B is expecting a message where the third field inside the encryption is tagged as being a nonce. The penetrator could change the outer most tag, to create:

$$\begin{array}{l}(\{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}}, \\ \qquad \{(\mathsf{agent}, A), (\mathsf{agent}, B), \\ \qquad\quad (\{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}}, N_b)\}_{shared(B,S)}),\end{array}$$

but the penetrator cannot change the inner tag without access to the appropriate key. Again this tagging scheme prevents the attack. Observe that the type flaw attack is prevented simply by having the participants examine the tags; they do not need to be able to tell the true types of fields.

In this paper we prove that in fact this tagging scheme is enough to prevent *all* type flaw attacks. The utility of this result to the designers and implementers of protocols should be obvious.

This result is also useful to protocol analysers. Most protocol analysis techniques adopt the *strong typing abstraction*, where all messages considered in the analysis are assumed to be well-typed. This corresponds to an assumption that all agents can "magically" tell the true types of messages. The result of this paper presents a way of justifying this apparently unrealistic abstraction.

More precisely, we show the following:

> If a protocol is secure under the strong typing abstraction, then it is secure under the tagging scheme.

In other words, this tagging scheme implements the strong typing abstraction. In fact, our approach is to show the following:

> If there is an attack upon a protocol under the tagging scheme, then there is an attack under the tagging scheme such that all fields are correctly tagged.

(We make the concept of "correctly tagged" precise in the next section, but essentially it means that all fields are tagged with a tag that represents their true type.)

In the next section we describe how we can model protocols using tagged messages. Our model is based upon the strand space model of [9]. We use the strand space framework because it provides a particularly suitable notation for the kind of reasoning required for the proof. However, the results of the paper are general and can be applied to other approaches to the analysis of security protocols (in the context of the Dolev-Yao model) such as those in [3, 4, 6, 7].

In Section 3 we prove our claim that this tagging scheme prevents all type flaw attacks. We sum up, and discuss possible implementations for the tagging scheme, in Section 4.

2 Modelling protocols

In this section we present the model we will be using to prove the main result of this paper; this model will be based upon the strand space model of [9]. We describe how we model tagged facts[1], and define what it means for a tagged fact to be correctly tagged. We then give a brief overview of the strand space model, showing how we model honest participants and penetrator capabilities. We also explain how breaches of security, and hence security properties, can be expressed in the strand space model.

2.1 Tags and facts

We assume some set $Atom$ of atomic values, partitioned into types $Agent$, $Nonce$, $PublicKey$, etc. By partitioning the types in this way, we assume that each atomic value has a unique true type: for values introduced by an honest agent, this will be the type that the honest agent intended for the value; the penetrator, however, will be allowed to tag such values with a different type.

[1] We use the term "fact" for messages or parts of messages, preferring to reserve the word "message" for complete messages of the protocol in question.

Tags

We assume that there is a tag corresponding to each base type; we will adopt obvious names for such tags. We will also assume tag "constructors" for pairing and encryption. The types of tags can be defined by:

$$\begin{aligned} Tag \ ::= \ & \mathsf{agent} \mid \mathsf{nonce} \mid \mathsf{pubkey} \mid \ldots \mid \\ & \mathsf{pair} \mid \mathsf{enc}\, Tag^* \; Tag. \end{aligned}$$

We assume that the tag for an encryption includes an indication of the encryption algorithm (e.g. DES or RSA public key encryption) that is claimed to have been used to produce the message. We include this algorithm tag because we want to be able to model the case where a key is used in the wrong algorithm. More precisely, we associate algorithms with types of keys (e.g. associating RSA public key encryption with the type of RSA public keys) and include the tag for that key type within the encryption tag. We also include the type of the body (as a sequence of tags) within the encryption tag. So $\mathsf{enc}\ \langle t_1, \ldots, t_n \rangle\ kt$ will indicate a tag that claims that the accompanying fact is encrypted with a key of type kt (using the appropriate algorithm), and where the body is a sequence of facts with tags $t_1, \ldots, t_n$; we will abbreviate this tag to $\{\!|t_1, \ldots, t_n|\!\}_{kt}$.

Tagged facts

We similarly define a type of tagged facts. We represent such tagged facts as $(Tag, Fact)$ pairs, where the tag gives the claimed type of the fact; the facts themselves are built up from atoms using constructors for pairing and encryption.

$$\begin{aligned} TaggedFact \ &::= \ Tag \times Fact \\ Fact \ &::= \ Atom \mid \\ & \quad\ \ \textsc{pair}\ TaggedFact\ TaggedFact \mid \\ & \quad\ \ \textsc{encrypt}\ Tag\ TaggedFact\ Fact. \end{aligned}$$

PAIR $tf1\ tf2$ represents a fact formed by concatenating tagged facts $tf1$ and $tf2$. For example, a correctly tagged pair of nonces $(N1, N2)$ would be written as $(\mathsf{pair}, \textsc{pair}\ (\mathsf{nonce}, N1)\ (\mathsf{nonce}, N2))$. We will write $(tf1, tf2)$ as an abbreviation for $(\mathsf{pair}, \textsc{pair}\ tf1\ tf2)$. We will drop superfluous parentheses within nested pairing, writing, for example, $(tf1, tf2, tf3)$; we will drop such parentheses completely when the pair forms the body of an encryption.

ENCRYPT $kt\ tf\ k$ represents tagged fact tf encrypted using key k and algorithm corresponding to kt; we will abbreviate this to $\{tf\}_k^{kt}$, and will tend to drop the kt where it matches the type of k.

We adopt a version of the *perfect encryption assumption* [3], that is, that an honest agent can tell whether it has correctly decrypted a message, i.e. whether the decrypting key and algorithm it used to decrypt the message correspond to the encrypting key and algorithm used to create the message. This can be implemented by including sufficient redundancy within the encryption.

Note also that the type of the body of an encryption is included both inside and outside the encryption, for example:

$$(\{|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\}_{\mathsf{shared\text{-}key}},\ \{(\mathsf{agent}, A), (\mathsf{agent}, B), (\mathsf{nonce}, N_b)\}_{shared(B,S)}),$$

which is a shorthand for

$$\begin{array}{l}(\ \mathsf{enc}\ \langle \mathsf{agent}, \mathsf{agent}, \mathsf{nonce}\rangle\ \mathsf{shared\text{-}key},\\ \quad \text{ENCRYPT}\ \mathsf{shared\text{-}key}\\ \qquad\qquad (\mathsf{pair}, \text{PAIR}\ (\mathsf{pair}, \text{PAIR}\ (\mathsf{agent}, A)\ (\mathsf{agent}, B))\\ \qquad\qquad\qquad\qquad (\mathsf{nonce}, N_b))\\ \qquad\qquad shared(B, S)\ ).\end{array}$$

We will often want to talk about the tag or fact components from a tagged fact, so we define projection functions as follows:

$$(t, f)_1 \mathrel{\widehat{=}} t, \qquad (t, f)_2 \mathrel{\widehat{=}} f.$$

Subfacts

We will want to talk about the sub-tagged-facts of a tagged fact. The subfact relation is defined as the smallest relation such that:

- $tf \sqsubset tf$;
- $tf \sqsubset (t, (tf1, tf2))$ if $tf \sqsubset tf1$ or $tf \sqsubset tf2$;
- $tf \sqsubset (t, \{tf'\}_k)$ if $tf \sqsubset tf'$.

We will also want to talk about the sub-untagged-facts of a tagged fact. We will write $f \sqsubset tf$ if $(t, f) \sqsubset tf$ for some tag t.

Correct tagging

We now define what it means for a tagged fact to be correctly tagged. We define this inductively over the structure of tags, as follows:

$$\begin{array}{l}\mathsf{well\text{-}tagged}(\mathsf{agent}, x) \Leftrightarrow x \in Agent,\\ \mathsf{well\text{-}tagged}(\mathsf{nonce}, x) \Leftrightarrow x \in Nonce,\\ \qquad \ldots\\ \mathsf{well\text{-}tagged}(\mathsf{pair}, x) \Leftrightarrow\\ \quad \exists tf1, tf2 : TaggedFact \bullet\\ \qquad x = \text{PAIR}\ tf1\ tf2\\ \qquad \wedge\ \mathsf{well\text{-}tagged}\ tf1 \wedge \mathsf{well\text{-}tagged}\ tf2,\\ \mathsf{well\text{-}tagged}(\{|ts|\}_{kt}, x) \Leftrightarrow\\ \quad \exists tf : TaggedFact\,;\, k : Fact \bullet\\ \qquad x = \{tf\}_k^{kt} \wedge \mathsf{well\text{-}tagged}(tf)\\ \qquad \wedge\ \mathsf{well\text{-}tagged}(kt, k) \wedge ts = \mathsf{get\text{-}tags}\ tf,\end{array}$$

where get-tags returns the sequence of tags labelling the body of an encryption:

$$\begin{array}{rcl}\mathsf{get\text{-}tags}(\mathsf{pair}, (tf1, tf2)) &=& \mathsf{get\text{-}tags}\ tf1 ^\frown \mathsf{get\text{-}tags}\ tf2,\\ \mathsf{get\text{-}tags}(t, f) &=& \langle t\rangle, \qquad \text{for } t \neq \mathsf{pair}.\end{array}$$

Note that for an encryption, the tag for the key must match the encryption algorithm used and also the type of the key used.

It is also useful to characterize when a message is correctly tagged at the outermost level:

$$\begin{array}{l}\mathsf{top\text{-}level\text{-}well\text{-}tagged}(\mathsf{agent}, x) \Leftrightarrow x \in Agent,\\ \mathsf{top\text{-}level\text{-}well\text{-}tagged}(\mathsf{nonce}, x) \Leftrightarrow x \in Nonce,\\ \qquad \ldots\\ \mathsf{top\text{-}level\text{-}well\text{-}tagged}(\mathsf{pair}, x) \Leftrightarrow\\ \quad \exists tf1, tf2 : TaggedFact \bullet x = \text{PAIR}\ tf1\ tf2,\\ \mathsf{top\text{-}level\text{-}well\text{-}tagged}(\{|ts|\}_{kt}, x) \Leftrightarrow\\ \quad \exists tf : TaggedFact\,;\, k : Fact \bullet\\ \qquad x = \{tf\}_k^{kt} \wedge ts = \mathsf{get\text{-}tags}\ tf.\end{array}$$

Note that for an encryption, the key tag must match the algorithm used for the encryption, but not necessarily the key used.

2.2 Strand spaces

As mentioned earlier, we will be using the strand space model from [9]. We give here a brief overview of the strand space model, and how we adapt it to deal with tagged facts.

A *strand* represents a sequence of communications by either an honest agent or the penetrator. Formally, it is a sequence of the form $\langle \pm tf_1, \pm tf_2, ..., \pm tf_n\rangle$, where $+tf$ represents the transmission of tagged fact tf and $-tf$ represents reception of tf. A *node* is any particular communication $\pm tf$.

A graph structure is defined on strands by means of two types of edge:

- If nodes n_i and n_{i+1} are consecutive nodes on the same strand then we write $n_i \Rightarrow n_{i+1}$. This represents the chronological sequence of communications along a strand.
- If node $n_i = +tf$ and $n_j = -tf$ then we write $n_i \rightarrow n_j$. This captures communications from one strand to another.

A *bundle* represents a particular history of the network. Formally, if $\mathcal{C} \subseteq (\rightarrow \cup \Rightarrow)$ is a finite set of edges, and $\mathcal{N}_\mathcal{C}$ the set of nodes appearing on any edge in $\mathcal{C}$, then $\mathcal{C}$ is a *bundle* if:

1. whenever $n_2 \in \mathcal{N}_\mathcal{C}$ and n_2 has negative sign, there exists a unique n_1 such that $n_1 \rightarrow n_2 \in \mathcal{C}$;

2. whenever $n_2 \in \mathcal{N}_\mathcal{C}$ and $n_1 \Rightarrow n_2$, we have $n_1 \Rightarrow n_2 \in \mathcal{C}$;

3. $\mathcal{C}$ is acyclic.

We will want to be able to talk about when a fact or tagged fact is first transmitted. If S is a set of tagged facts, then a node n is an *entry point* to S if the term of n is $+tf$ for some $tf \in S$, and for each node n' previous to n on the same strand, the term of n' is not in S. A tagged fact tf will be said to *originate* on a node n if n is an entry point to the set $\{tf' \mid tf \sqsubset tf'\}$. Similarly, an *untagged* fact f will be said to *originate* on a node n if n is an entry point to the set $\{tf' \mid f \sqsubset tf'\}$. An untagged fact is *uniquely originating* in a bundle $\mathcal{C}$ if it originates on a unique node of $\mathcal{C}$.

2.3 Honest strands

We assume that each role in the protocol is defined by a *strand template*: a sequence of templates for sent and received tagged facts, defining the operation of the agent in that role.

The templates for tagged facts will make use of some set Var of variables, and some set Fn of function identifiers (which will contain functions like PK, the public key function). Tagged fact templates can be defined as follows:

$$
\begin{aligned}
&TaggedTemplate ::= Tag \times Template\\
&Template ::=\\
&\quad Var \mid \text{APPLY}\ Fn\ Var^* \mid\\
&\quad \text{PAIR}\ TaggedTemplate\ TaggedTemplate \mid\\
&\quad \text{ENCRYPT}\ Tag\ TaggedTemplate\ Template.
\end{aligned}
$$

The template APPLY $g\ \langle v_1, \ldots, v_n\rangle$ represents the function g applied to the variables $v_1, \ldots, v_n$; we will denote this $g(v_1, \ldots, v_n)$.

For example, the role played by b in the Woo and Lam Protocol π_1 would be defined by the following sequence of templates (where we are adopting the same notational conventions as earlier):

$$
\begin{aligned}
&temp \mathrel{\widehat{=}}\\
&\quad \langle -(\mathsf{agent}, a),\\
&\quad\ +(\mathsf{nonce}, nb),\\
&\quad\ -(\{\!|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\!\}_{\mathsf{shared\text{-}key}}, x),\\
&\quad\ +(\{\!|\mathsf{agent}, \mathsf{agent},\\
&\quad\qquad \{\!|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\!\}_{\mathsf{shared\text{-}key}}|\!\}_{\mathsf{shared\text{-}key}},\\
&\quad\qquad \{(\mathsf{agent}, a), (\mathsf{agent}, b),\\
&\quad\qquad\quad (\{\!|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\!\}_{\mathsf{shared\text{-}key}}, x)\}_{shared(b,s)}),\\
&\quad\ -(\{\!|\mathsf{agent}, \mathsf{agent}, \mathsf{nonce}|\!\}_{\mathsf{shared\text{-}key}},\\
&\quad\qquad \{(\mathsf{agent}, a), (\mathsf{agent}, b), (\mathsf{nonce}, nb)\}_{shared(b,s)})\\
&\quad \rangle.
\end{aligned}
$$

Note that this strand uses five free variables: a, b, s, nb and x; and also the function $shared$. Note also that the encrypted component of the third message, which b does not decrypt, is represented by a variable x, modelling that b should accept *any* value for this component.

All strands representing an execution of a particular role can be formed by instantiating the free variables of the corresponding template, that is, by substituting the free variables consistently with facts.

For example, in the case of the template above, a typical execution (where there has been no interference from the penetrator) can be formed using the substitution function sub where:

$$
\begin{aligned}
&sub(a) = A,\ sub(b) = B,\ sub(s) = S,\ sub(nb) = N_b,\\
&\ sub(x) =\\
&\qquad \{(\mathsf{agent}, A), (\mathsf{agent}, B), (\mathsf{nonce}, N_b)\}_{shared(B,S)}.
\end{aligned}
$$

The execution in the attack described in Section 1 can be formed using the substitution function:

$$
\begin{gathered}
sub(a) = A, \quad sub(b) = B, \quad sub(s) = S,\\
sub(nb) = N_b, \quad sub(x) = N_b.
\end{gathered}
$$

(We argued above that it is not possible to produce penetrator strands to complete this attack.)

Implicit in the definition of a strand template is the assumption that when an honest agent first sees a free variable in a message that it receives, it will accept any value for the variable.

Formally, a substitution is a function:

$$sub : Var \to Fact.$$

Such a function can be lifted to complete tagged templates as follows[2]:

$$
\begin{aligned}
sub(t, v) &= (t, sub(v)), \qquad \text{for } v \in Var,\\
sub(t, g(v_1, \ldots, v_n)) &= (t, g(sub(v_1), \ldots, sub(v_n))),\\
&\qquad \text{for } g \in Fn, \text{ where the}\\
&\qquad \text{function application is}\\
&\qquad \text{defined,}\\
sub(\mathsf{pair}, (tf1, tf2)) &= (\mathsf{pair}, (sub(tf1), sub(tf2))),\\
sub(\{\!|ts|\!\}_{tk}, \{tf\}_k) &= \{\!|ts|\!\}_{tk}, \{sub(tf)\}_{sub(tk,k)_2}),
\end{aligned}
$$

and hence lifted to strand templates by applying sub to each message in turn.

We will assume that each strand template is consistently tagged, in the sense that the same tags are always given to the same variables. Formally, we define a type environment to be a function:

$$\rho : (Var \to Tag) \cup (Fn \to Tag^* \times Tag)$$

The idea is:

[2] Recall that a subscript "2" extracts the fact component of a tagged fact.

- for $v \in Var$, $\rho(v)$ gives the tag that ρ should receive, or, equivalently, the type with which v should be instantiated;

- for $g \in Fn$, $\rho(g)$ will be a pair of the form
$$(\langle t_1, \ldots, t_n \rangle, t),$$
where $t_1, \ldots, t_n$ give the types of the arguments of f, and t gives the type of the result.

We can then define a tagged template to be well tagged with respect to ρ as follows:

$$\begin{aligned}
&\mathsf{well\text{-}tagged}_\rho(t, v) \Leftrightarrow t = \rho(v), \qquad \text{for } v \in Var,\\
&\mathsf{well\text{-}tagged}_\rho(t, g(v_1, \ldots, v_n)) \Leftrightarrow\\
&\quad \rho(g) = (\langle \rho(v_1), \ldots, \rho(v_n) \rangle, t), \qquad \text{for } g \in Fn,\\
&\mathsf{well\text{-}tagged}_\rho(t, \text{PAIR } tt1\ tt2) \Leftrightarrow\\
&\quad t = \mathsf{pair} \wedge \mathsf{well\text{-}tagged}_\rho(tt1) \wedge \mathsf{well\text{-}tagged}_\rho(tt2),\\
&\mathsf{well\text{-}tagged}_\rho(t, \{tt\}_k^{kt}) \Leftrightarrow\\
&\quad t = \{|\mathsf{get\text{-}tags}(tt)|\}_{kt}\\
&\quad \wedge \mathsf{well\text{-}tagged}_\rho(tt) \wedge \mathsf{well\text{-}tagged}_\rho(kt, k),
\end{aligned}$$

where get-tags is defined analogously to earlier. Our assumption about strand templates being consistently tagged can be captured as follows:

Assumption 1. For each strand template $temp$, there is some type environment ρ such that all the tagged fact templates of $temp$ are well tagged with respect to ρ.

One corollary of this assumption is that each variable in a strand template receives a unique tag.

We will assume that the functions in Fn are partial, and only defined when they are applied to arguments of the correct types. For example, if we have a function PK that is intended to return an agent's public key, $PK(N)$ will not be defined for a nonce N. We assume that if the arguments $v_1, \ldots, v_n$ of a function g have tags $t_1, \ldots, t_n$, then $g(x_1, \ldots, x_n)$ will be defined only if $(t_1, x_1), \ldots, (t_n, x_n)$ are all well tagged.

We assume that the honest agents correctly follow the tagging scheme. This can be encapsulated in the following assumption:

Assumption 2. If the tagged fact (t, f) originates on a regular strand, then $\mathsf{top\text{-}level\text{-}well\text{-}tagged}(t, f)$.

This assumption has a number of facets:

- If an agent introduces an atomic term for a variable, then it introduces a value of the expected type.

- An honest agent will only tag a fact as being a pair if it was indeed created as a pair.

- An honest agent will only tag a fact as being an encryption if it is indeed created as an encryption; in this case the encryption tag will include the identity of the algorithm used, and the tags of the body; however, the key used for the encryption might not be of the expected type, because the honest agent might have received an ill-tagged key from the penetrator.

The above discussion has suggested that the bundle under consideration contains honest strands from a *single* protocol. In fact, this is not necessary: our results apply equally well when we consider bundles containing honest strands from several different protocols (as considered in [8]).

2.4 Penetrator strands

Following [9], we assume that there is some set T of messages that the penetrator can produce himself. In [9] this is a set of atomic values; in contrast, we assume some larger set, including some compound facts; we will say more about this set below. We will also assume some set K_P of keys that the penetrator has available.

Penetrator strands under the tagging scheme are exactly analogous to as in the standard strands model, but with the addition of a type of strand **R** representing manipulation of top-level tags. A penetrator strand is one of the following:

M ***Text message*** $\langle +(t, x) \rangle$ for $x \in T$, with (t, x) well-tagged.

F ***Flushing*** $\langle -tf \rangle$.

T ***Tee*** $\langle -tf, +tf, +tf \rangle$.

C ***Concatenation*** $\langle -tf,\ -tf',\ +(\mathsf{pair}, (tf, tf')) \rangle$.

S ***Separation*** $\langle -(\mathsf{pair}, (tf, tf')),\ +tf,\ +tf' \rangle$

K ***Key*** $\langle +(tk, k) \rangle$ with $\mathsf{well\text{-}tagged}(tk, k)$ and $k \in K_P$.

E ***Encryption*** $\langle -(tk, k),\ -tf,\ +(\{|ts|\}_{tk}, \{tf\}_k^{tk}) \rangle$, where $ts = \mathsf{get\text{-}tags}(tf)$.

D ***Decryption*** $\langle -(tk', k'),\ -(\{|t|\}_{tk}, \{tf\}_k^{tk}),\ +tf \rangle$, where tk and tk' are tags representing inverse key types, and k' is the decrypting key corresponding to k when they are considered as keys of types tk' and tk, respectively.

R ***Retagging*** $\langle -(t, f), +(t', f) \rangle$.

Note that the retagging strand only applies to the top level tag, and that all other strands do not interfere with the tags of their messages. However, this does not necessarily prevent inner components from being retagged; for example, one component of a pair can be retagged by separation, retagging that component, and then concatenation.

Note that when the penetrator produces a fact, it is initially correctly tagged at the top level:

Lemma 1. If fact f originates on a penetrator strand, then it does so with a tag t such that top-level-well-tagged(t, f).

Proof: The only strands that can be the origin of a fact are **M, C, K**, and **E**; in each case, the fact produced is indeed well-tagged at the top level. □

Of course, the above does not prevent the penetrator from changing the tag after the fact is produced; the following lemma shows that the only place this retagging can occur is on a retagging strand.

Lemma 2. Every top-level-ill-tagged fact (t, f) originates on a **R** strand.

Proof: Assumption 2 tells us that top-level-ill-tagged facts do not originate on honest strands. An argument similar to that in the previous lemma rules out all penetrator strands other than **R** strands. □

2.5 Security properties and attacks

In this section we consider two typical security properties, namely secrecy and authentication, and produce generic definitions of each. Our definitions are obtained by generalizing those of Thayer Fábrega et al. [9], who consider these properties for specific protocols.

The definitions given by Thayer Fábrega et al. talk about properties that should hold of the protocol under the assumption that the penetrator does not have access to certain sets of keys. For example, the properties one would expect to hold will depend on whether or not an agent is running the protocol with either the penetrator or someone whose secret key has been compromised. Our definition will generalize this to a set $Keys$, representing some keys that the penetrator may or may not have; this will be a set of function templates, for example $\{SK(a), SK(b)\}$, representing the secret keys of the agents a and b. Given a substitution function sub for a particular strand, each key $k \in Keys$ will take the value $sub(k)$; recall that we assume that the penetrator has access to the set of keys K_P; hence we can say that the penetrator knows none of the keys as follows:

$$\forall k \in Keys \bullet sub(k) \notin K_P.$$

Secrecy

We now give a generic definition of secrecy. The definition will say that there is a breach of security when there is some strand (of some minimal length) where certain keys have not been compromised, and the value of a particular variable v (intended to remain secret) becomes known to the penetrator.

Definition 1. Let $temp$ be the template for some role; let (t, v) be a tagged variable of $temp$; let h be a positive integer; and let $Keys$ be a set of function templates. We define a failure of secrecy to be where each of the following holds:

1. There is a strand $s = sub(temp)$ with $\mathcal{C}$-height at least h (i.e. at least the first h messages of s appear in the bundle $\mathcal{C}$).
2. $\forall k \in Keys \bullet sub(k) \notin K_P$.
3. There is a node in $\mathcal{C}$ with label $+sub(t, v)$.

This definition is parameterized by $temp$, v, h and $Keys$; in any given protocol, one would be interested in knowing whether this property holds for some particular values of these parameters.

Authentication

We now consider the property of authentication. We consider what it means for a particular role $r2$ to be authenticated to another role $r1$. For authentication to hold, we would expect that whenever there is a strand $s1$ of $r1$, there should be a "corresponding" strand $s2$ of $r2$; these strands should agree upon the identities of the agents involved, and possibly upon the values of some other variables (e.g. a session key that is established); we capture this aspect by specifying that the strands should agree on the values of all variables from some set X[3].

Definition 2. Let $temp1$ and $temp2$ be templates for two roles; let X be a set of variables of those templates; let $h1$ and $h2$ be positive integers; and let $Keys$ be a set of function templates. We define a failure of authentication to be where each of the following holds:

1. There is a strand $s1 = sub1(temp1)$ with $\mathcal{C}$-height at least $h1$.
2. $\forall k \in Keys \bullet sub1(k) \notin K_P$.
3. There is no strand $s2 = sub2(temp2)$ with $\mathcal{C}$-height at least $h2$ such that $\forall x \in X \bullet sub1(x) = sub2(x)$.

This definition is parameterized by $temp1$, $temp2$, X, $h1$, $h2$ and $Keys$.

3 How tagging prevents type flaw attacks

In this section we prove our main result, that if there is an attack upon a protocol under the tagging scheme, then there is an attack under the tagging scheme such that all fields are correctly tagged. More precisely, we show that whenever there is an attack upon a protocol under the tagging scheme,

[3] We are assuming that these variables are given the same names in the two different templates; this is not strictly necessary, but simplifies the notation.

then we can construct a renaming function ϕ over tagged facts, such that uniformly renaming all tagged facts under this function produces an attack in which all fields are correctly tagged.

Informally, if an honest agent is willing to accept some ill-tagged fact (t, f), then that agent's behaviour must be essentially independent of f, and so he should be willing to accept *any* value in its place; in particular, he should be willing to accept the tagged fact $\phi(t, f)$ (which will have the tag t) in place of (t, f). We proceed as follows:

1. In Section 3.1 we define the properties that ϕ must satisfy, and show that such a ϕ can always be constructed.
2. In Section 3.2 we show that if S is an honest strand, then $\phi(S)$ (renaming all the facts of S by ϕ) is also an honest strand.
3. In Section 3.3 we similarly show that if S is a penetrator strand, then there are one or two penetrator strands having the same set of nodes as $\phi(S)$, although possibly with a slightly different strand structure.
4. In Section 3.4 we show that given a bundle $\mathcal{C}$, there is a corresponding bundle $\mathcal{C}'$ where all terms are well-tagged; this bundle will contain the same set of terms on its nodes as $\phi(\mathcal{C})$, although possibly with a slightly different strand structure; we also show that a fact is uniquely originating in $\mathcal{C}'$ if it is uniquely originating in $\mathcal{C}$.
5. Finally, in Section 3.5 we show that if there is an attack in $\mathcal{C}$, then there is similarly an attack in $\mathcal{C}'$.

3.1 Defining the renaming function

The following definition captures the required properties of ϕ:

Definition 3. Given a bundle $\mathcal{C}$, we define

$$\phi : TaggedFact \rightarrow TaggedFact$$

to be a *renaming function* for $\mathcal{C}$ if:

1. ϕ preserves top-level tags: if $\phi(t, f) = (t', f')$ then $t = t'$.
2. ϕ returns well-tagged terms: $\textsf{well-tagged}(\phi(tf))$.
3. ϕ is the identity function over well-tagged terms: if $\textsf{well-tagged}(tf)$ then $\phi(tf) = tf$.
4. ϕ distributes through concatenations that are top-level-well-tagged:
$$\phi(\textsf{pair}, (tf1, tf2)) = (\textsf{pair}, (\phi(tf1), \phi(tf2))).$$
5. ϕ distributes through encryptions that are top-level-well-tagged:
$$\phi(\{\!|ts|\!\}_{tk}, \{tf\}_k^{tk}) = (\{\!|ts|\!\}_{tk}, \{\phi(tf)\}_{\phi(tk,k)_2}^{tk}), \quad \text{if } ts = \textsf{get-tags}(tf).$$
6. ϕ respects inverses of keys: if k and k' are inverses of one another, when considered as keys of types tk and tk', then $\phi(tk, k)$ and $\phi(tk', k')$ are also inverses of one another, when considered as keys of types tk and tk'.
7. If (t, f) appears in $\mathcal{C}$, and (t, f) is top-level-ill-tagged, then $\phi(t, f) \in T$. (Recall that T is the set of messages the penetrator can produce himself.)
8. When ϕ is applied to a top-level-ill-tagged fact tf of $\mathcal{C}$, it produces a fact that is essentially new, that is a fact that has no subfact in common with $\phi(tf')$ for any other fact tf' of $\mathcal{C}$:
$$\begin{aligned}&\forall tf \in facts(\mathcal{C}) \bullet \\ &\quad \neg\textsf{top-level-well-tagged}(tf) \wedge f \sqsubset \phi(tf) \Rightarrow \\ &\qquad \forall tf' \in facts(\mathcal{C}) \bullet tf \not\sqsubset tf' \Rightarrow f \not\sqsubset \phi(tf'),\end{aligned}$$
where $facts(\mathcal{C})$ is all the facts and subfacts of nodes of $\mathcal{C}$.

Note that condition 8 implies that ϕ is injective over the facts of $\mathcal{C}$.

We now show that it is always possible to find such a ϕ:

Lemma 3. Given a bundle $\mathcal{C}$ there is some renaming function ϕ for $\mathcal{C}$.

Proof: The following method gives a recipe for constructing such a ϕ. We build the definition of ϕ from the bottom up, defining it over the subfacts of a fact before the fact itself. Consider, then, a tagged fact (t, f), and suppose we have defined ϕ over all subfacts of f. We use a case analysis to construct $\phi(t, f)$:

1. If $\neg\textsf{top-level-well-tagged}(t, f)$ then pick a new value f' from T (for condition 7) such that $\textsf{well-tagged}(t, f')$, and none of the atoms of f' has been used previously (for condition 8); define $\phi(t, f) = (t, f')$.

   One proviso to this concerns keys: we should define ϕ over pairs of inverse keys simultaneously. If $k1$ and $k2$ are inverses of one another, when considered as keys of types $tk1$ and $tk2$, then pick $k1'$ and $k2'$ from T such that: $\textsf{well-tagged}(tk1, k1')$ and $\textsf{well-tagged}(tk2, k2')$; $(tk1, k1')$ and $(tk2, k2')$ are inverses; and none of the atoms of $k1'$ or $k2'$ has been used previously. Then define $\phi(tk1, k1) = (tk1, k1')$ and $\phi(tk2, k2) = (tk2, k2')$.

2. If $\mathsf{well\text{-}tagged}(t, f)$ then define $\phi(t, f) = (t, f)$ (for condition 3).
3. If $(t, f) = (\mathsf{pair}, (tf1, tf2))$ then define $\phi(t, f) = (\mathsf{pair}, (\phi(tf1), \phi(tf2)))$ (for condition 4).
4. If $(t, f) = (\{\!|ts|\!\}_{tk}, \{tf\}_k^{tk})$ with $ts = \mathsf{get\text{-}tags}(tf)$ then define $\phi(t, f) = (\{\!|ts|\!\}_{tk}, \{\phi(tf)\}_{\phi(tk,k)_2}^{tk})$ (for condition 5).

The only assumption we need is that T is big enough: we need to assume that the penetrator has access to sufficient supplies of values that he can always produce a new value in step 1. In practice, it is reasonable to assume that he is always able to produce new atomic facts, and use these to create new compound facts. $\square$

3.2 Regular strands

In this section we show that if S is a regular strand, then so is $\phi(S)$. By definition, S must be an instantiation of a strand template $temp$ under a substitution function sub. Consider the strand S' formed by instantiating $temp$ using the substitution function sub' defined by:

$$sub'(v) = \phi(t, sub(v))_2 \quad \text{where } t \text{ is the unique tag for } v \text{ in } temp.$$

Note that S' is also a regular strand, from our definition. The following lemma shows that the translation from S to S' corresponds to a renaming under ϕ, i.e. $\phi(S) = S'$.

Lemma 4. Let $temp$, ϕ, sub and sub' be as above; then

$$\phi(sub(temp)) = sub'(temp).$$

Proof: Let tt be a tagged template in $temp$. We show that $\phi(sub(tt)) = sub'(tt)$. Assumption 1 allows us to proceed by induction over the structure of tt as follows:

- Case tt is a variable, say $tt = (t, v)$; then:

$$\begin{aligned} & \phi(sub(tt)) \\ = \; & \phi(t, sub(v)) \\ = \; & \langle \text{by definition of } sub' \rangle \\ & (t, sub'(v)) \\ = \; & sub'(tt) \end{aligned}$$

- Case tt is a function application, say

$$tt = (t, g(v_1, \ldots, v_n)).$$

For $sub(tt)$ to be defined, we must have that $(t_1, sub(v_1)), \ldots, (t_n, sub(v_n))$ are all correctly tagged, where $t_1, \ldots, t_n$ are the tags for $v_1, \ldots, v_n$ in tt. Hence $\phi(t_i, sub(v_i)) = (t_i, sub(v_i))$ for each i. So:

$$\begin{aligned} & \phi(sub(tt)) \\ = \; & \phi(t, g(sub(v_1), \ldots, sub(v_n))) \\ = \; & \langle tt \text{ is well tagged; using the above} \rangle \\ & (t, g(\phi(t_1, sub(v_1))_2, \ldots, \phi(t_n, sub(v_n))_2)) \\ = \; & \langle \text{definition of } sub' \rangle \\ & (t, g(sub'(v_1), \ldots, sub'(v_n))) \\ = \; & sub'(tt). \end{aligned}$$

- Case tt is a pair, say $tt = (\mathsf{pair}, (tf1, tf2))$; then:

$$\begin{aligned} & \phi(sub(tt)) \\ = \; & \phi(\mathsf{pair}, (sub(tf1), sub(tf2))) \\ = \; & \langle \text{condition 4 of Definition 3} \rangle \\ & (\mathsf{pair}, (\phi(sub(tf1)), \phi(sub(tf2)))) \\ = \; & \langle \text{inductive hypothesis} \rangle \\ & (\mathsf{pair}, (sub'(tf1), sub'(tf2))) \\ = \; & sub'(\mathsf{pair}, (tf1, tf2)). \end{aligned}$$

- Case tt is an encryption, say

$$tt = (\{\!|t'|\!\}_{kt}, \{(t', f')\}_k).$$

Then:

$$\begin{aligned} & \phi(sub(\{\!|t'|\!\}_{kt}, \{(t', f')\}_k)) \\ = \; & \phi(\{\!|t'|\!\}_{kt}, sub(\{(t', f')\}_{sub(k)}) \\ = \; & \langle \text{condition 5 of Definition 3} \rangle \\ & (\{\!|t'|\!\}_{kt}, \{\phi(sub(t', f'))\}_{\phi((kt, sub(k))_2}) \\ = \; & \langle \text{inductive hypothesis} \rangle \\ & (\{\!|t'|\!\}_{kt}, \{sub'(t', f')\}_{sub'(kt,k)_2} \\ = \; & sub'((\{\!|t'|\!\}_{kt}, \{(t', f')\}_k)). \end{aligned}$$

$\square$

3.3 Penetrator strands

We now show that given the penetrator strands of $\mathcal{C}$ and a renaming function ϕ, we can construct corresponding strands in a bundle $\mathcal{C}'$, formed by replacing each tagged fact tf in $\mathcal{C}$ by $\phi(tf)$. We consider the possible cases one by one, giving the strand S in $\mathcal{C}$ and its corresponding strand S' in $\mathcal{C}'$. In the case of the **R** strand, we will change the strand structure, but we will keep the same set of tagged facts.

M ***Text message*** Let $S = \langle +(t, x) \rangle$ with $x \in T$ and $\mathsf{well\text{-}tagged}(t, x)$. Define $S' = \langle +\phi(t, x) \rangle$, which is an **M** strand because $\phi(t, x) = (t, x)$ and $x \in T$.

F ***Flushing*** Let $S = \langle -tf \rangle$. Define $S' = \langle -\phi(tf) \rangle$, which is an **F** strand.

T ***Tee*** Let $S = \langle -tf, +tf, +tf \rangle$. Define

$$S' = \langle -\phi(tf), +\phi(tf), +\phi(tf) \rangle,$$

which is a **T** strand.

C ***Concatenation***
Let $S = \langle -tf1,\ -tf2,\ +(\mathsf{pair},(tf1,tf2))\rangle$. Define

$$S' = \langle -\phi(tf1),\ -\phi(tf2),\ +\phi(\mathsf{pair},(tf1,tf2))\rangle$$

which is a valid concatenation strand, because

$$\phi(\mathsf{pair},(tf1,tf2)) = (\mathsf{pair},(\phi(tf1),\phi(tf2)))$$

by condition 4 of Definition 3.

S ***Separation***
Let $S = \langle -(\mathsf{pair},(tf1,tf2)),\ +tf1,\ +tf2\rangle$. Define

$$S' = \langle -\phi(\mathsf{pair},(tf1,tf2)),\ +\phi(tf1),\ +\phi(tf2)\rangle$$

which is a valid separation strand, again by condition 4 of Definition 3.

K ***Key*** Let $S = \langle +(tk,k)\rangle$ with $\mathsf{well\text{-}tagged}(tk,k)$ and $k \in K_P$. Define $S' = \langle +\phi(tk,k)\rangle = \langle +(tk,k)\rangle$, which is a **K** strand.

E ***Encryption***
Let $S = \langle -(tk,k),\ -tf,\ +(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk})\rangle$ where $ts = \mathsf{get\text{-}tags}(tf)$. Define

$$S' = \langle -\phi(tk,k),\ -\phi(tf),\ +\phi(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk})\rangle$$

which is a valid encryption strand because

$$\phi(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk}) = (\{\!|ts|\!\}_{tk},\{\phi(tf)\}^{tk}_{\phi(tk,k)_2})$$

by condition 5 of Definition 3.

D ***Decryption***
Let $S = \langle -(tk',k'),\ -(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk}),\ +tf\rangle$, with k and k' inverses of one another. Define

$$S' = \langle -\phi(tk',k'),\ -\phi(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk}),\ +\phi(tf)\rangle$$

which is a valid decryption strand, because

$$\phi(\{\!|ts|\!\}_{tk},\{tf\}_k^{tk}) = (\{\!|ts|\!\}_{tk},\{\phi(tf)\}^{tk}_{\phi(tk,k)_2})$$

by condition 5 of Definition 3, and $\phi(tk',k')$ and $\phi(tk,k)$ are inverses of one another by condition 6 of Definition 3.

R ***Retagging*** Let $S = \langle -(t1,f), +(t0,f)\rangle$. We proceed in two stages. We first construct the pair of strands $\langle -\phi(t1,f)\rangle$, which is a strand of type **F**, and $\langle +\phi(t0,f)\rangle$. If $\neg\mathsf{top\text{-}level\text{-}well\text{-}tagged}(t0,f)$ then this latter strand is of type **M**, from condition 7 of Definition 3, and we are done; see Figure 1 for a depiction of the strands in $\mathcal{C}$ and $\mathcal{C}'$.

Otherwise, $\mathsf{top\text{-}level\text{-}well\text{-}tagged}(t0,f)$, i.e. S retags f with a correct tag. Let $n0$ and $n0'$ be the negative and positive nodes of S, respectively. See Figure 2. We show that some earlier **R** strand, possibly this one, has initial node labelled with $-(t0,f)$. If $t1 = t0$ we are done. Otherwise $(t1,f)$ is top-level-ill-tagged, and so $(t1,f)$ originated on another **R** strand, from Lemma 2. We proceed similarly with this **R** strand: let $n1$ and $n1'$ be the two nodes, and let the term on the first node be $-(t2,f)$; if $t2 = t0$ we are done; otherwise $(t2,f)$ originated on another **R** strand. Continuing in this way, we can form a sequence of earlier and earlier **R** strands, each retagging f. Because the bundle is finite, this process must eventually stop, by reaching an **R** strand where the first node, call it nk, has label $-(t0,f)$. Let n be the predecessor under $\longrightarrow$ of nk.

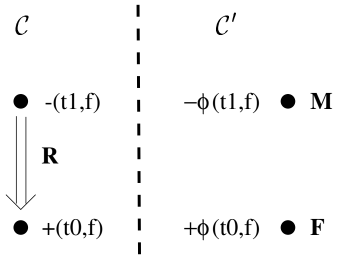

Figure 1. Replacing R strands (part 1)

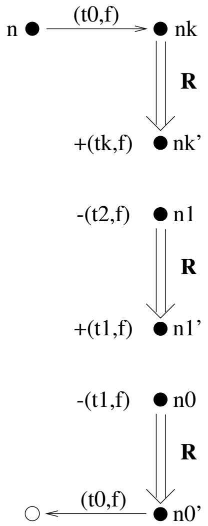

Figure 2. Replacing R strands (part 2)

We can construct new strands in $\mathcal{C}'$ as in Figure 3. Nodes $n0'$ and nk are removed; each of the other negative nodes on the **R** strands is replaced by an **F** strand; each of the other positive nodes on the **R** strands is

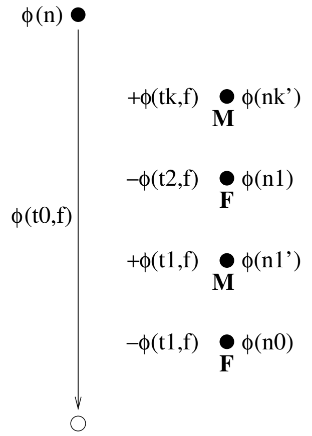

Figure 3. Replacing R strands (part 3)

replaced by an **M** strand (these nodes were ill-tagged in $\mathcal{C}$, so the corresponding facts in $\mathcal{C}'$ are elements of T); if $n0'$ had a $\longrightarrow$ successor in $\mathcal{C}$ then the corresponding node in $\mathcal{C}'$ becomes the $\longrightarrow$ successor of $\phi(n)$.

3.4 Bundles and unique origination

We have described, above, how the nodes from bundle $\mathcal{C}$ are replaced with nodes in bundle $\mathcal{C}'$. The graph structure of $\mathcal{C}'$ is mostly the same as that of $\mathcal{C}$, with each edge of the form $+n \longrightarrow -n$ in $\mathcal{C}$ replaced by an edge $+\phi(n) \longrightarrow -\phi(n)$ in $\mathcal{C}'$; the exception concerns **R** strands, which is dealt with above.

We now consider the question of unique origination. We produce a bundle $\mathcal{C}''$ with the same honest strands as $\mathcal{C}'$, but where no term is non-uniquely originating in the new bundle unless it was non-uniquely originating in $\mathcal{C}$.

Firstly, note that if fact f_0 originates on an honest node in $\mathcal{C}'$, then f_0 originates on the corresponding honest node in $\mathcal{C}$. There are three circumstances under which f_0 can originate on a penetrator strand in $\mathcal{C}'$:

1. f_0 originates on an **M** strand corresponding to an occurrence of f_0 on the corresponding **M** strand in $\mathcal{C}$;

2. f_0 originates on a **K** strand corresponding to an occurrence of f_0 on the corresponding **K** strand in $\mathcal{C}$;

3. f_0 originates on an **M** strand corresponding to an occurrence of a top-level-ill-tagged term (t, f) on an **R** strand in $\mathcal{C}$, with $f_0 \sqsubset \phi(t, f)$.

The first two possibilities do not raise any problems, for if the term originates multiple times in $\mathcal{C}'$ in these ways, then f_0 originates multiple times in $\mathcal{C}$. However, the third possibility introduces an origin not present in $\mathcal{C}$. We produce a new bundle $\mathcal{C}''$ by replacing the **M** strands from $\mathcal{C}'$ appropriately.

Note that if one origin of f_0 corresponds to case 3, above, then *all* origins of f_0 will correspond to case 3, because of condition 8 of Definition 3. Suppose there are multiple such origins for f_0 in $\mathcal{C}'$, say k in number. We form $\mathcal{C}''$ by replacing the k such **M** strands by a single **M** strand and $k-1$ **T** strands, as illustrated in Figure 4 in the case $k = 3$.

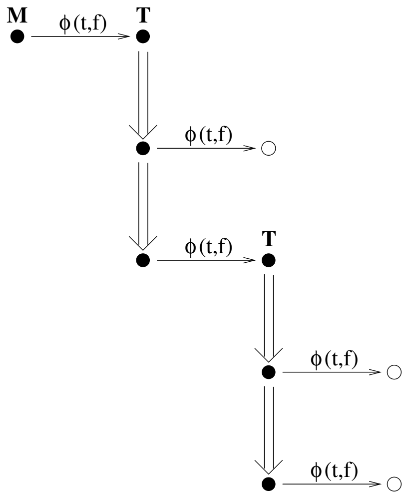

Figure 4. Achieving unique origination

The results of the previous four subsections can be summarized in the following theorem:

Theorem 1. If $\mathcal{C}$ is a bundle (under our tagging scheme) then there is a renaming function ϕ and a bundle $\mathcal{C}''$, such that:

- $\mathcal{C}''$ contains the tagged facts of $\mathcal{C}$ (considered as a set), renamed by ϕ;

- $\mathcal{C}''$ contains the same honest strands as $\mathcal{C}$, modulo the above renaming;

- facts are uniquely originating in $\mathcal{C}''$ if they were uniquely originating in $\mathcal{C}$;

- all tagged facts in $\mathcal{C}''$ are well-tagged.

3.5 Attacks

We now show that attacks upon the protocol are preserved by the transformation above. We show that if there is an attack in $\mathcal{C}$, then there is a corresponding attack in $\mathcal{C}''$; that is, essentially the same attack will work, but using well-tagged facts.

Secrecy

Following Definition 1, suppose there is an failure of secrecy in $\mathcal{C}$ as follows:

1. There is a strand $s = sub(temp)$ with $\mathcal{C}$-height at least h.
2. $\forall k \in Keys \bullet sub(k) \notin K_P$.
3. There is a node n with label $+sub(t, v)$.

We show that there is a corresponding attack in $\mathcal{C}''$. Let substitution sub' be defined as in Section 3.2:

$$sub'(v) = \phi(t, sub(v))_2, \quad \text{where } t \text{ is the unique tag for } v \text{ in } temp.$$

Then:

1. There is a strand $s' = sub'(temp) = \phi(sub(temp))$ with $\mathcal{C}''$-height at least h, corresponding to s, from the way we have constructed the honest strands of $\mathcal{C}''$.
2. $\forall k \in Keys \bullet sub'(k) \notin K_P$, because $sub'(k) = sub(k)$ for such k.
3. The node corresponding to n will have label
$$+sub'(t, v) = \phi(sub(t, v)).$$

Authentication

Following Definition 2, suppose there is an failure of authentication in $\mathcal{C}$ as follows:

1. There is a strand $s1 = sub1(temp1)$ with $\mathcal{C}$-height at least $h1$.
2. $\forall k \in Keys \bullet sub1(k) \notin K_P$.
3. There is no strand $s2 = sub2(temp2)$ with $\mathcal{C}$-height at least $h2$ such that $\forall x \in X \bullet sub1(x) = sub2(x)$.

We show that there is a corresponding attack in $\mathcal{C}''$. Let substitution $sub1'$ be defined as in Section 3.2:

$$sub1'(v) = \phi(t, sub1(v))_2, \quad \text{where } t \text{ is the unique tag for } v \text{ in } temp1.$$

Then:

1. There is a strand
$$s1' = sub1'(temp1) = \phi(sub1(temp1))$$
with $\mathcal{C}''$-height at least $h1$, corresponding to $s1$, from the way we have constructed the honest strands of $\mathcal{C}''$.
2. $\forall k \in Keys \bullet sub1'(k) \notin K_P$, because $sub1'(k) = sub1(k)$ for such k.
3. There is no strand $s2' = sub2'(temp2)$ with $\mathcal{C}''$-height at least $h2$ such that $\forall x \in X \bullet sub1'(x) = sub2'(x)$. Suppose there were such an $s2'$; then, by the way we have constructed the honest strands in $\mathcal{C}''$, $s2'$ would correspond to some strand $s2'' = sub2''(temp2)$ with $\mathcal{C}$-height at least $h2$ such that:
$$\forall v \in Var \bullet sub2'(v) = \phi(t, sub2''(v))_2 \quad \text{where } t \text{ is the unique tag for } v \text{ in } temp2.$$
But then we would have for every $x \in X$:
$$\begin{aligned} \phi(t, sub1(x))_2 &= sub1'(x) \\ &= sub2'(x) \\ &= \phi(t, sub2''(x))_2 \end{aligned}$$
where t is the tag for x in $temp1$ and $temp2$. But then by the injectivity of ϕ we would have $sub1(x) = sub2''(x)$, contradicting part 3 of the assumption.

3.6 Example: the adapted Needham-Schroeder Public-Key Protocol

We now show how the results of this paper apply to the adapted Needham-Schroeder Public-Key Protocol, considered in Section 1. The two roles of this protocol can be defined by the strand templates:

$$\begin{aligned}
&Init \mathrel{\widehat{=}} \\
&\quad \langle +(\{\!|\mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public\text{-}key}}, \\
&\qquad \{(\mathsf{nonce}, na), (\mathsf{agent}, a)\}_{PK(b)}), \\
&\quad -(\{\!|\mathsf{nonce}, \mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public\text{-}key}}, \\
&\qquad \{(\mathsf{nonce}, na), (\mathsf{nonce}, nb), (\mathsf{agent}, a)\}_{PK(a)}), \\
&\quad +(\{\!|\mathsf{nonce}|\!\}_{\mathsf{public\text{-}key}}, \{(\mathsf{nonce}, nb)\}_{PK(b)})\rangle, \\
&Resp \mathrel{\widehat{=}} \\
&\quad \langle -(\{\!|\mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public\text{-}key}}, \\
&\qquad \{(\mathsf{nonce}, na), (\mathsf{agent}, a)\}_{PK(b)}), \\
&\quad +(\{\!|\mathsf{nonce}, \mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public\text{-}key}}, \\
&\qquad \{(\mathsf{nonce}, na), (\mathsf{nonce}, nb), (\mathsf{agent}, a)\}_{PK(a)}), \\
&\quad -(\{\!|\mathsf{nonce}|\!\}_{\mathsf{public\text{-}key}}, \{(\mathsf{nonce}, nb)\}_{PK(b)})\rangle.
\end{aligned}$$

The analysis of this protocol in [9] establishes a number of properties of the protocol, under the strong typing abstraction, and under the additional assumptions that the responder never introduces the same value for nb as that received for na, and that nonces are uniquely originating.

For example, Proposition 5.2 of that paper establishes, the following 'responder's guarantee': in any bundle in which there is a responder's strand $s1 = sub1(Resp)$ such that $sub1(SK(a)) \notin K_P$, there is a corresponding initiator's strand $s2 = sub2(Init)$ such that $sub1$ and $sub2$

agree on a, b, na and nb. This means that there is no failure of authentication under our Definition 2 (with $temp1 = Resp$, $temp2 = Init$, $X = \{a, b, na, nb\}$, $h1 = 3$, $h2 = 3$ and $Keys = \{SK(a)\}$ in the notation of that definition).

We can immediately use the main result of this paper to deduce that the tagging scheme ensures that there is still no failure of authentication when the strong typing abstraction is dropped.

4 Conclusions

In this paper, we have shown that all type flaw attacks may be cheaply prevented by tagging each field with its intended type, and having honest participants check the tags of incoming messages. Our results generate no extra work in implementing protocols save adding a few extra bits of information into each message of the protocol; they generate no extra work in protocol analysis: protocols that have been proved correct under the strong typing abstraction are automatically secure under our scheme.

We have considered the properties of secrecy and authentication, but would expect the results to apply for other security properties, such as anonymity and non-repudiation, that can also be expressed in terms of correspondences between agent strands.

The key idea of the proof of the result was to show that any bundle corresponding to runs of the protocol can be transformed into another bundle in which all messages are correctly tagged, with equivalent strands of the protocol agents, and equivalent information available to the penetrator. Hence if there is such a bundle corresponding to an attack, then its transformation corresponds to an essentially similar but well-tagged attack, demonstrating that the attack is not based around a type flaw. Such a transformation is possible because of the requirement that protocol agents check all tags they have access to. Ill-tagged messages will not effect the agent's behaviour in any essential way, and so might as well be replaced by messages that do have the correct tags. This is what is achieved by the renaming function ϕ.

4.1 Implementing the tagging scheme

One possible approach to implementation would be to follow the structure of the Tag type. We could construct distinct bit patterns for each atomic type, and two more bit patterns to indicate a pair or encryption. Compound types could then be represented by concatenations of these basic bit patterns.

However, this would be somewhat inefficient, especially with complex types. A better approach is to identify all of the different types which are used in the execution of the protocol, and assign a unique tag number to each (with each tag number containing the same number of bits, to ensure unique readability).

Recall the seven message version of the adapted Needham-Schroeder Public-Key Protocol:

$$
\begin{array}{lll}
\text{Msg 1.} & a \to s & : \; b \\
\text{Msg 2.} & s \to a & : \; \{PK(b), b\}_{SK(s)} \\
\text{Msg 3.} & a \to b & : \; \{na, a\}_{PK(b)} \\
\text{Msg 4.} & b \to s & : \; a \\
\text{Msg 5.} & s \to b & : \; \{PK(a), a\}_{SK(s)} \\
\text{Msg 6.} & b \to a & : \; \{na, nb, b\}_{PK(a)} \\
\text{Msg 7.} & a \to b & : \; \{nb\}_{PK(b)}.
\end{array}
$$

This uses eight distinct types, and to each of these we could give a different tag number, represented in three bits:

$$
\begin{array}{rcl rcl}
\mathsf{nonce} & = & \mathtt{0}, & \{\!|\mathsf{public}, \mathsf{agent}|\!\}_{\mathsf{secret}} & = & \mathtt{4}, \\
\mathsf{agent} & = & \mathtt{1}, & \{\!|\mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public}} & = & \mathtt{5}, \\
\mathsf{public} & = & \mathtt{2}, & \{\!|\mathsf{nonce}, \mathsf{nonce}, \mathsf{agent}|\!\}_{\mathsf{public}} & = & \mathtt{6}, \\
\mathsf{pair} & = & \mathtt{3}, & \{\!|\mathsf{nonce}|\!\}_{\mathsf{public}} & = & \mathtt{7}.
\end{array}
$$

These type identifiers would then appear as the initial part of each message, and inside pair types and encryption types. Under this approach, the implementation would become:

$$
\begin{array}{lll}
1. & a \to s & : \; (\mathtt{1}, b) \\
2. & s \to a & : \; (\mathtt{4}, \{(\mathtt{3}, ((\mathtt{2}, PK(b)), (\mathtt{1}, b)))\}_{SK(s)}) \\
3. & a \to b & : \; (\mathtt{5}, \{(\mathtt{3}, ((\mathtt{0}, na), (\mathtt{1}, a)))\}_{PK(b)}) \\
4. & b \to s & : \; (\mathtt{1}, a) \\
5. & s \to b & : \; (\mathtt{4}, \{(\mathtt{3}, ((\mathtt{2}, PK(a)), (\mathtt{1}, a)))\}_{SK(s)}) \\
6. & b \to a & : \; (\mathtt{6}, \{(\mathtt{3}, ((\mathtt{0}, na), \\
 & & \qquad (\mathtt{3}, ((\mathtt{0}, nb), (\mathtt{1}, a)))))\}_{PK(a)}) \\
7. & a \to b & : \; (\mathtt{7}, \{(\mathtt{0}, nb)\}_{PK(b)}).
\end{array}
$$

As discussed earlier, we would still need to add information to the pair tag giving the lengths of the components, and we would need to add redundancy to the body of encryptions so as to implement the perfect encryption assumption.

It is important to realize that the results presented here do not depend for their validity on the exact type structure used in examples in this paper.

The central theorem guarantees that using the tagging scheme will prevent attacks that rely on type confusion between two types *that have distinct tags*. Different tagging schemes distinguish different types, resulting in different guarantees about what type flaw attacks will be prevented. So, for instance, if we used a tagging scheme of

$$Tag \; ::= \; \mathsf{atom} \mid \mathsf{pair} \mid \mathsf{enc}\; Tag^* \; Tag$$

then we would prevent all attacks involving passing off atoms as encryptions or pairs, but not attacks where a nonce is used in place of an agent's identity.

Furthermore, since the penetrator can always manipulate top-level tags and tags that are not protected by an encryption, such tags provide no useful guarantees and can be safely omitted. Thus in practice protocol implementations need only tag messages within encryptions, resulting in even less of a tagging overhead.

We believe that the tagging scheme could be simplified further, by combining the tags inside each encrypted component into a single *component number*; we believe this would be reasonably straightforward to prove formally. In fact, using component numbers in this way provides more protection than simply tagging fields with their types, because it would prevent the penetrator replaying one component in the place of another with the same type, for example, replaying the encrypted component from message 3 of the Woo and Lam Protocol as a message 5. The advantages of not allowing one component to be replayed in the place of another are well understood; see, for example, Principle 10 of [1].

Note that including a *message* number within each encrypted component is not enough: different components within the same message require different component numbers. For example, consider the following protocol, which aims to authenticate b to a and to establish na as a secret shared between a and b:

$$\begin{array}{ll} \text{Msg 1.} & a \to b \ : \ \{a, c\}_{PK(b)}, \{a, na\}_{PK(b)} \\ \text{Msg 2.} & b \to a \ : \ c, \{na\}_{PK(a)}. \end{array}$$

This protocol is correct under the strong typing assumption, and hence under the tagging scheme proposed in this paper. However, if each encrypted message is simply tagged with the message number in which it appears then the following attack exposes N_a:

$$\begin{array}{llll} \text{Msg } \alpha.1. & A \to B & : & \{1, A, C\}_{PK(B)}, \\ & & & \{1, A, N_a\}_{PK(B)} \\ \text{Msg } \alpha.2. & B \to A & : & C, \{2, N_a\}_{PK(A)} \\ \text{Msg } \beta.1. & I(A) \to B & : & \{1, A, N_a\}_{PK(B)}, \\ & & & \{1, A, N_a'\}_{PK(B)} \\ \text{Msg } \beta.2. & B \to I(A) & : & N_a, \{2, N_a'\}_{PK(A)}. \end{array}$$

The attack exploits the fact that the two parts of the first message have identical tags but different types. It can be prevented by replacing the message numbers with component numbers.

We anticipate that our work will be of much use to protocol analysers using model checkers, who often use techniques that will not detect type flaw attacks. In such situations, it is a simple matter to decide which tagging scheme to use: the tagging scheme used in the implementation of the protocol should match the typing scheme in the model. In this way, the model checker tests for attacks that do not rely on type flaws, and the tagging scheme guarantees that the result can be extended to cover all type confusion attacks not considered by the model checking.

Acknowledgements

We would like to thank Joshua Guttman and the anonymous referees for their useful comments on this paper.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996. Also in Proceedings of the 1994 IEEE Symposium on Security and Privacy, and Digital Equipment Corporation Systems Research Center, Research Report 125.

[2] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[3] W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[4] C. A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS '96, LNCS 1146*, pages 351–364, 1996.

[5] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[6] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[7] S. A. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Enginnering*, September 1998.

[8] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Mixed strand spaces. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[9] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2, 3):191–230, 1999.

[10] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.