JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 21, 59-84 (2005)

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

YAFEN LI, WUU YANG AND CHING-WEI HUANG
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, 300 Taiwan

A *type flaw attack* on a security protocol is an attack in which a field in a message that was originally intended to have one type is subsequently interpreted as having another type. Heather *et al.* proved that type flaw attacks can be prevented by 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 outermost level. The simplification process reduces the sizes of messages in the security protocol. We also formally prove that our simplified tagging scheme is as secure as Heather *et al.*' with the strand space method.

Keywords: network security, security protocol, type flaw, strand space, tagging

1. INTRODUCTION

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

Received November 13, 2003; revised June 29, 2004; accepted July 9, 2004.
Communicated by Ten-Hwang Lai.
* This work reported here was 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.

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 into believing is 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] proved that type flaw attacks can be prevented by 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 indicates encryption. Items in the same components are separated with commas.

Initial exchange

Msg 1. $A \rightarrow B : (agent, A), (nonce, N_a)$
Msg 2. $B \rightarrow S : (agent, B), (\{|nonce, timestamp|\}^{shared}$,
$\{(nonce, N_a), (timestamp, T_b)\}^{shared}_{Shared(B,S)}, (nonce, N_b)$
Msg 3. $S \rightarrow A : (\{|agent, nonce, shared, timestamp|\}^{shared}$,
$\{(agent, B), (nonce, N_a), (shared, K_{ab}), (timestamp, T_b)\}^{shared}_{Shared(A,S)}$,
$(\{|agent, shared, timestamp|\}^{shared}$,
$\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}^{shared}_{Shared(B,S)}), (nonce, N_b)$
Msg 4. $A \rightarrow B : (\{|agent, shared, timestamp|\}^{shared}$,
$\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}^{shared}_{Shared(B,S)})$,
$(\{|nonce|\}^{shared}, \{(nonce, N_b)\}^{shared}_{k_{ab}})$

Subsequent authentication

Msg 5. $A \rightarrow B : (nonce, N_a'), (\{|agent, shared, timestamp|\}^{shared}$,
$\{(agent, A), (shared, K_{ab}), (timestamp, T_b)\}^{shared}_{Shared(B,S)},)$
Msg 6. $B \rightarrow A : (nonce, N_b'), (\{|nonce|\}^{shared}, \{(nonce, N'_a)\}^{shared}_{K_{ab}})$
Msg 7. $A \rightarrow B : (\{|nonce|\}^{shared}, \{(nonce, N'_b)\}^{shared}_{K_{ab}})$

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 outermost level. The Neuman-Stubblebine protocol with our simplified tags is shown below:

Initial exchange

Msg 1. $A \rightarrow B : A, N_a$
Msg 2. $B \rightarrow S : B, \{(agent, nonce, timestamp), (A, N_a, T_b)\}^{shared}_{Shared(B,S)}, N_b$
Msg 3. $S \rightarrow A : \{(agent, nonce, shared, timestamp), (B, N_a, K_{ab}, T_b)\}^{shared}_{Shared(A,S)}$,
$\{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}, N_b$
Msg 4. $A \rightarrow B : \{(agent, shared, timestamp), (A, K_{ab}, T_b)\}_{Shared(B,S)}$,

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

Subsequent authentication

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

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

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

Consider the fourth message in the protocol. Note that, in our simplified tagging scheme, the outermost-level tags – $\{|agent, key, timestamp|\}^{shared}$ and $\{|nonce|\}^{shared}$ – are omitted. Furthermore, the three tags insides the encrypted component – *agent*, *key*, and *timestamp* – are combined into a single tag – (*agent*, *key*, *timestamp*). Because in a security protocol, the number of different possible tag types is 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 can also prove 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 outermost-level tags are omitted and the simplifications done in the A-scheme are employed. We then show that the B-scheme is as secure as the A-scheme. We can 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 employed in [2, 3], the definitions in this paper (in sections 3, 4, and 5) are adapted from those in the above works, with modifications needed to reflect our simplified tagging scheme. We provide these definitions in order to help the reader to understand the proofs which we will give later.

Attacks based on type flaws are quite common in security protocols. Meadows [5] also discussed a similar type flaw attack on the Needham-Schroder protocol [6]. The Woo-Lam protocol π_1 [9] is also vulnerable to type flaw attacks [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 their 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*: $K \to K$. *inv* maps a member of a key pair to the other for an asymmetric key and maps a symmetric key to itself.
- Two binary operators for encryption and joining, respectively are

$$encr: K \times A \to A$$
$$join: A \times A \to A$$

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 to 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
$\bullet+(A, N_a)$
$\Downarrow$
$\bullet-(B, \{A, N_a, k_{ab}, T_b\}_{K_{as}}, \{A, K_{ab}, T_b\}_{K_{as}}, N_b)$
$\Downarrow$
$\bullet+(\{A, K_{ab}, T_b\}_{K_{bs}}, \{N_b\}_{K_{ab}})$

Fig. 1. A strand.

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 $\langle \pm a_1, \pm a_2, \ldots, \pm a_n \rangle$, 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 (Fig. 1 shows a sample strand):

- If nodes 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 relationship between n_i and n_{i+1}.
- If nodes $n_i = + a$ and $n_j = - a$, then we write $n_i \to n_j$. This means that node n_i sends a 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 another strand receives the message. Fig. 2 shows 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 $n_2 \to n_1 \in C$.
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.

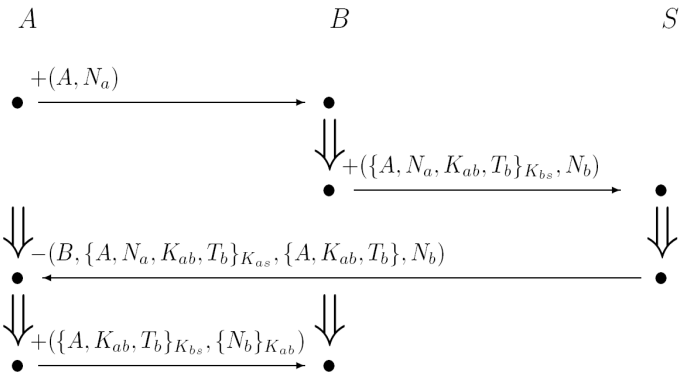

Fig. 2. A bundle.

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:

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

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

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

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 templates:

sub : $var \rightarrow simple\ fact$
$subt$: $template \rightarrow fact$
$subs$: $strand\ template \rightarrow honest\ strand$

These are defined as

$$subt(v) = sub(v), \forall v \in Var$$
$$subt(g(v_1, ..., v_n)) = (g(sub(v_1), ..., sub(v_n))), \forall g \in Fn$$
$$subt(\{t\}_k^{tk}) = (\{sub(t)\}_{subt(k)}^{tk})$$
$$subs((s_1, t_1), ..., (s_k, t_k)) = ((s_1, subt(t_1)), ..., (s_k, subt(t_k)))$$

2.3 Penetrators

Two ingredients characterize a 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 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, strand s 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, 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 outermost-level tags) is as

secure as that under Heather *et al*.'s (full) tagging scheme. That is, if there is an attack on 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 discussed in [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 will 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 as follows:

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

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 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:

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

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

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[1], 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 tagged facts $(t_1, f_1), (t_2, f_2), \ldots, (t_k, f_k)$ (here, *ts* and *fs* must have exactly the same number of elements). We also use *tf* to represent a tagged fact. We 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:

[1] This can be implemented by including sufficient redundancy within the encryption scheme.

- $(t, f) \sqsubset (t, f)$ (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 $(t_i, f_i) \sqsubset (ts', fs')$ 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 $f_i \sqsubset (ts', fs')$, for every f_i in f_s

Correct Tagging We will 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 will now define what it means for a tagged fact to be *correctly tagged at the outmost level*:

- $TopLevelWellTagged(agent, x) \Leftrightarrow x \in Agent$
- $TopLevelWellTagged(nonce, x) \Leftrightarrow x \in Nonce$
- $TopLevelWellTagged(timestamp, x) \Leftrightarrow x \in Timestamp$
- $TopLevelWellTagged(shared, x) \Leftrightarrow x \in Sharedkey$
- $TopLevelWellTagged(public, x) \Leftrightarrow x \in Publickey$
- $TopLevelWellTagged(private, x) \Leftrightarrow x \in Priavtekey$
- $TopLevelWellTagged(ts, fs) \Leftrightarrow TopLevelWellTagged(t_i, f_i)$, 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 next discuss 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 *uniquely originates 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:

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

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

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

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

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:

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

These are defined as

$subt(v) = sub(v)$
$subt(g(v_1, \ldots, v_n)) = (g(sub(v_1), \ldots, sub(v_n)))$, $\forall g \in Fn$
$subt(\{\{tt\}_k^{tk}) = (\{subtt(tt)\}_{subt(k)}^{tk})$
$subtt(tt) = (t_1, \ldots, t_k, subt(t'_1), \ldots, subt(t'_k))$
where $tt = (t_1, \ldots, t_k, t'_1, \ldots, t'_k)$ and each (t_i, t'_i) is a simple tagged template of tt
$subs(s) = ((s_1, subtt(tt_1)), \ldots, (s_k, subtt(tt_k)))$
where $s = ((s_1, tt_1), \ldots, (s_k, tt_k))$ 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 outermost level.

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

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 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 encryption might not have the expected type.

3.1.4 Penetrator traces

Following [1, 7], we assume that there is 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 modifications to the *M*, *C*, *S*, and *R* traces.

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

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 that non-simple tagged facts can be produced by means of concatenation:

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

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

C Concatenation $\langle - (ts_1, fs_1), \ldots, - (ts_k, fs_k), + (ts_1, \ldots, ts_k, fs_1, \ldots, fs_k) \rangle$;

S Separation $\langle - (t_1, \ldots, t_k, f_1, \ldots, f_k), + (t_1, f_1), \ldots, + (t_k, f_k) \rangle$;

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 $\langle - (tk', k'), - (\{|ts|\}^{tk}, \{(ts, fs)\}_k^{tk}), + (ts, fs) \rangle$. Here, tk and tk' are tags rep-

resenting 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 means of separation, retagging of particular components, and then concatenation. No other penetrator traces interfere with the tags of their messages.

We now have 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 will explain the transformation of 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 the terms are well-tagged. We will proceed in two steps: (1) We will define the properties the transformation ϕ must satisfy. (2) We will then show that such a transformation ϕ can always be constructed.

3.2.1 Defining the transformation function

The following definition captures the required properties of the transformation function ϕ:

Definition Given a bundle C, we define

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

as a transformation function for C if:

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

[2] T is the set of simple facts that the penetrator can produce.

8. When ϕ 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, $\forall tf \in SimpleTaggedFacts(C)$[3] $\forall f \in T$ $[[\neg TopLevelWellTagged(tf) \wedge f \sqsubset \phi(tf)] \Rightarrow \forall tf' \in TaggedFacts(C)$[4]$[tf \not\sqsubset tf' \Rightarrow f \not\sqsubset \phi(tf')]]$.

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

Lemma 2 Given a bundle C, there exists a transformation function ϕ 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 ϕ 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 ϕ, *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 ϕ is also an honest strand.

3.2.3 Transforming penetrator strands

We will show that, given the penetrator strands in a bundle C and a transformation ϕ, 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 $S = \langle - (t_1, f_1), \ldots, - (t_k, f_k), + (t_1, \ldots, t_k, f_1, \ldots, f_k) \rangle$. Define $S' = \langle - \phi(t_1, f_1), \ldots, - \phi(t_k, f_k), + \phi(t_1, \ldots, t_k, f_1, \ldots, f_k) \rangle$. Because $\phi(t_1, \ldots, t_k, f_1, \ldots, f_k) = (\phi(t_1, f_1)_1, \ldots, \phi(t_k, f_k)_1, \phi(t_1, f_1)_2, \ldots, \phi(t_k, f_k)_2)$, S' is a well-tagged C strand.

---

[3] $SimpleTaggedFacts(C)$ is the set of simple tagged facts in C.
[4] $TaggedFacts(C)$ is the set of tagged facts in C.

S Separation Let $S = \langle -(t_1, \ldots, t_k, f_1, \ldots, f_k), +(t_1, f_1), \ldots, +(t_k, f_k)\rangle$. Define $S' = \langle +\phi(t_1, \ldots, t_k, f_1, \ldots, f_k), -\phi(t_1, f_1), \ldots, -\phi(t_k, f_k)\rangle$. Because $\phi(t_1, \ldots, t_k, f_1, \ldots, f_k) = (\phi(t_1, f_1)_1, \ldots, \phi(t_k, f_k)_1, \phi(t_1, f_1)_2, \ldots, \phi(t_k, f_k)_2)$, 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-taged K strand.

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

D Decryption Let $S = \langle -(tk', k'), -(\{|ts|\}^{tk}_k, \{(ts, fs)\}^{tk}_k), +(ts, fs)\rangle$, where k and k', which have types tk and tk', respectively, are inverses of each other. Define $S' = \langle -\phi(tk', k'), -\phi(\{|ts|\}^{tk}_k, \{(ts, fs)\}^{tk}_k), +\phi(ts, fs)\rangle$. Because $\phi(\{|ts|\}^{tk}_k, \{(ts, fs)\}^{tk}_k) = (\{|ts|\}^{tk}_k, \{\phi(ts, fs)\}^{tk}_{\phi(tk,k)_2})$, 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 $\neg TopLevelWellTagged(t_0, f)$, 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 Fig. 3.

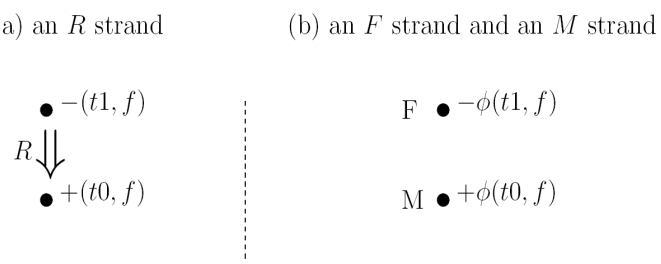

Fig. 3. If $\neg TopLevelWellTagged(t_0, f)$, then replace the R strand in (a) with an F and an M strands in (b).

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

If $t_1 = t_0$, we are done. Here, we have a well-tagged F strand and a well-tagged M strand. Otherwise, $\neg TopLevelWellTagged(t_1, f)$. According to Lemma 1, the 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 an 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 $n0'$ and nk.
2. Each of the negative nodes on the R strands is replaced with an F strand.
3. Other nodes on the R strands are replaced with an M strand and some T strands if the terms of nodes are the same. Suppose the terms of nodes n'_{k-1}, n'_2 and n'_1 are the same. We have the situation shown in Fig. 4.
4. If $t_k = t_0$ (that is, we can find an earlier R strand with an initial node labelled with $-(t_0, f)$ or an earlier M strand with a node labelled with $-(t_0, f)$), then the $\rightarrow$ successor of n_0' in C becomes the $\rightarrow$ successor of $\phi(n)$ in C'. This is shown in part (1) of Fig. 4. If not (that is, we can find an earlier M strand to produce (t, f) for some $t \in Tag$), then 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 Fig. 4.

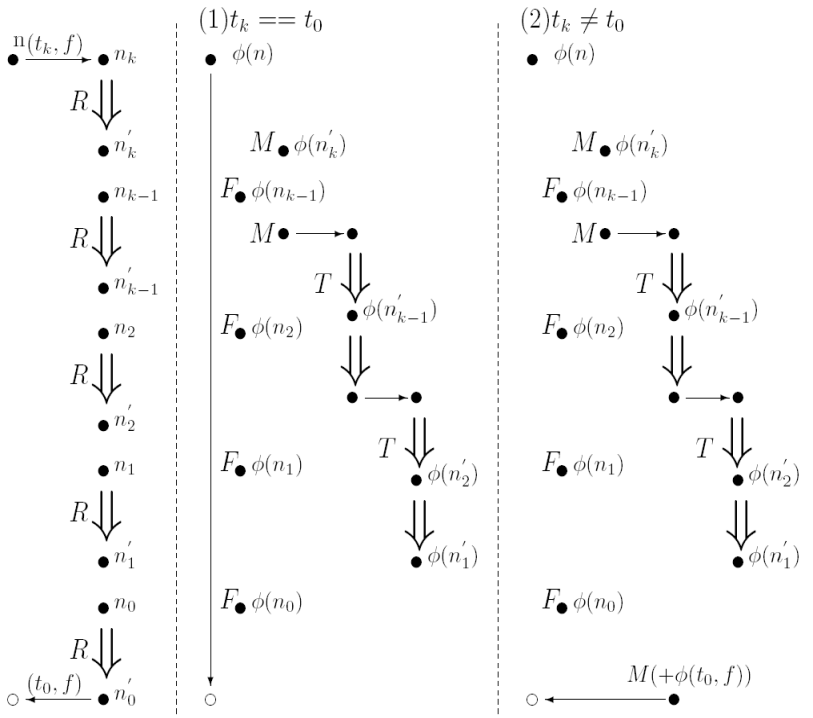

Fig. 4. If *TopLevelWellTagged*(t_0, f), then replace the R strands with some F, M, and T strands.

3.2.4 Unique origination

We will 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 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 above four possibilities do not cause any problems, for if the term originates multiple times in C', then 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 ϕ 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 will prove our first result: if there is a type flaw attack on a protocol under the A-scheme, there is a type flaw attack under the 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 let 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, then there will be a failure of secrecy in C' as well.

Theorem (Authentication) Let *temp*1 and *temp*2 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 let C' be a well-tagged bundle in the HLS-scheme, obtained by transforming C. If there is a failure of authentication in C, then 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 will present the new model that will be used to prove the second point: a protocol under the simplified tagging scheme without the outermost-level

tags is as secure as one under the simplified tagging scheme with the outermost-level tags; hence, is as secure as one 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 be a corresponding attack under the A-scheme (and, hence, under the HLS-scheme). The implication of this result is that, even without the outermost-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:

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

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

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

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 outermost-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 that maps a variable to a simple fact. A substitution can be lifted to the templates, the tagged templates, and the strand templates:

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

These are defined as follows:

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

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 is 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 strands: M', F', T', C', S', K', E', or D'. Note that we do not need the R' strands in this B-scheme because the outermost-level tags are omitted.

M′ Text message $\langle + f \rangle$ for $f \in T$. Note that an M' strand only produces simple facts and non-simple facts can be produced by means of concatenation:
F′ Flushing $\langle - fs \rangle$.
T′ Tee $\langle - fs, + fs, + fs \rangle$.
C′ Concatenation $\langle - fs_1, \ldots, - fs_k, + (fs_1, \ldots, fs_k) \rangle$.
S′ Separation $\langle - (f_1, \ldots, f_k), + f_1, \ldots, + f_k \rangle$.
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 to be keys of the 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.

4.2.1 Transforming honest strands

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

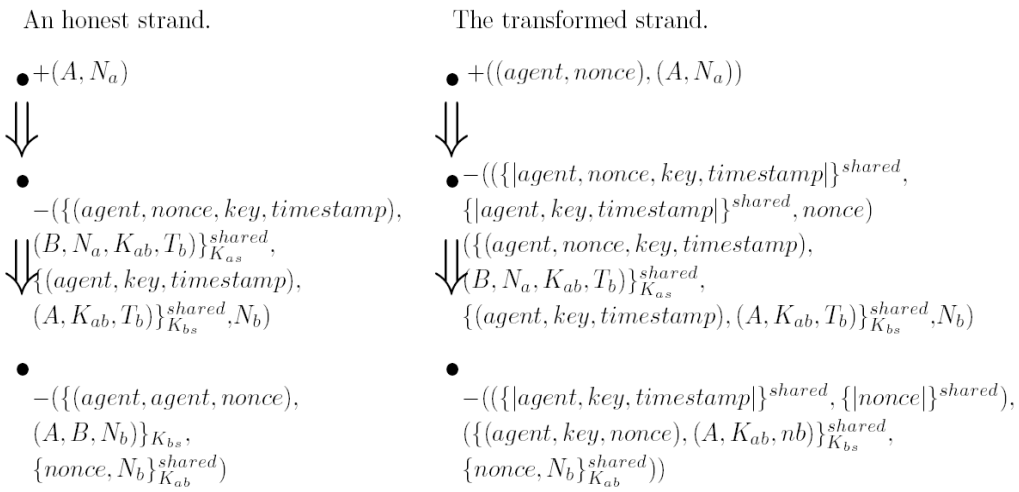

Fig. 5. Transforming honest strands.

4.2.2 Transforming penetrator strands

Since we know how to transform honest strands, what we need to do now is 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 turn until the set P becomes an empty set. There are five cases as 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 Fig. 6), then 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.

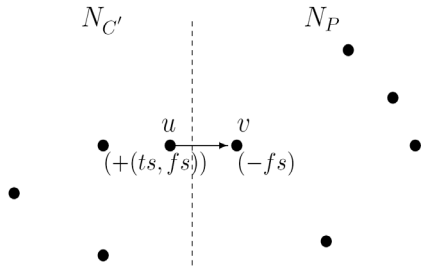

Fig. 6. We prepare to replace the edge $u \rightarrow v$.

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 Fig. 7), then there are two cases to consider.

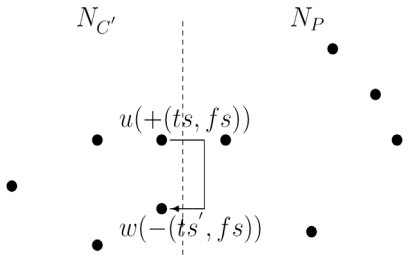

Fig. 7. Create a new edge $u \rightarrow w$.

Case 1. If the outermost-level tags of u and w are the same, then we are done.

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 Fig. 8.

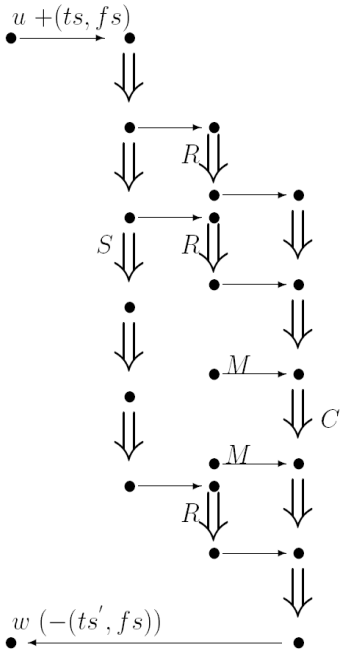

Fig. 8. The case $ts \neq ts'$.

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 predecessors of both u and v on the same strand have the outermost-level tags (shown in Fig. 9), then we can find the outermost-level tag of v as follows:

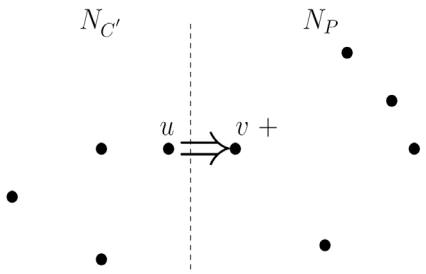

Fig. 9. Finding the outermost-level tag of v.

Case 1. If v belongs to a T' (tee) strand, then the outermost-level tags of the node v and the node following v are the same as those of u.

Case 2. If v belongs to a C' (concatenation) strand, then the outermost-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, then we substitute some M and R strands and an S strand for the S' strand. The S strand separates the tagged fact. The M and K strands produce facts that are terms of nodes in the S' strand but are not terms of nodes in the S strand. Finally, we adjust the related edges properly.

Case 4. If v belongs to an E' (encryption) strand, then we have the situation shown in Fig. 10.

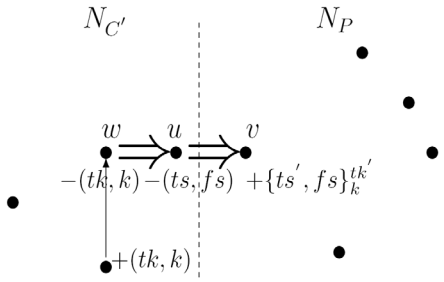

Fig. 10. An E' strand.

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

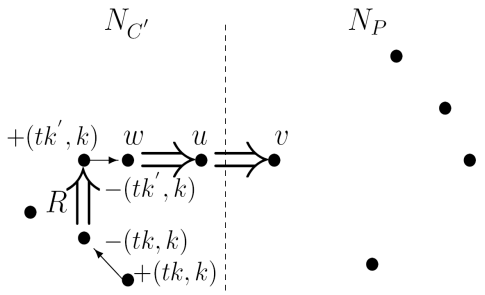

Fig. 11. Introduce an *R* strand.

If $ts \neq ts'$ (as shown in Fig. 10), then we use an *S* strand to separate (*ts*, *fs*), create some *M* and *K* strands to produce facts that are sub-facts of (*ts′*, *fs*) but are not sub-facts of (*ts*, *fs*), use *R* strands to change the tags, and use a *C* strand to concatenate the tagged facts. We then add the *M*, *K*, *R*, and *C* strands to *C′*. We replace the term (*ts*, *fs*) of node *u* with (*ts′*, *fs*) and adjust the related edges. The outermost-level tag of the term of node *v* is $\{|ts'|\}^{tk'}$. This is shown in Fig. 12.

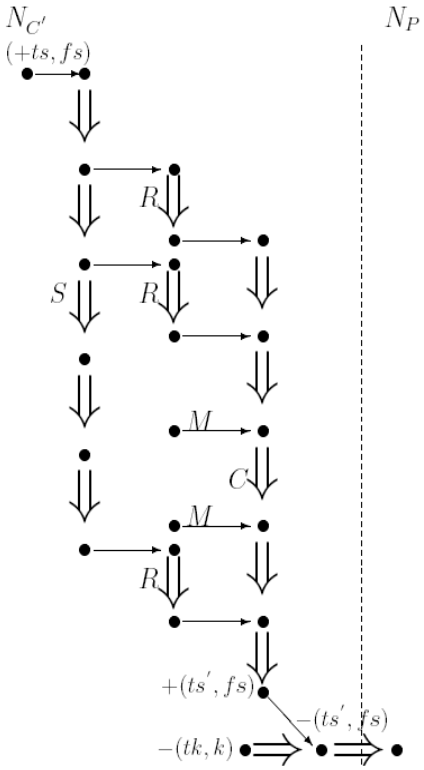

Fig. 12. Replace the – (*ts*, *fs*) tag of node *u* with – (*ts′*, *fs*) in Fig. 10.

Case 5. If v belongs to a D' (decryption) strand, then we have the situation shown in Fig. 13.

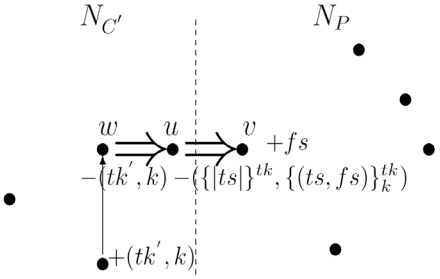

Fig. 13. A D' strand.

If tk and tk' are not tags representing inverse key types, then we can use an R strand to change the tag of w (shown in Fig. 13) to (tk'', k) (shown in Fig. 14). (Note that t'' is chosen to be the inverse key type of tk.) The outermost-level tag of v is *GetTags*(the encryption body of the second node). This is shown in Fig. 14.

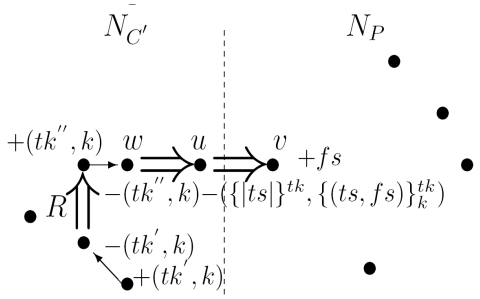

Fig. 14. Use an R strand to change the tag of w.

Finally, we replace the outermost-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 Fig. 15), then we can select an appropriate tag $t \in Tag$. We 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.

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 Fig. 16), then we will consider the following two cases.

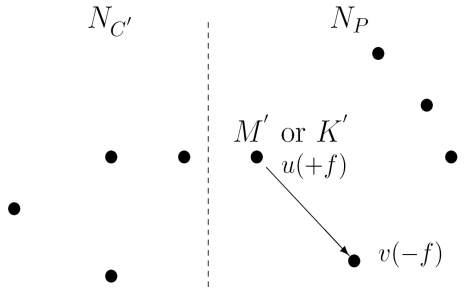

Fig. 15. An M' or K' strand in N_P.

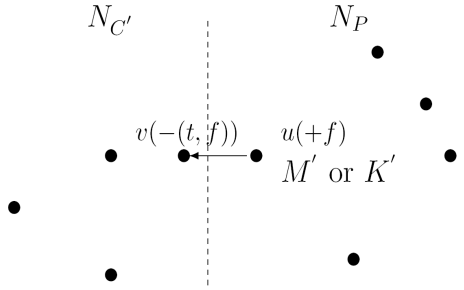

Fig. 16. An M' or K' strand from N_P to NC'

Case 1. Suppose (t, f) is a simple fact.

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

Subcase 2. If node u is on a K' (key) strand, we do one of the following:

- If *WellTagged*(t, f), then we replace $(+ f)$ with $(+ (t, f))$, which is a K strand.
- If $\neg$*WellTagged*(t, f), then 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.

We 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 the M', K', T', C', S', E', and D' strands in C is transformed into some corresponding M, K, T, C, S, E, D, and R strands in C'.

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

Lemma 6 Let *temp* be a strand template in the B-scheme, and let *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 let 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 outermost-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 will prove our second result; that is, if there is a type flaw attack on a protocol under the B-scheme, then 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 let $temp'$ 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 let C' be a tagged bundle (in the A-scheme) obtained by transforming C. If there is a failure of secrecy in C, then there will be a failure of secrecy in C' as well.

Theorem (Authentication) Let $temp1$ and $temp2$ be templates for two roles in the B-scheme. Let $temp1'$ and $temp2'$ be 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 let C' be a tagged bundle (in the A-scheme) obtained by transforming C. If there is a failure of authentication in C, then there will be a failure of authentication in C' as well.

5. CONCLUSIONS

Heather, Lowe, and Schneider showed that adding tags to fields in security messages can prevent type flaw attacks. We have 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 outermost-level tags. Any attempt to fake the outermost-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, and the two procedures have complimentary emphases. While we hope that a computer program does what it is intended for, which is explicitly prescribed in the program's text, the most important property of a security protocol is that it contains no *hidden* holes or weaknesses. Such holes and weaknesses are not explicitly mentioned or even hinted at 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*, 1994, pp. 192-200.
2. F. J. T. Fabrega, 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*, 1998, pp. 160-171.
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*, 2000, pp. 255-268.
4. G. Lowe, "A hierarchy of authentication specifications," in *Proceedings of 10th Computer Security Foundations Workshop*, 1997, pp. 31-43.
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, 1996, pp. 351-346.
6. R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, 1978, pp. 993-999.
7. B. C. Neuman and S. G. Stubblebine, "A note on the use of timestamps as nonces," *ACM Operating Systems Reviews*, Vol. 27, 1993, pp. 10-14.
8. L. C. Paulson, "Proving security protocols correct," in *Proceedings of 14th Symposium on Logic in Computer Science*, 1999, pp. 370-381.
9. T. Y. C. Woo and S. S. Lam, "A lesson on authentication protocol design," *ACM Operating Systems Reviews*, Vol. 28, 1994, pp. 24-37.

Yafen Li (黎雅芬) received her B.S. degree in Computer Science and Information Engineering from National Chiao Tung University in 2001 and the M.S. degree in Computer and Information Science from National Chiao Tung University in 2003. Currently she is an engineer in Trend Micro Company. She is very interested in the study of network security.

Wuu Yang (楊武) received his B.S. degree in Computer Science from National Taiwan University in 1982 and the M.S. and Ph.D. degrees in Computer Science from University of Wisconsin at Madison in 1987 and 1990, respectively. Currently he is a professor in the National Chiao Tung University, Taiwan, R.O.C. Dr. Yang's current research interests include Java and network security, programming languages and compilers, and attribute grammars. He is also very interested in the study of human languages and human intelligence.

Ching-Wei Huang (黃經緯) received his B.S. degree in Mathematics from National Tsing Hua University in 1994 and the M.S. degree in Computer Science from National Chiao Tung University in 1997. Currently he is doing his Ph.D degree in Computer Science. His current research interests include distributed systems, peer-to-peer systems, and parallel computing.