

# On Preventing Type Flaw Attacks on Security Protocols With a Simplified Tagging Scheme

YAFEN LI, WUU YANG and CHING-WEI HUANG

National Chiao-Tung University, Hsin-Chu, Taiwan, R.O.C.

## Abstract

A *type flaw attack* on a security protocol is an attack where a field in a message that was originally intended to have one type is subsequently interpreted as having another type. Heather et al. proves that type flaw attacks can be prevented with the technique of tagging each field with the information that indicates its intended type. We simplify Heather et al.'s tagging scheme by combining all the tags inside each encrypted component into a single tag and by omitting the tags on the outmost level. The simplification reduces the sizes of messages in the security protocol. We also formally prove our simplified tagging scheme is as secure as Heather et al.'s with the strand space method.<sup>1</sup>

**KEY WORDS:** network security, security protocol, type flaw, strand space, tagging

## 1 Introduction

A *type flaw attack* on a security protocol is an attack where a field of a message that was originally intended to have one type is subsequently interpreted as

---

<sup>1</sup>The work reported here is supported by National Science Council, Taiwan, R.O.C., under grant NSC 92-2213-E-009-070. A short summary of this paper was presented at the 4th International Symposium on Information and Communication Technologies, Las Vegas, Nevada, June 16-18, 2004.

having another type [3]. For example, consider the Neuman-Stubblebine protocol[7].

**Initial exchange**

**Msg 1.**  $A \rightarrow B : A, N_a$

**Msg 2.**  $B \rightarrow S : B, \{A, N_a, T_b\}_{Shared(B,S)}, N_b$

**Msg 3.**  $S \rightarrow A : \{B, N_a, K_{ab}, T_b\}_{Shared(A,S)}, \{A, K_{ab}, T_b\}_{Shared(B,S)}, N_b$

**Msg 4.**  $A \rightarrow B : \{A, K_{ab}, T_b\}_{Shared(B,S)}, \{N_b\}_{k_{ab}}$

**Subsequent authentication**

**Msg 5.**  $A \rightarrow B : N'_a, \{A, K_{ab}, T_b\}_{Shared(B,S)}$

**Msg 6.**  $B \rightarrow A : N'_b, \{N'_a\}_{K_{ab}}$

**Msg 7.**  $A \rightarrow B : \{N'_b\}_{K_{ab}}$

In [1], Carlsen describes a type flaw attack on the Neuman-Stubblebine protocol. The attack is shown below, where  $P_x$  denotes the penetrator which masquerades as the principal  $x$ .

**Msg 1.**  $P_a \rightarrow B : A, N_P$

**Msg 2.**  $B \rightarrow P_s : B, \{A, N_P, T_b\}_{Shared(B,S)}, N_b$

**Msg 4.**  $P_a \rightarrow B : \{A, N_P, T_b\}_{Shared(B,S)}, \{N_b\}_{N_P}$

In the attack, the two penetrators  $P_a$  and  $P_s$  (which could possibly be the same attacker) collaborate to cheat  $B$ .  $B$  will decode message 4 with the secret key  $Shared(B, S)$  shared by  $B$  and the real server  $S$  to obtain  $N_P$ , which  $B$  is fooled to believe to be the secret session key between  $B$  and  $A$ . Once the protocol for the initial exchange is compromised, subsequent authentication can be attacked in a trivial manner.

Heather et al.[3] proves that type flaw attacks can be prevented with the technique of tagging each field with the information that indicates its intended type. The Neuman-Stubblebine protocol with Heather et al.'s tags is shown below. We follow the notation in [3]. A pair of curly brackets with a suitable superscript

and/or a subscript means encryption. Items in the same components are separated with commas.

**Initial exchange**

**Msg 1.**  $A \rightarrow B : (agent, A), (nonce, N_a)$

**Msg 2.**  $B \rightarrow S : (agent, B), (\{|nonce, timestamp|\}^{shared},$   
 $\{(nonce, N_a), (timestamp, T_b)\}_{Shared(B,S)}, (nonce, N_b)$

**Msg 3.**  $S \rightarrow A : (\{|agent, nonce, shared, timestamp|\}^{shared},$   
 $\{(agent, B), (nonce, N_a), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(A,S)},$   
 $(\{|agent, shared, timestamp|\}^{shared},$   
 $\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}, (nonce, N_b)$

**Msg 4.**  $A \rightarrow B : (\{|(agent, shared, timestamp)|\}^{shared},$   
 $\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)},$   
 $(\{|nonce|\}^{shared}, \{(nonce, N_b)\}_{k_{ab}}^{shared})$

**Subsequent authentication**

**Msg 5.**  $A \rightarrow B : (nonce, N'_a), (\{|(agent, shared, timestamp)|\}^{shared},$   
 $\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}_{Shared(B,S)}$

**Msg 6.**  $B \rightarrow A : (nonce, N'_b), (\{|nonce|\}^{shared}, \{(nonce, N'_a)\}_{K_{ab}}^{shared})$

**Msg 7.**  $A \rightarrow B : (\{|nonce|\}^{shared}, \{(nonce, N'_b)\}_{K_{ab}}^{shared})$

In this paper, we simplify the tagging scheme by combining all the tags inside each encrypted component into a single tag and by omitting the tags on the outmost level. The Neuman-Stubblebine protocol with our simplified tags is shown below:

**Initial exchange**

**Msg 1.**  $A \rightarrow B : A, N_a$

**Msg 2.**  $B \rightarrow S : B, \{(agent, nonce, timestamp), (A, N_a, T_b)\}_{Shared(B,S)}, N_b$

**Msg 3.**  $S \rightarrow A :$

$\{(agent, nonce, shared, timestamp), (B, N_a, K_{ab}, T_b)\}_{Shared(A,S)},$

$\{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}, N_b$

**Msg 4.**  $A \rightarrow B : \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)},$

$\{(nonce, N_b)\}_{k_{ab}}^{shared}$

**Subsequent authentication**

**Msg 5.**  $A \rightarrow B : N'_a, \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}^{shared}$

**Msg 6.**  $B \rightarrow A : N'_b, \{(nonce, N'_a)\}_{K_{ab}}^{shared}$

**Msg 7.**  $A \rightarrow B : \{(nonce, N'_b)\}_{K_{ab}}^{shared}$

Consider the fourth message in the protocol. Note that, in our simplified tagging scheme, the outmost-level tags—  $\{|agent, key, timestamp|\}^{shared}$  and  $\{|nonce|\}^{shared}$ —are omitted. Furthermore, the three tags inside the encrypted component—*agent*, *key* and *timestamp*—are combined into a single tag —  $(agent, key, timestamp)$ . Because in a security protocol, the kinds of all possible tags are very limited, the combined tag can be represented with a single, small integer.

Following Heather-Lowe-Schneider’s proof method [3] (which will be referred to as the *HLS-scheme* in this paper), we also proved that our simplified tagging scheme is as secure as the HLS-scheme. The proofs proceed in two stages. In the first stage, we first define an *A-scheme* in which all tags inside each encrypted component are combined into a single tag. We show that the A-scheme is as secure as the HLS-scheme. In the second stage, the A-scheme is further simplified. We define a *B-scheme* in which the outmost-level tags are omitted in addition to the simplifications made in the A-scheme. We then show that the B-scheme is as secure as the A-scheme. We then conclude that the B-scheme, our simplified tagging scheme, is as secure as Heather et al.’s (full) tagging scheme.

Since we adopt the same model and proof techniques of [2, 3], the definitions in this paper (in Sections 3, 4, and 5) are adapted from the above references, with necessary modifications to reflect our simplified tagging scheme. We provide these

definitions in order to facilitate the reader to understand our proofs given later.

Attacks based on type flaws are quite common in security protocols. Meadows [5] also discusses a similar type flaw attack on the Needham-Schroder protocol [6]. The Woo-Lam protocol  $\pi_1$  [9] is also subject to a type flaw attack [3].

The remainder of this paper is organized as follows: Section 2 defines the strand space model [2]. Section 3 defines the A-scheme and shows that the A-scheme is as secure as the HLS-scheme. In Section 4, we define the B-scheme and show that the B-scheme is as secure as the A-scheme. Section 5 concludes this paper.

## 2 Background

### 2.1 Strand Spaces

Our proof method is based on *strand spaces* [2]. We will briefly review strand spaces and its notations. In this section,  $A$  denotes the set of all possible messages that can be sent or received by principals in a protocol.  $T$  denotes the set of atomic messages.  $K$  denotes the set of keys. The elements of  $A$  are called *terms* (or *facts*). There are a unary operator and two binary operators defined on  $A$ :

- A unary operator  $inv: K \rightarrow K$ .  $inv$  maps a member of a key pair to the other for an asymmetric key and it maps a symmetric key to itself.
- Two binary operators for encryption and joining, respectively:

$$encr : K \times A \rightarrow A$$

$$join : A \times A \rightarrow A$$

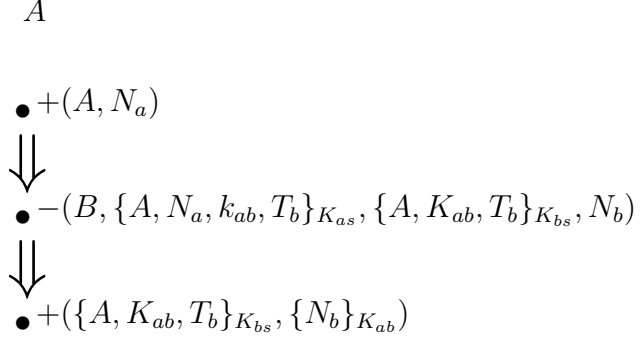


Figure 1: A strand.

We will write  $inv(K)$  as  $K^{-1}$ ,  $encr(K, m)$  as  $\{m\}_K$ , and  $join(a, b)$  as  $ab$ . We will refer to the set of ciphertexts of the form  $\{h\}_k$  as  $E$  and the set of terms of the form  $ab$  as  $C$ . A term (fact)  $f$  is simple if  $f \in T \cup K \cup E$ .

A *strand* represents a sequence of events that a principle may be engaged in. That is, each strand is a sequence of message transmissions and receptions. Formally, it has the form  $\langle \pm a_1, \pm a_2, \dots, \pm a_n \rangle$ , where  $+a$  represents the transmission of message  $a$  and  $-a$  represents the reception of message  $a$ . An element of the strand is called a *node*.

A graph structure is defined on strands with two types of edges: Figure 1 is a sample strand.

- If node  $n_i$  and  $n_{i+1}$  are consecutive steps on the same strand then we write  $n_i \Rightarrow n_{i+1}$ . This represents the causal relation between  $n_i$  and  $n_{i+1}$ .
- If nodes  $n_i = +a$  and  $n_j = -a$  then we write  $n_i \rightarrow n_j$ . This means that node  $n_i$  sends the message which is received by node  $n_j$ .

A *bundle* is a finite subgraph of this graph. It consists of a number of strands—legitimate or otherwise—hooked together where one strand sends a message and

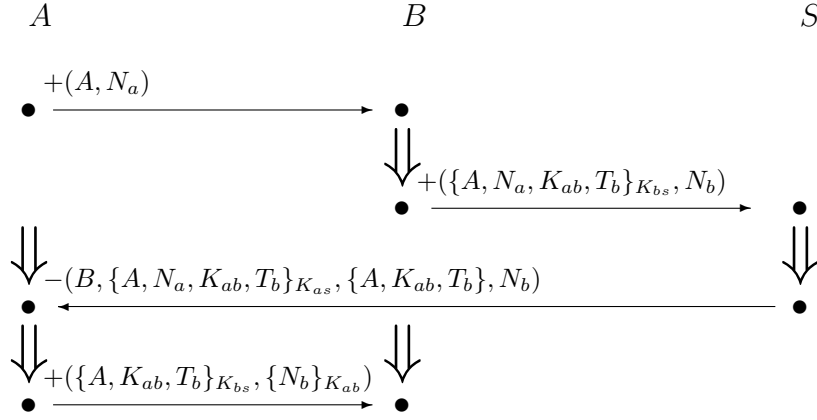


Figure 2: A bundle.

another strand receives the message. Figure 2 is a sample bundle that involves three strands. Formally, let  $C$  be a set of edges and let  $N_c$  be the set of nodes incident with any edge in  $C$ .  $C$  is a bundle if:

1.  $C$  is finite.
2. If  $n_1 \in N_c$  and  $n_1$  has a negative sign, then there is a unique  $n_2$  such that the edge  $n_2 \rightarrow n_1 \in C$ .
3. If  $n_1 \in N_c$  and  $n_2 \Rightarrow n_1$  then the edge  $n_2 \Rightarrow n_1 \in C$ .
4.  $C$  is acyclic.

We will speak of a node as being in the bundle  $C$  if in fact it is in  $N_c$ .

## 2.2 Honest Agents

### 2.2.1 Strand Templates

As in [3], we assume that each role in the protocol is defined by a strand template.

A strand template is defined as follows.

$$\begin{aligned}
 \textit{StrandTemplate} & ::= (\textit{Sign} \times \textit{Template})^* \\
 \textit{Sign} & ::= + \mid - \\
 \textit{Template} & ::= \textit{Var} \mid \textit{Fn}(\textit{Var}^*) \mid \{\textit{Template}^*\}_{\textit{Template}}^{\textit{Tag}}
 \end{aligned}$$

As in [3], the template  $g(v_1 \dots v_n)$  represents the function  $g$  applied to variables  $v_1 \dots v_n$ . It is only defined when they are applied to arguments of the correct types. The template  $t_k^{\textit{tag}}$  represents template  $t$  encrypted using key  $k$  and an encryption algorithm corresponding to  $\textit{tag}$ . For example, the roles  $A, B, S$  in the Neuman-Stubblebine protocol[7] would be defined by the following three strand templates:

$$\begin{aligned}
 \textit{temp}_a & \cong \langle +(a, n_a), -(\{b, n_a, k_{ab}, t_b\}_{\textit{Shared}(a,s)}, x, n_b), +(x, \{n_b\}_{k_{ab}}) \rangle \\
 \textit{temp}_b & \cong \langle -(a, n_a), +(b, \{a, n_a, t_b\}_{\textit{Shared}(b,s)}, n_b), \\
 & \quad -(\{a, k_{ab}, t_b\}_{\textit{Shared}(b,s)}, \{n_b\}_{k_{ab}}) \rangle \\
 \textit{temp}_s & \cong \langle -(b, \{a, n_a, t_b\}_{\textit{Shared}(b,s)}, n_b), \\
 & \quad +(\{b, n_a, k_{ab}, t_b\}_{\textit{Shared}(a,s)}, \{a, k_{ab}, t_b\}_{\textit{Shared}(b,s)}, n_b) \rangle
 \end{aligned}$$

All strands representing an execution of a particular role can be obtained by instantiating the free variables of the corresponding strand template.

**Definition.** An *honest strand* is one that results from the application of a substitution to a strand template.

Formally, a *substitution* is a function that maps a variable to a simple fact. Such a substitution function can be lifted to the templates and the strand tem-

plates:

$$sub : var \longrightarrow simple\ fact$$

$$subt : template \longrightarrow fact$$

$$subs : strand\ template \longrightarrow honest\ strand$$

which are defined as

$$subt(v) = sub(v), \forall v \in Var$$

$$subt(g(v_1, \dots, v_n)) = (g(sub(v_1), \dots, sub(v_n))), \forall g \in Fn$$

$$subt(\{t\}_k^{tk}) = (\{subt(t)\}_{subt(k)}^{tk})$$

$$subs((s_1, t_1), \dots, (s_k, t_k)) = ((s_1, subt(t_1)), \dots, (s_k, subt(t_k)))$$

## 2.3 Penetrators

Two ingredients characterize the penetrator's capabilities: a set of keys known initially to the penetrator and a set of penetrator traces that allow the penetrator to generate new messages from the messages he intercepts.

The atomic actions available to the penetrator (i.e., a penetrator trace) include the **M** (text message), **F** (flushing), **T** (tee), **C** (concatenation), **S** (separation), **K** (key), **E** (encryption), and **D** (decryption) strands. According to [1, 7], it is possible to extend the set of penetrator traces given here to model other abilities of the penetrator.

**Definition.** Each *penetrator strand* is one that results from the application of a substitution to a penetrator trace.

## 2.4 Security Properties

As in [3], failures of a security protocol include the failures of secrecy and authentication.

There is a breach of secrecy if there is a strand  $s$  in which the value of a particular variable  $v$  (which is intended to remain secret) becomes known to the penetrator, even if the secret keys have not been compromised. In this case, the strand  $s_1$  represents a penetrator strand. There is a breach of authentication if there is a penetrator strand  $s_1$  without a corresponding honest strand  $s_2$ , even if the secret keys have not been compromised. Here,  $s_1$  and  $s_2$  are related by two substitutions  $sub1$  and  $sub2$  that agree on some set of variables  $X$ . In this case, the strand  $s_1$  represents a penetrator strand. Note that if  $s_1$  is an honest strand, there must be a corresponding honest strand  $s_2$ .

## 3 The A-Scheme

### 3.1 Defining the A-Scheme

In this subsection, we present the model that will be used to prove the first point—a protocol under our simplified tagging scheme (but with the outmost-level tags) is as secure as that under Heather et al.’s (full) tagging scheme. That is, if there is an attack upon a protocol under the A-scheme, then there must exist a corresponding attack under the HLS-scheme. This model is based on the strand space model of [2, 3, 7].

Our proof proceeds as follows. First we model the abilities of a penetrator with the *penetrator strands* in the A-scheme. Then we show that every penetrator strand in the A-scheme corresponds to a penetrator strand in the HLS-scheme. Hence, the A-scheme would not introduce any new security attacks that are not

present in the HLS-scheme.

### 3.1.1 Tags and Facts

**Tags** As in [3], we assume that atomic values are partitioned into types, including agent, nonce, time-stamp, public key, etc., and we will adopt the obvious names for each tag. The types of tags can be defined by:

$$Tag ::= agent \mid nonce \mid timestamp \mid public \mid \dots \mid \{ | Tag^* | \}^{Tag}$$

We assume that the tag for an encrypted item 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. We also include the type of the body within the encryption tag.

**Tagged Facts** We represent the tagged facts as  $(Tags, Facts)$  pairs, where the tags give the claimed types of the corresponding facts.

$$Fact ::= Atom \mid \{ | TaggedFact | \}_{Fact}^{Tag}$$

$$TaggedFact ::= Tag^* \times Fact^*$$

A tagged fact  $tf$  is *simple* if  $tf = (t, f)$ , where  $t \in Tag$  and  $f \in Fact$ . We also adopt the perfect encryption assumption<sup>2</sup>, that is, that an honest agent can tell whether it has correctly decrypted a message. We will use the notations  $ts$  and  $fs$  to represent a set of tags and a set of facts, respectively. We also use the pair  $(ts, fs)$  to represent a set of ordered pairs of simple

---

<sup>2</sup>This can be implemented by including sufficient redundancy within the encryption.

tagged facts  $(t_1, f_1), (t_2, f_2), \dots, (t_k, f_k)$  (here  $ts$  and  $fs$  must have exactly the same number of elements). We also use  $tf$  to represent a tagged fact. We will often want to talk about the tag or fact components of a tagged fact, so we define projection functions as follows:

$$(ts, fs)_1 = ts$$

$$(ts, fs)_2 = fs$$

**Sub-tagged-fact relation** The sub-tagged-fact relation  $\sqsubset$  is defined inductively as follows.

- $(t, f) \sqsubset (t, f)$  (reflexive)
- $(t, f) \sqsubset (ts, fs)$  if  $(t, f) \sqsubset (t_i, f_i)$ , for some  $(t_i, f_i)$  in  $(ts, fs)$ .
- $(ts, fs) \sqsubset (ts', fs')$  if  $(t_i, f_i) \sqsubset (t'_i, f'_i)$  for every  $(t_i, f_i)$  in  $(ts, fs)$ .
- $(ts, fs) \sqsubset (\{|ts|\}^{tk}, \{(ts', fs')\}_k^{tk})$  if  $(ts, fs) \sqsubset (ts', fs')$

**Sub-fact relation** The sub-fact relation  $\sqsubset$  is defined in terms of the *Sub-tagged-fact relation*, as follows.

- $f \sqsubset (ts, fs)$  if  $\exists t \in Tag, (t, f) \sqsubset (ts, fs)$ .
- $fs \sqsubset (ts', fs')$  if  $f_i \sqsubset (t'_i, f'_i)$ , for every  $f_i$  in  $fs$ .

**Correct Tagging** We now define what it means for a tagged fact to be *correctly tagged*.

- $WellTagged(agent, x) \Leftrightarrow x \in Agent$
- $WellTagged(nonce, x) \Leftrightarrow x \in Nonce$
- $WellTagged(timestamp, x) \Leftrightarrow x \in Timestamp$

- $WellTagged(shared, x) \Leftrightarrow x \in Sharedkey$
- $WellTagged(public, x) \Leftrightarrow x \in Publickey$
- $WellTagged(private, x) \Leftrightarrow x \in Privatekey$
- $WellTagged(ts, fs) \Leftrightarrow WellTagged(t_i, f_i)$ , for every pair  $(t_i, f_i)$  in  $(ts, fs)$ .
- $WellTagged(\{|ts|\}^{tk}, x) \Leftrightarrow \exists (ts, fs) \in TaggedFact, \exists k \in Fact, x = \{(ts, fs)\}_k^{tk} \wedge WellTagged(ts, fs) \wedge WellTagged(tk, k)$ .

**Top-Level Correct Tagging** We now define what it means for a tagged fact to be *correctly tagged at the outmost level*.

- $TopLevelWellTagged(agent, x) \Leftrightarrow x \in Agent$
- $TopLevelWellTagged(nonce, x) \Leftrightarrow x \in Nonce$
- $TopLevelWellTagged(timestamp, x) \Leftrightarrow x \in Timestamp$
- $TopLevelWellTagged(shared, x) \Leftrightarrow x \in Sharedkey$
- $TopLevelWellTagged(public, x) \Leftrightarrow x \in Publickey$
- $TopLevelWellTagged(private, x) \Leftrightarrow x \in Privatekey$
- $TopLevelWellTagged(ts, fs) \Leftrightarrow TopLevelWellTagged(t_i, f_i)$ , for every pair  $(t_i, f_i)$  in  $(ts, fs)$ .
- $TopLevelWellTagged(\{|ts|\}^{tk}, x) \Leftrightarrow \exists (ts, fs) \in TaggedFact, \exists k \in Fact, x = \{(ts, fs)\}_k^{tk}$ .

Note that in  $TopLevelWellTagged(\{|ts|\}^{tk}, x)$ , we intentionally leave out the two requirements  $WellTagged(ts, fs)$  and  $WellTagged(tk, k)$ .

### 3.1.2 Origination

We will want to talk about the *origination* of a fact or tagged fact. If  $S$  is a set of tagged facts, the term of node  $n$  is  $+tf$  for some  $tf \in S$ , and for each node  $n'$  previous to  $n$ , the term of  $n'$  is not in  $S$ , then  $n$  is an *entry point* to  $S$ . The node  $n$  is the *origination* of a tagged fact  $tf$ , if  $n$  is the entry point of the set of tagged facts  $\{tf' \mid tf \sqsubset tf'\}$ . Similarly, the node  $n$  is the *origination* of a fact  $f$ , if  $n$  is the entry point of  $\{tf' \mid f \sqsubset tf'\}$ . A fact or tagged fact is *uniquely originating in a bundle  $C$*  if it originates on a unique node of  $C$ .

### 3.1.3 Strand Templates

As in [3], we use the strand templates to define roles in a protocol. A strand template is defined as follows:

$$\begin{aligned}
 \textit{StrandTemplate} & ::= (\textit{Sign} \times \textit{TaggedTemplate})^* \\
 \textit{Sign} & ::= + \mid - \\
 \textit{TaggedTemplate} & ::= \textit{Tag}^* \times \textit{Template}^* \\
 \textit{Template} & ::= \textit{Var} \mid \textit{Fn}(\textit{Var}^*) \mid \{\textit{TaggedTemplate}\}_{\textit{Template}}^{\textit{Tag}}
 \end{aligned}$$

As in [3], the template  $g(v_1, \dots, v_n)$  represents the function  $g$  applied to variables  $v_1, \dots, v_n$ . It is only defined when they are applied to arguments of the correct types. The template  $\{t1\}_{t2}^{t3}$  represents template  $t1$  encrypted with key  $t2$  and algorithm  $t3$ . A tagged template  $tt$  is *simple* if  $tt = (t, t')$ , where  $t \in \textit{Tag}$  and  $t' \in \textit{Template}$ .

For example, the role  $A$  in the Neuman-Stubblebine protocol [7] would be defined by the following strand template:

$$\begin{aligned}
 & \textit{temp}_a \cong \\
 & < +(\textit{agent}, \textit{nonce}), (a, n_a),
 \end{aligned}$$

$$\begin{aligned}
& - ( \{ \{ agent, nonce, key, timestamp \} \}^{shared}, \\
& \quad \{ \{ agent, key, timestamp \} \}^{shared}, nonce ) \\
& \quad ( \{ ( agent, nonce, key, timestamp ), ( b, n_a, k_{ab}, t_b ) \}^{shared}_{Shared(a,s)}, \\
& \quad \{ ( agent, key, timestamp ), ( a, k_{ab}, t_b ) \}^{shared}_{Shared(b,s)}, n_b ), \\
& + ( \{ \{ agent, key, timestamp \} \}^{shared}, \{ \{ nonce \} \}^{shared} ), \\
& \quad ( \{ ( agent, key, timestamp ), ( a, k_{ab}, t_b ) \}^{shared}_{Shared(b,s)}, \\
& \quad \{ ( nonce, n_b ) \}^{shared}_{k_{ab}} ) >
\end{aligned}$$

**Definition.** An *honest strand* is one that results from the application of a substitution to a strand template.

All strands representing an execution of a particular role can be formed by instantiating the free variables of the corresponding strand template. Formally, a substitution is a function mapping a variable to a simple fact. A substitution can be lifted to the templates, the tagged templates, and the strand templates.

$$\begin{aligned}
sub & : var \longrightarrow simple\ fact \\
subt & : template \longrightarrow fact \\
subtt & : tagged\ template \longrightarrow tagged\ fact \\
subs & : strand\ template \longrightarrow honest\ strand
\end{aligned}$$

which are defined as

$$\begin{aligned}
subt(v) & = sub(v) \\
subt(g(v_1, \dots, v_n)) & = (g(sub(v_1), \dots, sub(v_n))), \forall g \in Fn \\
subt(\{tt\}_k^{tk}) & = (\{subtt(tt)\}_{subt(k)}^{tk}) \\
subtt(tt) & = (t_1, \dots, t_k, subt(t'_1), \dots, subt(t'_k))
\end{aligned}$$

where  $tt = (t_1, \dots, t_k, t'_1, \dots, t'_k)$  and each  $(t_i, t'_i)$  is a simple tagged template of  $tt$ .

$$\text{subs}(s) = ((s_1, \text{subtt}(tt_1)), \dots, (s_k, \text{subtt}(tt_k)))$$

where  $s = ((s_1, tt_1), \dots, (s_k, tt_k))$  and

each  $s_i \in \{+, -\}$ .

We also assume that each strand template is always consistently tagged, that is, the same tags are always given to the same variables. We also assume that simple tagged facts on honest strands are always well tagged, at least on the outmost level.

**Honest Strand Assumption.** If the simple tagged fact  $(t, f)$  originates on an honest strand, then  $\text{TopLevelWellTagged}(t, f)$ .

This assumption implies a number of facts:

- If an honest agent introduces a simple term for a variable, then he/she introduces a value of the expected type.
- An honest agent will only tag a fact as being an encryption if it is indeed created as an encryption. The encryption tag will include the identity of the algorithm used and the tags of the body. However, the agent might receive ill-tagged keys from the penetrator. Hence, the key used for the encryption might not have the expected type.

### 3.1.4 Penetrator Traces

Following [1, 7], we assume that there are some set  $T$  of simple facts that the penetrator can produce and some set  $K_p$  of keys that the penetrator has available. Penetrator traces under the tagging scheme are exactly analogous to those in [3], but with the modifications to the  $M$ ,  $C$ ,  $S$  and  $R$  traces.

A penetrator trace is one of the following  $M$ ,  $F$ ,  $T$ ,  $C$ ,  $S$ ,  $K$ ,  $E$ ,  $D$ , and  $R$  strands.

**M Text message**  $\langle +(t, f) \rangle$  for  $f \in T$ . The penetrator spontaneously generates a text message from the simple fact available to him/her. Note that the  $M$  strand only produces simple tagged facts and non-simple tagged facts can be produced by concatenation.

**F Flushing**  $\langle -(ts, fs) \rangle$ .

**T Tee**  $\langle -(ts, fs), +(ts, fs), +(ts, fs) \rangle$ .

**C Concatenation**  $\langle -(ts_1, fs_1), \dots, -(ts_k, fs_k), +(ts_1, \dots, ts_k, fs_1, \dots, fs_k) \rangle$ .

**S Separation**  $\langle -(t_1, \dots, t_k, f_1, \dots, f_k), +(t_1, f_1), \dots, +(t_k, f_k) \rangle$ .

**K Key**  $\langle +(tk, k) \rangle$  with  $WellTagged(tk, k)$  and  $k \in K_p$ .

**E Encryption**  $\langle -(tk, k), -(ts, fs), +(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) \rangle$ .

**D Decryption**  $\langle -(tk', k'), -(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}), +(ts, fs) \rangle$ , where  $tk'$  and  $k'$  are tags representing inverse key types and  $k'$  is the decryption 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  $R$  strand only applies to simple tagged facts. Non-simple tagged facts can be retagged by separation, retagging particular components, and then concatenation. All other penetrator traces do not interfere with the tags of their messages.

We can reach the following lemma.

**Lemma 1.** Every simple tagged fact  $(t, f)$  that is top-level-ill-tagged (i.e., not top-level-well-tagged) originates on an  $R$  or  $M$  strand.

## 3.2 Transforming Bundles

In this subsection, we transform arbitrary bundles into *well-tagged* bundles. We will show that, given a bundle  $C$  in the A-scheme, we can construct a corresponding

bundle  $C'$  in the HLS-scheme in which all terms are well-tagged. We proceed in two steps: (1) We define the properties the transformation  $\phi$  must satisfy. (2) We show that such a transformation  $\phi$  can always be constructed.

### 3.2.1 Defining the Transformation Function

The following definition captures the required properties of the transformation function  $\phi$ :

**Definition.** Given a bundle  $C$ , we define

$$\phi : \textit{tagged fact} \rightarrow \textit{tagged fact}$$

to be a transformation function for  $C$  if:

1.  $\phi$  preserves the outmost-level tags. That is, if  $\phi(ts, fs) = (ts', fs')$  then  $ts = ts'$ .
2.  $\phi$  returns well-tagged terms. That is,  $\text{WellTagged}(\phi(ts, fs))$ .
3.  $\phi$  is the identity function over well-tagged terms. That is, if  $\text{WellTagged}(ts, fs)$  then  $\phi(ts, fs) = (ts, fs)$ .
4.  $\phi(t_1, \dots, t_k, f_1, \dots, f_k) = (t_1, \dots, t_k, \phi(t_1, f_1)_2, \dots, \phi(t_k, f_k)_2)$
5.  $\phi(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) = (\{|ts|\}^{tk}, \{\phi(ts, fs)\}_{\phi(tk, k)_2}^{tk})$ .
6.  $\phi$  respects inverses of keys. That is, if  $k$  and  $k'$  are inverses of one another when considered as keys of types  $tk$  and  $tk'$ , respectively, then  $\phi(tk, k)$  and  $\phi(tk', k')$  are also inverses of one another when considered as keys of types  $tk$  and  $tk'$ , respectively.
7. If  $(t, f)$  is a simple tagged fact of  $C$  that is (top-level-)ill-tagged, the penetrator can always find a fact  $f'$  in  $T^3$  such that  $(t, f')$  is (top-level-)well-tagged.

---

<sup>3</sup> $T$  is the set of simple facts that the penetrator can produce.

8. When  $\phi$  is applied to a simple tagged fact  $tf$  of  $C$  that is top-level-ill-tagged, it produces a fact that is essentially new. That is,  $\forall tf \in SimpleTaggedFacts(C)$ <sup>4</sup>  $\forall f \in T [[\neg TopLevelWellTagged(tf) \wedge f \sqsubset \phi(tf)] \Rightarrow \forall tf' \in TaggedFacts(C)$ <sup>5</sup>  $[tf \not\sqsubset tf' \Rightarrow f \not\sqsubset \phi(tf')]]$

Note that the transformation function  $\phi$  is injective over the tagged facts of  $C$ . We can show that it is always possible to find the required  $\phi$ .

**Lemma 2.** Given a bundle  $C$ , there exists a transformation function  $\phi$  for  $C$ .

### 3.2.2 Transforming Honest Strands

In this subsection, we will show that if a strand  $S = subs(temp)$  obtained by instantiating a template  $temp$  with respect to the substitution  $sub$  is an honest strand, then the strand obtained by transforming each term of  $S$  with  $\phi$  is also an honest strand. Consider the strand  $S' = subs'(temp)$  with respect to the substitution  $sub'$  defined as

$$sub'(v) = \phi(t, sub(v))_2$$

where  $t$  is the unique tag for  $v$  in  $temp$ . Note that  $S'$  is also an honest strand.

**Lemma 3.** Let  $\phi$ ,  $sub$  and  $sub'$  be defined as above.  $subtt$  and  $subtt'$  are derived from  $sub$  and  $sub'$ , respectively. Then  $\phi(subtt(tt)) = subtt'(tt)$ .

**Lemma 4.** If a strand  $S = subs(temp)$  obtained by instantiating a template  $temp$  with respect to the substitution  $sub$  is an honest strand, then the strand obtained by transforming each term of  $S$  using the transformation  $\phi$  is also an honest strand.

---

<sup>4</sup> $SimpleTaggedFacts(C)$  is the set of simple tagged facts in  $C$ .

<sup>5</sup> $TaggedFacts(C)$  is the set of tagged facts in  $C$ .

### 3.2.3 Transforming Penetrator Strands

We will show that given the penetrator strands in a bundle  $C$  and a transformation  $\phi$ , we can construct corresponding penetrator strands in a bundle  $C'$  that are well-tagged. we will consider each penetrator trace in turn.

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

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

**T Tee** Let  $S = \langle -tf, +tf, +tf \rangle$ . Define  $S' = \langle -\phi(tf), +\phi(tf), +\phi(tf) \rangle$ , which is a well-tagged  $T$  strand.

**C Concatenation** Let  $S = \langle -(t_1, f_1), \dots, -(t_k, f_k), +(t_1, \dots, t_k, f_1, \dots, f_k) \rangle$ . Define  $S' = \langle -\phi(t_1, f_1), \dots, -\phi(t_k, f_k), +\phi(t_1, \dots, t_k, f_1, \dots, f_k) \rangle$ . Because  $\phi(t_1, \dots, t_k, f_1, \dots, f_k) = (\phi(t_1, f_1)_1, \dots, \phi(t_k, f_k)_1, \phi(t_1, f_1)_2, \dots, \phi(t_k, f_k)_2)$ ,  $S'$  is a well-tagged  $C$  strand.

**S Separation** Let  $S = \langle -(t_1, \dots, t_k, f_1, \dots, f_k), +(t_1, f_1), \dots, +(t_k, f_k) \rangle$ . Define  $S' = \langle +\phi(t_1, \dots, t_k, f_1, \dots, f_k), -\phi(t_1, f_1), \dots, -\phi(t_k, f_k) \rangle$ . Because  $\phi(t_1, \dots, t_k, f_1, \dots, f_k) = (\phi(t_1, f_1)_1, \dots, \phi(t_k, f_k)_1, \phi(t_1, f_1)_2, \dots, \phi(t_k, f_k)_2)$ ,  $S'$  is a well-tagged  $S$  strand.

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

**E Encryption** Let  $S = \langle -(tk, k), -(ts, fs), +(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) \rangle$ , where  $ts = GetTags(ts, fs)$ . Define  $S' = \langle -\phi(tk, k), -\phi(ts, fs), +\phi(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) \rangle$ . Because  $\phi(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) = (\{|ts|\}^{tk}, \{\phi(ts, fs)\}_{\phi(tk, k)_2}^{tk})$ ,  $S'$  is a well-tagged  $E$  strand.

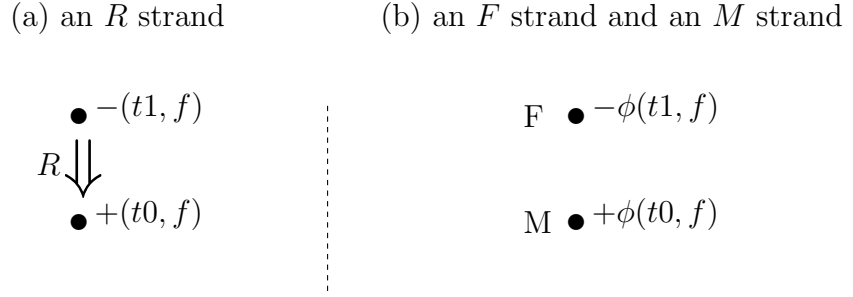


Figure 3: If  $\neg \text{TopLevelWellTagged}(t_0, f)$ , replace the  $R$  strand in (a) with an  $F$  and an  $M$  strands in (b).

**D Decryption** Let  $S = \langle -(tk', k'), -(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}), +(ts, fs) \rangle$ , where  $k$  and  $k'$ , which have types  $tk$  and  $tk'$ , respectively, are inverses of each other. Define  $S' = \langle -\phi(tk', k'), -\phi(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}), +\phi(ts, fs) \rangle$ . Because  $\phi(\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}) = (\{|ts|\}^{tk}, \{\phi(ts, fs)\}_{\phi(tk, k)_2}^{tk})$ ,  $S'$  is a well-tagged  $D$  strand.

**R Retagging** Let  $S = \langle -(t_1, f), +(t_0, f) \rangle$ . We proceed in two stages in this case. First,  $-\phi(t_1, f)$  is a well-tagged  $F$  strand. Second, if  $\neg \text{TopLevelWellTagged}(t_0, f)$ , then  $+\phi(t_0, f)$  is a well-tagged  $M$  strand. We will replace the  $R$  strand in  $S$  with an  $F$  and an  $M$  strands in  $S'$ . This situation is illustrated in Figure 3.

On the other hand, if  $\text{TopLevelWellTagged}(t_0, f)$ , we will show that an earlier  $R$  strand has the initial node labelled with  $-(t_0, f)$  or an earlier  $M$  strand produces the tagged fact  $+(t, f)$  for some  $t \in \text{Tag}$  in  $C$ .

If  $t_1 = t_0$  we are done. Here we have a well-tagged  $F$  and a well-tagged  $M$  strands. Otherwise,  $\neg \text{TopLevelWellTagged}(t_1, f)$ . According to Lemma 1, top-level-ill-tagged simple tagged fact must originate on an  $M$  or  $R$  strand. If the top-level-ill-tagged simple tagged fact  $(t_1, f)$  originates on an  $M$  strand,

then we are done. Otherwise, the simple tagged fact  $(t_1, f)$  originates on a  $R$  strand. Then let the  $R$  strand be  $\langle -(t_2, f) + (t_1, f) \rangle$ .

If  $t_2 = t_0$ , we are done. If not,  $(t_2, f)$  originates on another  $M$  or  $R$  strand. We may repeat the above argument for  $(t_2, f)$ .

Continuing in this way, we can find a sequence of earlier and earlier  $R$  strands or a sequence of earlier and earlier  $R$  strands and an  $M$  strand. Because the bundle is finite, this argument eventually stops. Hence, we may construct the corresponding penetrator strands as follows:

1. Remove nodes  $n_0'$  and  $nk$ .
2. Each of the negative nodes on the  $R$  strands is replaced by an  $F$  strand.
3. Other nodes on the  $R$  strands are replaced by an  $M$  strand and some  $T$  strands if the terms of nodes are the same. Suppose the term of nodes  $n'_{k-1}$ ,  $n'_2$ , and  $n'_1$  are the same. We have the situation in Figure 4.
4. If  $t_k = t_0$  (that is, we can find an earlier  $R$  strand with the initial node labelled with  $-(t_0, f)$  or an earlier  $M$  strand with node labelled with  $-(t_0, f)$ ), the  $\rightarrow$  successor of  $n'_0$  in  $C$  becomes the  $\rightarrow$  successor of  $\phi(n)$  in  $C'$ . This is shown in part (1) of Figure 4. If not (that is, we can find an earlier  $M$  strand to produce  $(t, f)$  for some  $t \in Tag$ ), we use an  $M$  strand to produce  $\phi(t_0, f) = (t_0, f)$ . The  $\rightarrow$  successor of  $n'_0$  in  $C$  becomes the  $\rightarrow$  successor of the node on the  $M$  strand in  $C'$ . This is shown in part (2) of Figure 4.

### 3.2.4 Unique Origination

We now consider the question of unique origination. We want to show that, if a fact  $f_0$  originates on an honest node in  $C'$  in the HLS-scheme, then  $f_0$  originates

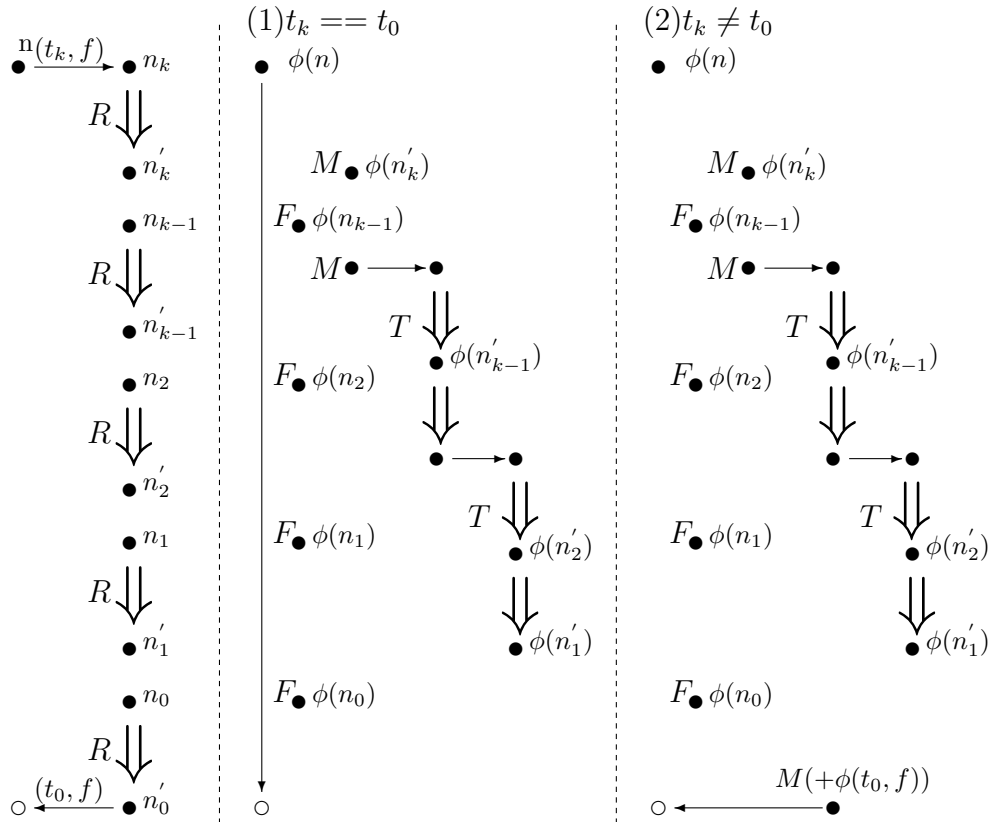


Figure 4: If  $TopLevelWellTagged(t_0, f)$ , replace the  $R$  strands with some  $F$ ,  $M$ , and  $T$  strands.

on the corresponding honest node in  $C$  in the A-scheme. On the other hand, suppose that  $f_0$  originates on a penetrator node in  $C'$ . There are four possibilities to consider:

1.  $f_0$  originates on an  $M$  strand corresponding to an occurrence of  $f_0$  on the corresponding  $M$  strand in  $C$ .
2.  $f_0$  originates on a  $K$  strand corresponding to an occurrence of  $f_0$  on the corresponding  $K$  strand in  $C$ .
3.  $f_0$  originates on an  $M$  strand corresponding to an occurrence of a top-level-ill-tagged fact  $(t, f)$  on an  $R$  strand in  $C$ , with  $f_0 \sqsubset \phi(t, f)$ .
4.  $f_0$  originates on an  $M$  strand corresponding to an occurrence of a top-level-well-tagged fact  $(t, f)$  on an  $R$  strand in  $C$ , with  $f_0 \sqsubset (t, f)$ .

The four possibilities do not raise any problems, for if the term originates multiple times in  $C'$ ,  $f_0$  originates multiple times in  $C$ .

The above result is summarized in the following theorem.

**Theorem 5.** If  $C$  is a bundle under the A-scheme then there exist a transformation  $\phi$  and a bundle  $C'$  in the HLS-scheme, such that

1.  $C'$  contains the tagged fact  $\phi(tf)$  for each tagged fact  $tf$  of  $C$ .
2.  $C'$  contains the corresponding honest strand for each honest strand of  $C$ .
3. If facts uniquely originate in  $C$ , then they also uniquely originate in  $C'$ .
4. All tagged facts of  $C'$  are well-tagged.

### 3.3 Secrecy and Authentication

In this subsection we prove our first result—if there is a type flaw attack on a protocol under A-scheme, there is a type flaw attack under HLS-scheme. We will discuss secrecy and authentication separately.

**Theorem (Secrecy).** Let  $temp$  be the strand template for some role in the A-scheme. Let  $(t, v)$  be a tagged variable. Let  $h$  be a positive integer. Let  $Keys$  be a set of function templates. Let  $C$  be a bundle in the A-scheme and  $C'$  be the well-tagged bundle obtained by transforming  $C$ . (Note that  $C'$  is in the HLS-scheme.) If there is a failure of secrecy in  $C$ , there will be a failure of secrecy in  $C'$  as well.

**Theorem (Authentication).** Let  $temp1$  and  $temp2$  be the templates for two roles in the A-scheme. Let  $X$  be the set of variables in the templates. Let  $h1$  and  $h2$  be two positive integers. Let  $Keys$  be a set of function templates. Let  $C$  be a bundle in the A-scheme and  $C'$  be a well-tagged bundle in the HLS-scheme obtained by transforming  $C$ . If there is a failure of authentication in  $C$ , there will be a failure of authentication in  $C'$  as well.

## 4 The B-Scheme

### 4.1 Defining the B-Scheme

In this subsection, we present the new model that will be used to prove the second point—A protocol under the simplified tagging scheme without the outmost-level tags is as secure as that under the simplified tagging scheme with the outmost-level tags and, hence, is as secure as that under Heather et al.’s (full) tagging scheme. That is, if there is an attack upon a protocol under the B-scheme, then there must exist a corresponding attack under the A-scheme (and hence the HLS-scheme).

The implication of this result is that, even without the outmost-level tags, we can still prevent all type flaw attacks. This B-scheme is based on the previous A-scheme.

#### 4.1.1 Strand Templates

The new strand template is defined as follows:

$$\begin{aligned}
\textit{StrandTemplate} & ::= (\textit{Sign} \times \textit{Template}^*)^* \\
\textit{Sign} & ::= + \mid - \\
\textit{TaggedTemplate} & ::= \textit{Tag}^* \times \textit{Template}^* \\
\textit{Template} & ::= \textit{Var} \mid \textit{Fn}(\textit{Var}^*) \mid \{\textit{TaggedTemplate}\}_{\textit{Template}}^{\textit{Tag}}
\end{aligned}$$

Note that in *StrandTemplate*, the outmost-level tags are omitted in the above definition. For example, the roles *A* in the Neuman-Stubblebine protocol [7] would be defined by the following strand template:

$$\begin{aligned}
& \textit{temp}_a \cong \\
& < +( a, n_a), \\
& -( \{(agent, nonce, key, timestamp), (b, n_a, k_{ab}, tb)\}_{\textit{Shared}(a,s)}^{\textit{shared}}, \\
& \quad \{(agent, key, timestamp), (a, k_{ab}, tb)\}_{\textit{Shared}(b,s)}^{\textit{shared}}, n_b), \\
& +( \{(agent, key, timestamp), (a, k_{ab}, tb)\}_{\textit{Shared}(b,s)}^{\textit{shared}}, \\
& \quad \{(nonce, n_b)\}_{k_{ab}}^{\textit{shared}}) >
\end{aligned}$$

The strand template in this B-scheme is almost exactly the same as that in the previous A-scheme defining the same role in the same protocol except that the outmost-level tags are omitted in this B-scheme.

**Definition.** An *honest strand* is one that results from the application of a substitution to a strand template.

All strands representing an execution of a particular role can be formed by instantiating the free variables of the corresponding strand template. Formally, a substitution is a function mapping a variable to a simple fact. A substitution can be lifted to the templates, the tagged templates, and the strand templates.

$$\begin{aligned}
sub & : var \longrightarrow simple\ fact \\
subt & : template \longrightarrow fact \\
subtt & : tagged\ template \longrightarrow tagged\ fact \\
subs & : strand\ template \longrightarrow honest\ strand
\end{aligned}$$

which are defined as

$$\begin{aligned}
subt(v) & = sub(v) \\
subt(g(v_1, \dots, v_n)) & = (g(sub(v_1), \dots, sub(v_n))), \forall g \in Fn \\
subt(\{tt\}_k^{tk}) & = (\{subtt(tt)\}_{subt(k)}^{tk}) \\
subtt(tt) & = (t_1, \dots, t_k, subt(t'_1), \dots, subt(t'_k)) \\
& \text{where } tt = (t_1, \dots, t_k, t'_1, \dots, t'_k) \text{ and each } (t_i, t'_i) \text{ is a} \\
& \text{simple tagged fact of } tt. \\
subs(s) & = ((s_1, subt(t_1)), \dots, (s_k, subt(t_k))) \\
& \text{where } s = ((s_1, t_1), \dots, (s_k, t_k)) \text{ and} \\
& \text{each } s_i \in \{+, -\}.
\end{aligned}$$

We also assume that each strand template is consistently tagged, that is, the same tags are always given to the same variables.

### 4.1.2 Penetrator Traces

Penetrator traces in the B-scheme are analogous to those in the A-scheme. We also assume that there are some set  $T$  of simple facts that the penetrator can produce and some set  $K_p$  of keys that the penetrator has available. A penetrator trace is one of the following  $M'$ ,  $F'$ ,  $T'$ ,  $C'$ ,  $S'$ ,  $K'$ ,  $E'$ , and  $D'$  strands. Note that we do not need the  $R'$  strands in this B-scheme because the outmost-level tags are omitted.

**M' Text message**  $\langle +f \rangle$  for  $f \in T$ . Note that the  $M'$  strand only produces simple facts and non-simple facts can be produced by concatenation.

**F' Flushing**  $\langle -fs \rangle$ .

**T' Tee**  $\langle -fs, +fs, +fs \rangle$ .

**C' Concatenation**  $\langle -fs_1, \dots, -fs_k, +(fs_1, \dots, fs_k) \rangle$ .

**S' Separation**  $\langle -(f_1, \dots, f_k), +f_1, \dots, +f_k \rangle$ .

**K' Key**  $\langle +k \rangle$  with some  $tk \in Tag$  such that  $WellTagged(tk, k)$  and  $k \in K_p$ .

**E' Encryption**  $\langle -k, -fs, +\{(ts, fs)\}_k^{tk} \rangle$ , where  $ts$  is the set of tags of facts in  $fs$  and  $tk$  is the interpreted key type of  $k$ .

**D' Decryption**  $\langle -k', -\{(ts, fs)\}_k^{tk}, +fs \rangle$ , where  $tk$  and  $tk'$  are tags representing inverse key types and  $k'$  is the decryption key corresponding to  $k$  when they are considered as keys of types  $tk'$  and  $tk$ , respectively.

## 4.2 Transforming Bundles

In this subsection, we will show that each bundle in the B-scheme can be transformed into a corresponding bundle in the A-scheme.

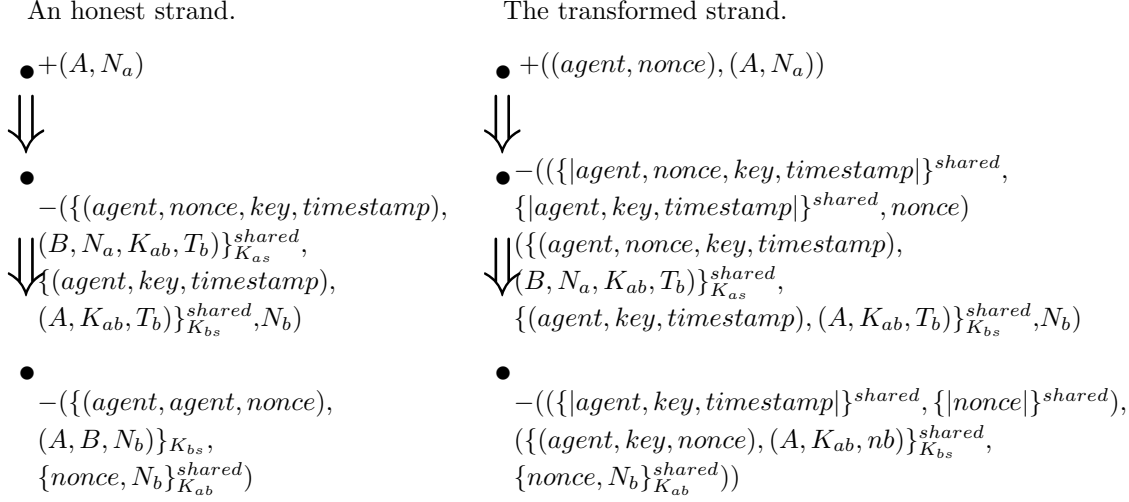


Figure 5: Transforming honest strands.

#### 4.2.1 Transforming Honest Strands

Every honest strand in the B-scheme can be transformed to an honest strand in the A-scheme by prepending the corresponding outmost-level tags of the strand template. For example, the strand produced by role  $A$  in the Neuman-Stubblebine protocol [7] is shown in Figure 5.

#### 4.2.2 Transforming Penetrator Strands

Since we know how to transform the honest strands, what we need to do is to transform the penetrator strands. Let  $C$  be a bundle in the B-scheme. Initially, let  $C'$  be the set of the honest strands obtained by transforming the honest strands of  $C$ . Let  $P = C - C'$ , which contains the penetrator strands and the  $\rightarrow$  edges of  $C$ . Let  $N_C$  and  $N_{C'}$  be the sets of nodes incident with an edge in  $C$  and  $C'$ , respectively. Let  $N_P = N_C - N_{C'}$ .

Now we want to transform the edges in  $P$ . Depending on the sources and targets of the edges as well as the signs on the nodes, we will consider each edge in

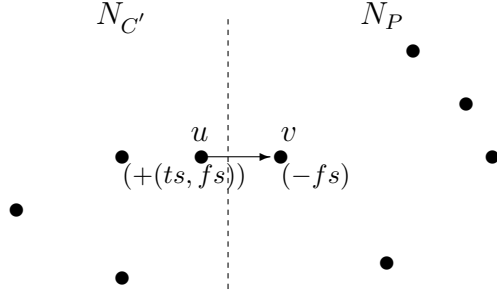


Figure 6: We prepare to replace the edge  $u \rightarrow v$ .

turn until the set  $P$  becomes an empty set. There are five cases discussed below.

**First, we consider edges from  $N_{C'}$  to  $N_P$ .**

If there exists a  $\rightarrow$  edge in  $P$  from a node  $u(+ts, fs)$  in  $N_{C'}$  to a node  $v(-fs)$  in  $N_P$  (shown in figure 6), we can replace the  $v(-fs)$  node with a  $v(-ts, fs)$  node, add a new edge  $u \rightarrow v$  to  $C'$ , and remove the original edge  $u \rightarrow v$  from  $P$ .

**Second, we consider edges from  $N_{C'}$  to  $N_{C'}$ .**

If there exists a  $\rightarrow$  edge in  $P$  from the node  $u$  to another node  $w$  in  $N_{C'}$  (shown in Figure 7), there are two cases to consider.

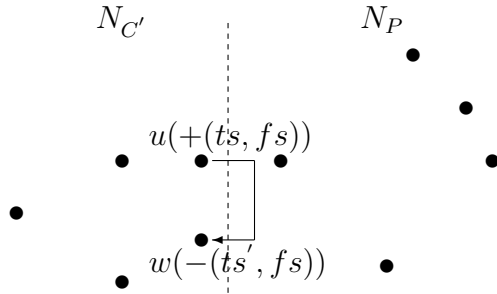


Figure 7: Create a new edge  $u \rightarrow w$ .

**case 1** If the outmost-level tags of  $u$  and  $w$  are the same, we are done.

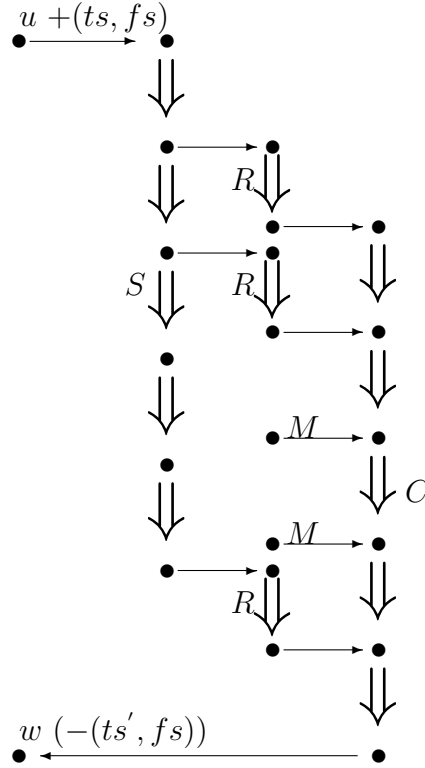


Figure 8: The case  $ts \neq ts'$ .

**case 2** Otherwise, we can use an  $S$  strand to separate  $term(u)$  into simple tagged facts. If there are sub-facts in  $term(w)$  that are not the sub-facts in  $term(u)$ , we can use some  $M$  and  $K$  strands to produce these sub-facts. We then use  $R$  strands to change the tags and use a  $C$  strand to concatenate the tagged facts. Finally, add the  $R$ ,  $M$ ,  $K$ ,  $S$  strands and the appropriate  $\rightarrow$  edges to the set  $C'$ , and remove the edge  $u \rightarrow w$  from  $P$ . This is done in figure 8.

**Third, we consider edges from  $N_{C'}$  to  $N_P$ .**

If there exists a  $\Rightarrow$  edge in  $P$  from a node  $u$  in  $N_{C'}$  to a node  $v$  in  $N_P$  (that is,  $u$  and  $v$  are in the same strand) such that  $v$  has a positive sign and the

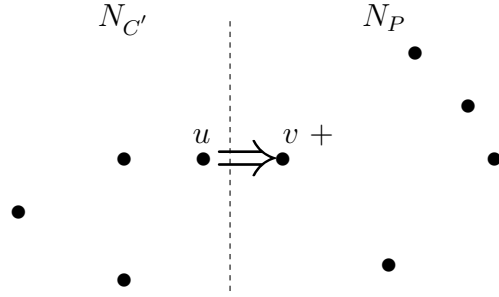


Figure 9: Finding the outmost-level tag of  $v$ .

predecessors of both  $u$  and  $v$  on the same strand has the outmost-level tags (shown in Figure 9), we can find the outmost-level tag of  $v$  as follows:

**case 1** If  $v$  belongs to a  $T'$  (tee) strand, the outmost-level tags of the node  $v$  and the node following  $v$  are the same as that of  $u$ .

**case 2** If  $v$  belongs to a  $C'$  (concatenation) strand, the outmost-level tag of  $v$  is the concatenation of the outmost-level tags of the predecessors on the same strand.

**case 3** If  $v$  belongs to an  $S'$  (separation) strand, we use some  $M$  and  $R$  strands and an  $S$  strand to substitute for the  $S'$  strand. The  $S$  strand separates the tagged fact. The  $M$  and  $K$  strands produce the facts that are terms of nodes in the  $S'$  strand but are not terms of nodes in the  $S$  strand. Finally, adjust the related edges properly.

**case 4** If  $v$  belongs to an  $E'$  (encryption) strand, we have the situation shown in figure 10.

If  $tk \neq tk'$ , we can use an  $R$  strand to retag the term  $(tk, k)$  with  $(tk', k)$  and adjust the related edges. This is done in Figure 11.

If  $ts \neq ts'$  (in Figure 10), we use an  $S$  strand to separate  $(ts, fs)$ , create some  $M$  and  $K$  strands to produce the facts that are sub-facts of  $(ts', fs)$

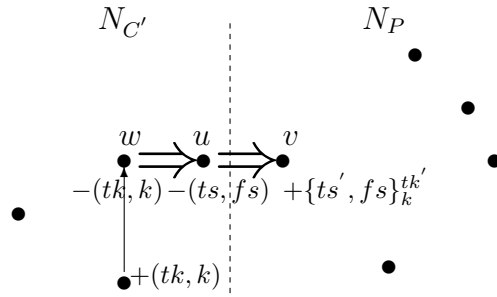


Figure 10: An  $E'$  strand.

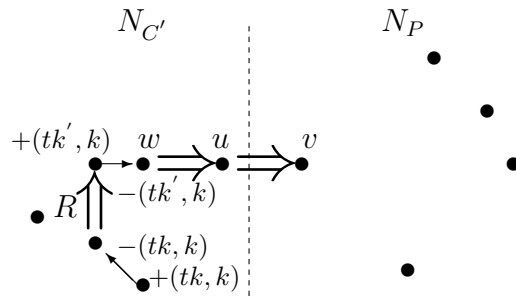


Figure 11: Introduce an  $R$  strand.



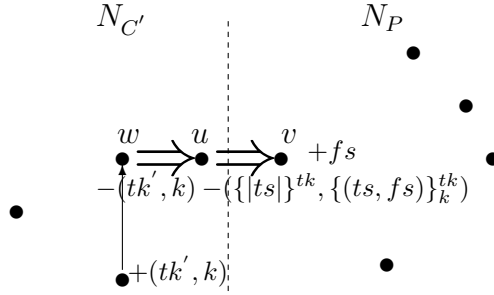


Figure 13: A  $D'$  strand.

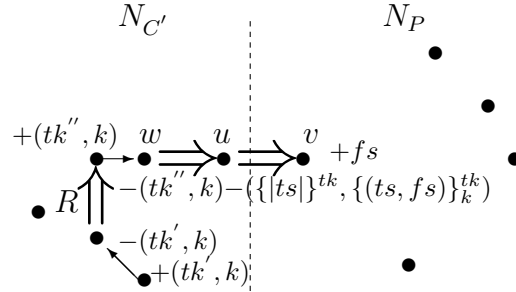


Figure 14: Use an  $R$  strand to change the tag of  $w$ .

strand to change the tag of  $w$  (in Figure 13) to  $(tk'', k)$  (shown in Figure 14). (Note that  $t''$  is chosen to be the inverse key type of  $tk$ .) The outmost-level tag of  $v$  is  $GetTags$ (the encryption body of the second node). This is done in figure 14.

Finally, replace the outmost-level tag of  $v$  with the tag we have built up and move the new edges from  $P$  to  $C'$ .

**Fourth, we consider edges from  $N_P$  to  $N_P$ .**

If there exists a  $\rightarrow$  edge in  $P$  from a node  $u(+f)$  in  $N_P$  on an  $M'$  or  $K'$  strand to a node  $v(-f)$  in  $N_P$  (as shown in figure 15), we can select an appropriate tag  $t \in Tag$ . Replace  $(+f)$  with  $(+(t, f))$ , which is an  $M$  or  $K$  strand, replace  $(-f)$  with  $(-(t, f))$ , add the edge  $u \rightarrow v$  to  $C'$ , and remove the original edge  $u \rightarrow v$  from  $P$ .

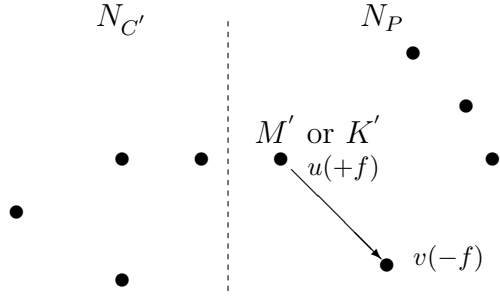


Figure 15: An  $M'$  or  $K'$  strand in  $N_P$ .

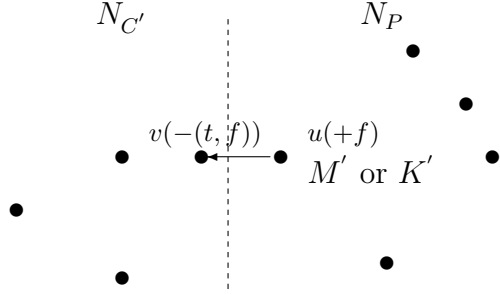


Figure 16: An  $M'$  or  $K'$  strand from  $N_P$  to  $N_{C'}$ .

**Finally, we consider edges from  $N_P$  to  $N_{C'}$ .**

If there exists a  $\rightarrow$  edge in  $P$  from a node  $u(+f)$  in  $N_P$  on an  $M'$  or  $K'$  strand to a node  $v(-(t, f))$  in  $N_{C'}$  (shown in figure 16), we will consider the following two cases.

**case 1** Suppose  $(t, f)$  is a simple fact.

**subcase 1** If node  $u$  is on an  $M'$  (message) strand, then replace  $(+f)$  with  $(+(t, f))$ , which is an  $M$  strand.

**subcase 2** If node  $u$  is on a  $K'$  (key) strand:

- If  $WellTagged(t, f)$ , replace  $(+f)$  with  $(+(t, f))$ , which is a  $K$  strand.
- If  $\neg WellTagged(t, f)$ , we can find a tag  $t' \in Tag$  such that  $WellTagged(t', f)$ .

We replace  $(+f)$  with  $(+(t', f))$ , which is a  $K$  strand, use an  $R$  strand to change  $(t, f)$  to  $(t', f)$ , and adjust the related edges.

**case 2** Suppose  $(t, f)$  is not a simple fact. We can use some  $M, K, R$ , and  $C$  strands to produce the non-simple tagged fact  $(t, f)$ . We then add the  $M, K, R$  and  $C$  strands to  $C'$  and adjust the related edges.

Add the  $u \rightarrow v$  edge to  $C'$  and remove the original edge  $u \rightarrow v$  from  $P$ .

After every edge is examined in the above manner, each of  $M', K', T', C', S', E',$  and  $D'$  strands in  $C$  will be transformed to some corresponding  $M, K, T, C, S, E, D,$  and  $R$  strands in  $C'$ .

We may prove that an honest strand in the B-scheme corresponds to an honest strand in the A-scheme and they are related by *similar* substitutions.

**Lemma 6.** Let  $temp$  be a strand template in the B-scheme and  $temp'$  be a strand template that defines the same role in the same protocol in the A-scheme. Let  $C$  be a bundle in the B-scheme and  $C'$  be a tagged bundle (in the A-scheme) obtained by transforming  $C$ . For each honest strand ( $subs'(temp')$ ) with respect to the substitution  $sub'$  in  $C'$ , there exists a corresponding honest strand ( $subs(temp)$ ) with respect to the substitution  $sub$  in  $C$  such that  $sub(v) = sub'(v)$ , for every  $v \in Var$ .

### 4.2.3 Unique Origination

Note that the facts on the honest strands in the B-scheme that are not interpreted according to the corresponding correct outmost-level tags must originate on the penetrator strands. Therefore, if a fact  $f$  originates on an honest or penetrator node in  $C'$ , then  $f$  originates on the corresponding honest or penetrator node, respectively, in  $C$ .

### 4.3 Secrecy and Authentication

In this subsection, we prove our second result, that is, if there is a type flaw attack on a protocol under the B-scheme, there is a type flaw attack under the A-scheme. We will discuss secrecy and authentication separately.

**Theorem (Secrecy).** Let  $temp$  be a template in the B-scheme and  $temp'$  be the template in the A-scheme for the same role. Let  $(t, v)$  be a tagged variable. Let  $h$  be a positive integer. Let  $Keys$  be a set of function templates. Let  $C$  be a bundle in the B-scheme and  $C'$  be a tagged bundle (in the A-scheme) obtained by transforming  $C$ . If there is a failure of secrecy in  $C$ , there will be a failure of secrecy in  $C'$  as well.

**Theorem (Authentication).** Let  $temp1$  and  $temp2$  be the templates for two roles in the B-scheme. Let  $temp1'$  and  $temp2'$  be the templates for the same roles in the A-scheme. Let  $X$  be the set of variables in the templates. Let  $h1$  and  $h2$  be two positive integers. Let  $Keys$  be a set of function templates. Let  $C$  be a bundle in the B-scheme and  $C'$  be a tagged bundle (in the A-scheme) obtained by transforming  $C$ . If there is a failure of authentication in  $C$ , there will be a failure of authentication in  $C'$  as well.

## 5 Conclusion

Heather, Lowe, and Schneider show that adding tags to fields in security messages can prevent type flaw attacks. We further simplified their scheme by combining and omitting certain tags. The main contribution of our work is the insight that an attacker *cannot* fake the outmost-level tags. The faked outmost-level tags will be detected by the real participants in a security protocol. All type flaw attacks, if they ever occur, must be related to fields inside an encrypted message in a security protocol.

Verification of security protocols is similar to verification of computer programs but the two carry complementing emphases. While we hope a computer program does what it is intended for, which is explicitly prescribed in the program's text, the most important properties of a security protocol is that it contains no *hidden* holes or weaknesses. These holes and weaknesses are not explicitly mentioned or even imagined in the description of the behavior of a security protocol.

## References

- [1] U. Carlsen, "Cryptographic protocol flaws: know your enemy," In Proceedings of Computer Security Foundations Workshop VII, 192-200, June 1994.
- [2] F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman, "Strand spaces: why is a security protocol correct?" In Proceedings of 1998 IEEE Symposium on Security and Privacy, 160-171, May 1998.
- [3] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," In Proceedings of 13th IEEE Computer Security Foundations Workshop, 255 -268, 2000.
- [4] G. Lowe, "A hierarchy of authentication specifications," In Proceedings of 10th Computer Security Foundations Workshop, 31-43, June 1997.
- [5] C.A. Meadows, "Analyzing the Needham-Schroder public-key protocol: A comparison of two approaches," In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, ed. *ESORICS'96*, LNCS 1146, 351-346, 1996.
- [6] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Comm. ACM*, 21(12), 993-999, 1978.

- [7] B.C. Neuman and S.G. Stubblebine, "A note on the use of timestamps as nonces," *ACM Operating Systems Reviews*, 27(2), 10-14, April 1993.
- [8] L.C. Paulson, "Proving security protocols correct," In Proceedings of 14th Symposium on Logic in Computer Science, 370-381, 1999.
- [9] T.Y.C. Woo and S.S. Lam, "A lesson on authentication protocol design," *ACM Operating Systems Reviews*, 28(3), 24-37, 1994.