On Preventing Replay Attacks on Security Protocols

Sreekanth Malladi, Jim Alves-Foss, Robert B. Heckendorn
Center for Secure and Dependable Systems
Department of Computer Science
University of Idaho
Moscow, ID 83844 USA
{msskanth, jimaf, heckendo}@cs.uidaho.edu

***Abstract*— *Replay attacks* on security protocols have been discussed for quite some time in the literature. However, the efforts to address these attacks have been largely incomplete, lacking generality and many times in fact, proven unsuccessful. In this paper we address these issues and prove the efficacy of a simple and general scheme in defending a protocol against these attacks. We believe that our work will be particularly useful in security critical applications and to protocol analyzers that are unable to detect some or all of the attacks in this class.**

***Index Terms*—security protocols, replay attacks, adapted strand spaces, run identifiers, component numbers.**

I. Introduction

REPLAY attacks have been discussed for quite some time in the literature (eg. [1], [2], [3], [4]). We generalize the definition of a replay attack as: *an attack on a security protocol using replay of messages from a different context into the intended (or original and expected) context, thereby fooling the honest participant(s) into thinking they have successfully completed the protocol run.*

Syverson in [2] presents an exhaustive taxonomy of replay attacks classifying them at the highest level as run-external and run-internal attacks, basing on the origin of messages. Each of these can be further classified into interleavings, classic replays, reflections, deflections and straight replays basing on message destination.

Traditional methods to include nonces have proven to be of little value against replay attacks (eg. [5], [6]). Attempts to use time-stamps in messages were beset with problems such as dealing with the asynchronous world [1]. Gong and Syverson present fail-stop protocols under a restrictive class of protocol design rules, that avoid these attacks under certain conditions [3]. Some other attempts have been exclusively directed towards countering reflections [7]. These mechanisms work by including the identity of sender, recipient or both in messages. Suggestions of binding cryptographic keys to their intended use, specialized use of shared keys to identify sender and receiver have been cited in numerous places including [8], [9], [10], [4] but are effective only in limited contexts eg. reflections and deflections but not straight replays. Syverson discusses these counter-measures exhaustively in [2].

Project funded in part by DARPA under grant MDA972-00-1-0001

Strategies presented to counter reflection attacks using format asymmetry fail if the format asymmetry is itself attackable. Although format asymmetry (type-flaws) was proven to withstand type-flaw attacks by Heather *et. al* [11], the suggested scheme of component numbering as a corollary does not consider attacks using interleaving of different protocols[1]. Guttman *et. al* prove "protocol independence" through disjoint encryption and suggest a protocol numbering scheme to achieve disjoint encryption. However, there is not yet an international standard on protocol numbering (to identify each protocol). Another suggestion in the same paper is to use a different keying material for each application—an indeed strong assumption since it is unlikely to be followed by all users due to the high cost of certified keys. This was also discussed in [8].

Thus, one can observe some visible characteristics in all these solutions—They are either unsuccessful, or too specific. Some of them are too expensive to implement and some others are interdependent (eg. as discussed above where solutions using format asymmetry depend on type tagging or component numbering and these in turn depend on using disjoint encryption). Hence, an interesting question to ask would be, "can these interdependencies be taken advantage of to launch new kinds of attacks?".

Most of the automated analyzers also fail to detect at least one attack in this class. Although, NRL protocol analyzer was observed to have been able to detect all types of replays given by Syverson, it was still observed to be difficult to analyze for specific attacks in this class [12]. However, prevention is another matter.

Carlsen presents a list of type information that can be attached to messages and elements [13]. These include protocol identifier, protocol run identifier, primitive types of data items, transmission step identifier and message subcomponent identifier. Aura [4] studies these techniques and comes up with some strategies against replay attacks that are neither necessary nor sufficient but enhance the robustness of a protocol. A recent trend in the literature has been to prove protocol security against specific attacks by suggesting tagging messages with one of the type information suggested by Carlsen (eg. [14], [11]).

[1]Intuitively, different protocols can have different messages with the same component number, still making it vulnerable to type-flaws. The original suggestion to tag with primitive types of data items, however, would be effective in the presence of interleaving of different protocols but is expensive to implement

In the same spirit, we prove that *all* replay attacks can be prevented by tagging each encrypted component with a *session-id* (another name for Carlsen's protocol run identifier) and a *component number* (renaming Carlsen's message subcomponent identifier). However, unlike previous attempts, our suggestion is a *general* solution and prevents all types of replay attacks in Syverson's taxonomy. Although it is not an entirely new solution, it solves the problem of replays using a combined solution that is devoid of any possible vulnerabilities due to interdependencies.

Introducing component numbers inside encryptions is intuitive, but the generation and use of session-ids requires some explanation. Some have discussed tagging messages with all the information that is in possession of a principal and relevant to the protocol [15]. This is also called the *principle of full information*. Aura [4] hints at a trivial way of including a hash of all previous messages in a protocol run, almost as a substitute to the principle of full information and the run identifier. This is prudent to some extent. In fact, as Aura points out, it is enough to include only a hash of the redundant data that is already known to the receiver. This doesn't adversely affect the performance for obvious reasons. However, careful observation of this suggestion reveals a possible vulnerability—two protocol runs can overlap in the executed information at a typical stage of the runs!

Therefore, we suggest a different approach to generate session-ids to identify runs. For the purpose of this discussion, it suffices to know that such identifiers can be generated to be used in an effective way and will possess a necessary property—remaining unique to every protocol run. Briefly, all participants need to choose a random number, and combine those into a single long string of random bits. This value should be hashed together with the identities of all principals, reducing the chance of an accidental match in session-ids to a great extent. Two features are necessary to generate such an identifier 1. Every principal should possess the same hash function. 2. A change in one of the random numbers/principals' identities should make the resulting value differ from it's original value.

Observe that the generated session-id is different in properties from other similarly used identifiers. For example, the run identifier used in Otway-Rees protocol [16] is generated by only one participant and hence was shown to be prone to replay attacks [17]. It is also similar to "cookies" coined in Photuris [18] which are an add-on that can be used to make a protocol more resistant to DOS attacks. A cookie is a unique nonce computed from the names of the sending and receiving parties and local secret information that only the sender possesses. Cookies provide initially weak authentication to users while they aid in subsequent establishment of strong authentication. Session-ids are similar to cookies in the kind of initial assumptions and their ultimate use, except, unlike cookies where only the sender is aware of the value, a session-id is publicly known.

However, the publicly known identifier needs to be unique for every protocol run. Intuitively, a dishonest principal might use a different value from the pre-agreed upon value. (In fact, this will be definitely true if he replays components from previously completed runs and not from interleaved runs). However, the proof we are going to present will establish that even such behavior does not succeed in breaking a protocol in this scheme. Further, hashing with participants' identities (cited as useful in numerous places including [4], [7], [10]) prevents other attempts to spoof user identities and launch man-in-middle type attacks.

The proof we are going to present in this paper follows a simple concept to establish the desired results, following a proof structure laid out in [11]:

If a protocol is secure in the absence of replays, it is also secure under our tagging scheme in the presence of replay attacks
OR

Whenever there is an attack on a protocol using the tagging scheme, there should also be an attack on the protocol in the absence of replays

The utility of the result of this paper is manifold. Firstly, it reduces the task of protocol analyzers that fail to detect any subset of replay attacks and increases the trust in the remaining analysis. Secondly, it gives more leverage to a protocol designer with the implicit protection that it provides against many known threats. Lastly, it is relatively inexpensive to implement such a scheme especially compared to those for example, that require unique keys for each application.

The rest of the paper is organized as follows: In the next section (2), we will introduce our model of a protocol. In section 3, we will show that any given bundle can be transformed into an equivalent but well-tagged bundle. In section 4, we will prove our main result. We illustrate our concepts on the Otway-Rees protocol as an example in section 5 and end with a conclusion.

II. The Protocol Model

A. Tags, Facts and Subfacts

Tags and Facts[2] are defined in the model as:

$$\begin{aligned} Tag &::= \mathsf{JOIN}\ SID\ CNO \\ Fact &::= UF \mid EF \mid \mathsf{JOIN}\ Fact\ Fact \\ UF &::= \mathsf{JOIN}\ Atom\ UF \\ EF &::= \mathsf{ENCR}\ TF \\ TF &::= \mathsf{JOIN}\ Tag\ Fact \end{aligned}$$

where UF, EF, TF represent unencrypted, encrypted and tagged fact respectively.

$\mathbf{uf}$, $\mathbf{ef}$ denote the set of unencrypted and encrypted facts respectively. $\mathbf{Atom}$ is the set of atomic values (eg. $Alice$, Bob, N_A, $\mathsf{PubKey}(A)$ etc.) assumed to be contained in a protocol.

Often we need a *projection function* on tagged facts to obtain the tag or fact components, defined as:

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

We will denote the set of session-ids as $\mathbf{sid}$ and the set of component numbers as $\mathbf{cno}$. A typical tag in a run α shall be written as, (sid_α, cno). The first part is the session-id of α and the second part is the component number for one of the

[2]Facts, components or messages will be used interchangeably but message will be used to mean the entire collection of facts sent in a single protocol step.

encrypted facts of α. JOIN and ENCR represent concatenating two data items and encrypting a data item respectively. When two data items $a,\ b$ are to be concatenated, we will write, $a\ .\ b$ or (a, b). When a data item a is to be encrypted with a key k, we will write, $\{a\}_k$. Also, subfact relation $\sqsubset$ on facts is defined as follows:

Definition 1: The subfact relation is the smallest relation on facts such that:

1) $f \sqsubset f$;
2) $f \sqsubset \{tf'\}_{k'}$ if $f \sqsubset (tf')_2$;
3) $f \sqsubset (f_1, f_2)$ if $f \sqsubset f_1 \vee f \sqsubset f_2$.

Also, $f \sqsubset tf$ if $(t, f) \sqsubset tf$.

B. Adapting Strand Space Model with Tagged Facts

In this section, we define a new adapted strand space model from the original strand space model [19] to suit tagged facts.

Definition 2: A $strand$ is a sequence of communications by either an agent (honest or dishonest) in a protocol run. It is represented as a sequence of facts $\langle \pm f1, \pm f2, \ldots, \pm fn \rangle$. A '+' indicates transmission of a fact and '$-$' indicates reception of a fact. Every node in the set of nodes $\mathcal{N}$ transmits or receives a fact ($\pm f$) and belongs to a unique strand.

1) Let n_i, n_{i+1} be consecutive nodes on the same strand. Then, there exists an edge $n_i \Rightarrow n_{i+1}$ in the strand.
2) If $n_i = +f$ and $n_j = -f$ are nodes belonging to different strands, then there exists an edge $n_i \rightarrow n_j$.
3) $\mathcal{N}$ together with both the sets of edges $n_i \Rightarrow n_{i+1}$ and $n_i \rightarrow n_j$ is a directed graph, $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$.

A $bundle$ represents a particular event history of the communication. It is an acyclic, finite subgraph of $\langle \mathcal{N}, (\rightarrow \cup \Rightarrow) \rangle$. Formally, if $\rightarrow_C \subset \rightarrow$, $\Rightarrow_C \subset \Rightarrow$ and $(\rightarrow_C \cup \Rightarrow_C)$ is a finite set of edges, then $C = (\rightarrow_C \cup \Rightarrow_C)$ is a bundle if:

1) whenever $n_2 \in \mathcal{N}_C$ receives a fact, there exists a unique n_1 such that $n_1 \rightarrow_C n_2$;
2) Whenever $n_2 \in \mathcal{N}_C$ with $n_1 \Rightarrow n_2$, $n_1 \Rightarrow_C n_2 \in C$;
3) C is acyclic.

A node n is an *entry point* to a set of facts $\mathbf{F}$ if no node previous to n has a fact $+f$ with $f \in \mathbf{F}$. A fact *originates* on n if n is an entry point to $\{f' \mid f \sqsubset f'\}$. Similarly, a tagged fact tf *originates* on n if n is an entry point to $\{tf' \mid tf \sqsubset tf'\}$. A fact or a tagged fact is $uniquely$ originating in a bundle if it originates on a unique node of the bundle.

1) Honest Strands: We use the concept of strand templates [11] to define roles in a protocol. These templates make use of a set of variables, **Var** defined as below:

$$\begin{aligned} Var &::= UV \mid EV \mid \mathsf{JOIN}\ Var\ Var \\ UV &::= AtomVar \mid \mathsf{JOIN}\ AtomVar\ UV \\ EV &::= \mathsf{ENCR}\ TaggedVariable \\ TaggedVariable &::= \mathsf{JOIN}\ TagVar\ Var \end{aligned}$$

where, UV - Unencrypted Variable, EV - Encrypted Variable. $TagVar$ - Tag Variable.

Var consists of unencrypted (**uv**), encrypted (**ev**) and function variables (**fv**)[3]) with $\mathbf{fv} \subset \mathbf{uv}$. We define an instantiation function ins to instantiate elements in **Var**, to correspond to elements in **Fact**.

Let,

$$\mathsf{ins}_t : \mathbf{TagVar} \rightarrow \mathbf{Tag}, \text{ and } \mathsf{ins}_v : \mathbf{AtomVar} \rightarrow \mathbf{Atom}$$

so that, $\forall tv \in \mathbf{TagVar} \wedge v \in \mathbf{AtomVar}$,
$\mathsf{ins_t}(tv) = t$ and $\mathsf{ins_v}(v) = f$ for some $t \in \mathbf{Tag} \wedge f \in \mathbf{Atom}$.

Combining $\mathsf{ins_t}$ and $\mathsf{ins_v}$, we define ins as:

$$\mathsf{ins} : \mathbf{Var} \rightarrow \mathbf{Fact}$$

so that $\forall v \in \mathbf{Var}, \mathsf{ins}(v) = f,$ for some $f \in \mathbf{Fact}$.

Also, ins can be defined on all possible variables as:

$$\mathsf{ins}(v) = \begin{cases} \{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(k')} & v = \{(tv', v')\}_{k'} \\ (\mathsf{ins}(v_1), \mathsf{ins}(v_2)) & v = (v_1, v_2). \end{cases}$$

As an example, let $temp$ represent the role 'b' in the Woo and Lam protocol Π_1 [15]:

$$\begin{aligned} temp \mathrel{\widehat{=}} \langle -(a), +(n_b), -(x), +(\{t_2, a, b, x\}_{sh_{bs}}), \\ -(\{t_3, a, b, n_b\}_{sh_{bs}}) \rangle \end{aligned}$$

('x' represents that 'b' is not expected to decrypt this variable, according to the protocol).

A typical execution by *Bob* (B) in a run β with *Alice* (A), would look like (assuming that Bob is honest):

$$\begin{aligned} \mathsf{ins}(temp) = \langle -(A),\ +(N_B),\ -(X), \\ +(\{(SID_\beta, CNO_2), A, B, X\}_{sh_{BS}}), \\ -(\{(SID_\beta, CNO_3), A, B, N_B\}_{sh_{BS}}) \rangle \end{aligned}$$

where,

$$\begin{aligned} \mathsf{ins}(a) = A,\ \mathsf{ins}(b) = B,\ \mathsf{ins}(s) = S, \mathsf{ins}(n_b) = N_B, \\ \mathsf{ins}(x) = \{(SID_\beta, CNO_1), A, B, N_B\}_{sh_{BS}} (= X). \end{aligned}$$

It is important here to note that, the function ins can be defined to map to a new set of values each time. So that, this captures the aspect of different protocol runs containing different values. It is also interesting to see for which mapping of the ins function, we will be able to obtain an attack on a protocol.

2) Correct-tagging: An encrypted fact is said to be well-tagged if the tag component of the fact has the correct session-id and component number in it, i.e. the encrypted fact is being generated and sent in the expected context. We capture this using the formalizations that we present below.

Let Σ^* represent the strand spaces of all possible protocol runs, where the single protocol run $\alpha \in \Sigma^*$ and,

$$\mathsf{CNo} : \Sigma^* \times \mathbf{ef} \rightarrow \mathbf{cno} \text{ such that,}$$

[3]Function variables are of the form, APPLY $\mathsf{F_n}$ UV where $\mathsf{F_n}$ is any function. eg. $Hash$, PK etc.

$\forall f_1, f_2 \in \mathbf{ef} \cdot f_1 \neq f_2 \Rightarrow \mathsf{CNo}(\alpha, f_1) \neq \mathsf{CNo}(\alpha, f_2)$

Also let, $\mathsf{SId} : \Sigma^* \rightarrow \mathbf{sid}$ such that $\forall \alpha \in \Sigma^* \cdot \ \mathsf{SId}(\alpha) \in \mathbf{sid}$

$\forall \alpha_1, \alpha_2 \in \Sigma^* \cdot \alpha_1 \neq \alpha_2 \Rightarrow \mathsf{SId}(\alpha_1) \neq \mathsf{SId}(\alpha_2)$

These properties ensure unique tags for every subcomponent of any protocol and any run of a protocol.

Using SId and CNo, an ideal tag environment, ω can be formally defined:

$$\omega : \Sigma^* \times \mathbf{ef} \rightarrow \mathbf{Tag} \quad \text{such that,}$$

$\forall \alpha \in \Sigma^*, \forall f \in \mathbf{ef} \cdot \omega(\alpha, f) = (\mathsf{SId}(\alpha), \mathsf{CNo}(\alpha, f))$
So that, for each protocol run α, $\mathsf{SId}(\alpha)$ is the session-id for α. (when the context is understood, we will simply write $\omega(f)$).

Definition 3: Let $f = \{tf'\}_{k'}$ be a fact in a protocol run α; then well-tagged(f) can be inductively defined on all possible facts as follows:

- if $(tf')_2 \in \mathbf{uf}$ then[4] well-tagged(f) $\Leftrightarrow (tf')_1 = \omega(f)$;
- if $(tf')_2 \in \mathbf{ef}$ then, well-tagged(f) *iff* well-tagged($(tf')_2$) $\wedge\ (tf')_1 = \omega(f)$;
- well-tagged(f_1, f_2) $\Leftrightarrow$ (well-tagged(f_1) $\wedge$ well-tagged(f_2))

Note that, well-tagged is a partial function. Therefore, it is undefined for facts that are not encrypted.

Assumption 1: There exists an ideal tag environment, ω, for each set of strands representing a protocol run, that is obtained by instantiating a set of strand templates such that all the facts in the protocol run are well-tagged with respect to ω.

Assumption 2: If the fact f originates on a regular strand, then well-tagged(f).

In other words, we assume that an honest agent always tags an encrypted fact with a correct tag (as defined above). On the other hand, we allow a penetrator to tag a message with any arbitrary tag. If a fact is ill-tagged, this would mean that either the penetrator has "replayed" it from another context or he did not chose to tag with the correct component number and/or the pre-agreed upon session-id.

3) Penetrator Strands: We make two changes in the penetrator strands in this model—firstly, we assume that a penetrator possesses a set of encrypted facts, $\mathbf{E_P}$ that he would have somehow obtained (e.g., by eavesdropping over a network, or obtained in a previous run in which he was a legitimate user), in addition to a set of keys $\mathbf{K_P}$ and texts, $\mathbf{T_P}$. Secondly, we include a replaying (**R**) strand to capture the action of a penetrator replaying an encrypted component.

Definition 4: A penetrator strand is one of the following:

[4]By definition, subscript "2" is a projection that returns the fact component of a tagged fact (section II-A).

M ***Text message*** $\langle +f \rangle$ with $f \in \mathbf{T_P}$.
F ***flushing*** $\langle -f \rangle$.
T ***Tee*** $\langle -f, +f, +f \rangle$.
C ***Concatenation*** $\langle -f_1, -f_2, +f_1 f_2 \rangle$.
S ***Separation*** $\langle -f_1 f_2, +f_1, +f_2 \rangle$.
K ***Key*** $\langle +k \rangle$ with $k \in \mathbf{K_P}$.
E ***Encryption*** $\langle -k, -f, +\{(t, f)\}_k \rangle, k \in \mathbf{K_P}$.
D ***Decryption*** $\langle -k^{-1}, -\{(t, f)\}_k, +f \rangle, k \in \mathbf{K_P}$.
R ***Replaying*** $\langle +f \rangle, f \in \mathbf{E_P}$.

Note that we consider not only replaying of encrypted components, but also replaying of unencrypted components. In fact, we allow the penetrator to replay a message of any type into a message of a different (expected) type. For example, an $Atom$ in place of an $Atom$, an $Atom$ in place of an encrypted component, an encrypted component in place of an encrypted component and so on. For example, sending a message, $f_1.f_2$ with $f_1 \in \mathbf{uf}$ and $f_2 \in \mathbf{ef}$ in place of $f_3 \in \mathbf{uf}$ can be constructed as a sequence of **M** and **R** strands.

However, this does not restrict the generality of the scheme. As we will show in the proof, the scheme also prevents all such type flaw attacks too, due to component numbering. The session-id helps in preventing all type-flaw attacks that occur not only in the same run or a different run of the same protocol, but also those that occur from a different run using a different protocol.

Lemma 1: Every ill-tagged fact originates on an **E** or an **R** strand.

Proof. According to assumption 1, ill-tagged facts do not originate on honest strands. The only possible strands for the origin of a fact are, **M, K, E** and **R** strands. In the case of **M** and **K** strands, there is no tagging. In the case of n **E** strand, it may or may not be well-tagged (we do not restrict the penetrator to put correct tags inside encryptions). That leaves us with **R** strands. Since these involve replaying old messages, they would necessarily be ill-tagged.

III. Transforming Bundles

A. Overview

In this section, we focus on transforming bundles to "well-tagged bundles". We will show that, given a bundle C, we can change all the tags in C so that the resulting bundle has facts, all of which are well-tagged. Since we assume that an honest agent checks all accessible tags in a message that he receives, we do not change the tags inside those messages. On the other hand, receiving an ill-tagged fact that cannot be decrypted should not change the behavior of an honest agent when it is changed to well-tagged. This is the sole concept around which our transformation revolves.

B. The Transformation Function, ψ

The definition below states the required properties of ψ:

Definition 5: Given a bundle C, executed in an ideal tag environment ω, we define $\psi : \mathbf{Fact} \rightarrow \mathbf{Fact}$ to be a transformation function that transforms C as a well-tagged bundle if:

1) ψ preserves unencrypted facts: if $\psi(f) = f$ if $f \in \mathbf{uf}$.
2) ψ returns well-tagged terms: well-tagged($\psi(f)$).

3) ψ is the identify function over well-tagged terms: if well-tagged(f) then $\psi(f) = f$.
4) ψ distributes through encryptions:

$$\psi(\{(t, f)\}_k) = \{(\omega(f), \psi(f))\}_k$$

5) ψ distributes through concatenations:

$$\psi(f1, f2) = (\psi(f1), \psi(f2))$$

6) When ψ is applied to an ill-tagged fact f of C, it produces a fact that has an essentially new tag. i.e. a fact that has a tag not in common with $\psi(f')$ for any other fact f' of C:
$\forall f, f'' \in \mathbf{ef}, \neg$well-tagged$(f) \wedge f' \sqsubset \psi(f) \wedge (f')_2 \neq (f'')_2 \Rightarrow$
$(f')_1 \neq \psi(f'')_1$

This establishes an injectivity property for ψ over facts of C.

The following lemma proves that such a ψ can always be found:

Lemma 2: For any given bundle C in an ideal tag environment ω, it is possible to define some transformation function ψ for C.

Proof: The method we give below gives a recipe for constructing a transformation function ψ defined in definition 5. Let there be a fact f. We shall define how the transformation would be done on all possible smallest sub facts of f before defining it on f itself.

1) If $f \in \mathbf{uf}$, lift the transformation function on f for condition 1: $\psi(f) = f$.
2) If $f = \{(t', f')\}_{k'} \in \mathbf{ef}$ and well-tagged(f), then, define $\psi\{(t', f')\}_{k'} = \{(t', f')\}_{k'}$ for condition 3.
3) If $f = \{(t', f')\}_{k'} \in \mathbf{ef}$, $f' \in \mathbf{uf}$ and $\neg$well-tagged(f), then, define $\psi\{(t', f')\}_{k'} = \{(t'', f')\}_{k'}$ so that $t'' = \omega(f')$ for condition 3.
4) If $f = \{(t', f')\}_{k'} \in \mathbf{ef}$, $f' \in \mathbf{ef}$ and $\neg$well-tagged(f), then, define $\psi\{(t', f')\}_{k'} = \{(t'', \psi(f'))\}_{k'}$ where $t'' = \omega(f')$ for condition 4.
5) If $f = (f1, f2)$, then define $\psi(f) = (\psi(f1), \psi(f2))$ (for condition 5).
6) Since ω generates unique tags, condition 6 is satisfied.

■

C. Regular Strands

If S is a regular strand ($= \mathsf{ins}(temp)$ for some strand template $temp$), then $\psi(S)$ is also a regular strand. Consider, $S' = \mathsf{ins}'(temp)$ and $\mathsf{ins}'(v) = \psi(\mathsf{ins}(v))$ so that, $\mathsf{ins}'(v) = \mathsf{ins}(v)$, $\forall v \in \mathbf{uv}$. The following lemma proves that S' is a regular strand obtained by transforming all the facts in S to be well-tagged, using ψ.

Lemma 3: Let $temp, \psi, \mathsf{ins}, \mathsf{ins}'$ be as above; Then, $\psi(\mathsf{ins}(temp)) = \mathsf{ins}'(temp)$.

Proof: Let v be a variable in $temp$. We will do a case analysis of all the possible forms that v can take in $temp$:

- Case v is an encrypted fact; say, $v = \{(tv', v')\}_{kv'}$; then:

$$\begin{aligned}
\psi(\mathsf{ins}(v)) &= \psi(\{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')}) \\
&= \{(\mathsf{ins_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')} \\
&\quad \langle\text{from condition 3 of definition 5 and since} \\
&\quad \text{well-tagged}(\mathsf{ins}(v)) \text{ from assumption 2}\rangle \\
&= \{(\mathsf{ins'_t}(tv'), \mathsf{ins_v}(v'))\}_{\mathsf{ins_v}(kv')} \\
&\quad \langle\text{by assumption 2, and since} \\
&\quad \text{well-tagged}(\mathsf{ins}'(v)), \mathsf{ins'_t}(tv') = \mathsf{ins_t}(tv')\rangle \\
&= \mathsf{ins}'(v) \\
&\quad \langle\text{from definition of ins and } \mathsf{ins'_v}(v') = \mathsf{ins_v}(v'), \\
&\quad \mathsf{ins'_v}(kv') = \mathsf{ins_v}(kv')\rangle
\end{aligned}$$

- Case v is a pair; say, $v = (v_1, v_2)$; then,

$$\begin{aligned}
\psi(\mathsf{ins}(v)) &= \psi(\mathsf{ins}(v_1), \mathsf{ins}(v_2)) && \langle\text{by definition of ins}\rangle \\
&= (\psi(\mathsf{ins}(v_1)), \psi(\mathsf{ins}(v_2))) && \langle\text{cond. 5 of def. 3}\rangle \\
&= (\mathsf{ins}'(v_1), \mathsf{ins}'(v_2)) && \langle\text{from above results}\rangle \\
&= \mathsf{ins}'(v_1, v_2) \\
&= \mathsf{ins}'(v)
\end{aligned}$$

■

D. Penetrator Strands

In this section we will show how we transform penetrator strands in C to equivalent penetrator strands in the well-tagged bundle (say C'), without any additional penetrator knowledge. We consider each type of penetrator strand defined in definition 4 and define a corresponding strand in C'. In most cases, we will preserve the strand structure, but will retain the same set of facts in every case.

- **M *Text message***: Let $S = \langle +x \rangle$ with $x \in \mathbf{T_P}$. Define $S' = \langle +\psi(x) \rangle$, which is an **M** strand because $\psi(x) = x$ when $x \in \mathbf{uf}$ from condition 1 of definition 5.
- **F *Flushing***: Let $S = \langle -f \rangle$. Define $S' = \langle -\psi(f) \rangle$, which is an **F** strand.
- **T *Tee***: Let $S = \langle -f, +f, +f \rangle$. Define $S' = \langle -\psi(f), +\psi(f), +\psi(f) \rangle$, which is a **T** strand.
- **C *Concatenation***: Let $S = \langle -f_1, -f_2, +f_1 f_2 \rangle$. Define $S' = \langle -\psi(f_1), -\psi(f_2), +\psi(f_1, f_2) \rangle$ which is a valid concatenation strand, because $\psi(f_1, f_2) = (\psi(f_1), \psi(f_2))$ by condition 5 of definition 5.
- **S *Separation***: Let $S = \langle -f_1 f_2, +f_1, +f_2 \rangle$. Define $S' = \langle -\psi(f_1, f_2), +\psi(f_1), +\psi(f_2) \rangle$ which is a valid separation strand, again by condition 5 of definition 5.
- **K *Key***: Let $S = \langle +k \rangle$ with $k \in \mathbf{K_P}$. Define $S' = \langle +\psi(k) \rangle$, which is a **K** strand since $\psi(k) = k$, $\forall k \in \mathbf{uf}$ from condition 1 of definition 5.
- **E *Encryption***: Let $S = \langle -k, -f, +\{(t, f)\}_k \rangle, k \in \mathbf{K_P}$. Define $S' = \langle -\psi(k), -\psi(f), +\psi(\{(t, f)\}_k) \rangle$ which is a valid encryption strand because, $\psi(\{(t, f)\}_k) = \{\omega(f), \psi(f)\}_k$ by condition 4 of definition 5.
- **D *Decryption***: Let $S = \langle -k^{-1}, -\{(t, f)\}_k, +f \rangle, k \in \mathbf{K_P}$. Define $S' = \langle -\psi(k^{-1}), -\psi(\{(t, f)\}_k), +\psi(f) \rangle$,

which is a valid decryption strand because, $\psi(\{(t,f)\}_k) = \{(\omega(f),\psi(f))\}_k$ by condition 4 of definition 5.

- **R *Replaying***: Let $S = \langle +f\rangle, f \in \mathbf{E_P}$. Define $S' = \langle +\psi(f)\rangle$ which is a valid replaying strand because, $\psi(f)$ is merely well-tagged without any additional change in the message.

One special case here concerns when the penetrator receives a well-tagged fact and sends another fact in it's place, either by replaying or by "retagging": i.e. a combination of (a) **F** and **R** strands: $\langle -f, +f'\rangle$ with $f' \in \mathbf{E_P}$ or (b) **D** and **E** strands: $\langle -f, +f'\rangle$ ($\langle -\{(t_1,f_1)\}_{k_1}, -k_1, +f_1, +\{(t_2,f_2)\}_{k_2}, -k_2, +f_2\rangle$), with $k_1, k_2 \in \mathbf{K_P}, f = \{(t_1,f_1)\}_{k_1}$ and $f' = \{(t_2,f_2)\}_{k_2}$.

E. Preserving Unique Origination in Bundles

Using ψ, for every edge $+n \rightarrow -n$ in C, we have created a similar edge $+\psi(n) \rightarrow -\psi(n)$ transforming C into a well-tagged bundle. We have applied ψ only on the tag component of a tagged fact. We did not introduce new facts any where in the bundle. We have changed the strand structure in the special case where a penetrator receives a fact and replays a fact or retags a fact, which is dealt with as explained above.

Lemma 4: Let C and C' be defined as above. If f_0 is a fact, uniquely originating in C, then, f_0 also originates uniquely in C'.

Proof: ψ is effectively applied only on ill-tagged facts. From Lemma 1, they originate only on **E** and **R** strands. By definition, ψ does not change the fact component. Also, by the injectivity property of ψ, there is no duplication of tags. Hence, unique origination is preserved in C'. ■

IV. Proof

A. Overview

In this section we will prove our main result, whenever there is an attack on a protocol using our tagging scheme, there is also an attack on the protocol in the absence of replays. To be precise, we need to prove that,

If there is an attack on a protocol under the tagging scheme, there is also an attack on the protocol when adopting the tagging scheme with all tagged messages correctly tagged.

In other words, an attack on a protocol using the tagging scheme does not revolve around replays. We will consider an attack to mean a failure of authentication at the end of a protocol run. Any other properties can be similarly considered and proven, e.g., non-repudiation, anonymity, fairness. For defining security properties and their violations, we will follow the definitions and terminology given by Heather *et. al* [11]..

B. Authentication

Theorem 1: Let C be a bundle and C' be a well-tagged bundle obtained by transforming C. i.e. $\psi(C) = C'$. If there is a failure of authentication in C, there is also a failure of authentication in C'.

Proof:

Suppose there is a failure of authentication in C as below [11, definition 2]:

1) There is a strand $s1 = \mathsf{ins1}(temp1)$ with C-height at least $h1$. (an honest strand in C with C at a minimal height $h1$).
2) $\forall k \in \mathbf{Keys}\ .\ \mathsf{ins1}(k) \notin \mathbf{K_P}$. (no secret keys are compromised).
3) There is no strand $s2 = \mathsf{ins2}(temp2)$ with C-height at least $h2$ such that $\forall x \in \mathbf{X} \cdot\ \mathsf{ins1}(x) = \mathsf{ins2}(x)$. (there is no matching strand for $s1$ in C, agreeing on some data set $\mathbf{X}$). (This says that since there is no matching honest participant to match the accepted authentication in $s1$, the authentication is bogus.)

We show that there is a corresponding attack in C'. Let, $\mathsf{ins1}'(v) = \psi(\mathsf{ins1}(v))$. Then:

1) There is a strand $s1' = \mathsf{ins1}'(temp1) = \psi(\mathsf{ins1}(temp1))$ with C'-height at least $h1$, corresponding to $s1$, from the way we have constructed the honest strands of C'.
2) $\forall k \in \mathbf{Keys} \cdot\ \mathsf{ins1}'(k) \notin \mathbf{K_P}$, because $\mathsf{ins1}'(k) = \mathsf{ins1}(k)$ for such k.
3) There is no strand $s2' = \mathsf{ins2}'(temp2)$ with C'-height at least $h2$ such that $\forall x \in \mathbf{X}\ .\ \mathsf{ins1}'(x) = \mathsf{ins2}'(x)$.

Suppose there were such an $s2'$; then, by the way we have constructed the honest strands in C', $s2'$ would correspond to some strand $s2'' = \mathsf{ins2}''(temp2)$ with C'-height at least $h2$ such that:

$\forall v \in \mathbf{Var}\ .\ \mathsf{ins2}'(v) = \psi(\mathsf{ins2}''(v))$

But then, we would have for every $x \in \mathbf{X}$:

$$\begin{aligned}\psi(\mathsf{ins1}(x)) &= \mathsf{ins1}'(x)\\ &= \mathsf{ins2}'(x)\\ &= \psi(\mathsf{ins2}''(x))\end{aligned}$$

However, contradicting part 3 of the definition, we would have $\mathsf{ins1}(x) = \mathsf{ins2}''(x)$, because of the injectivity property of ψ (condition 6 of definition 5).

■

C. Example

We illustrate our concepts with the Otway-Rees protocol. Thayer *et. al* establish a number of properties of this protocol in [20, section 7.2]. This protocol can be represented in our scheme in the form of the following templates:

$$\begin{aligned}init \mathrel{\widehat{=}} \langle &+(M, A, B\{t_1, M, N_a, A, B\}_{K_{AS}}),\\ &-(M\{t_3, N_a, K_{AB}\}_{K_{AS}})\rangle\end{aligned}$$

$$\begin{aligned}resp \mathrel{\widehat{=}} \langle &-(M, A, B\{t_1, M, N_a, A, B\}_{K_{AS}}),\\ &+(M, A, B\{t_1, M, N_a, A, B\}_{K_{AS}}\{t_2, N_b, M, A, B\}_{K_{BS}}),\\ &-(M\{t_3, N_a, K_{AB}\}_{K_{AS}}\{t_4, N_b, K_{AB}\}_{K_{BS}}),\\ &+(M, \{t_3, N_a, K_{AB}\}_{K_{AS}})\rangle\end{aligned}$$

$$\begin{aligned}serv \mathrel{\widehat{=}} \langle &-(M, A, B\{t_1, M, N_a, A, B\}_{K_{AS}}\{t_2, M, N_b, A,\\ &B\}_{K_{BS}}), +(M\{t_3, N_a, K_{AB}\}_{K_{AS}}\{t_4, N_b, K_{AB}\}_{K_{BS}})\rangle\end{aligned}$$

Thayer *et. al* also extend these results when this protocol is executed in a "mixed" environment [21], when:

1) $\mathsf{Ticket}(\mathsf{L_0})$ = set of all terms of the form $\{NK'\}_K$ is $unserved$ in Σ. i.e. secondary strands (strands belonging to other runs) do not have entry points to this set.
2) $\mathsf{Request}(\mathsf{L_0})$ = set of all terms of the form $\{NMAB\}_K$ is *strongly unserved* i.e. no term of this set ever originates on a secondary strand.
3) $\mathsf{L_0} \cap \mathbf{K_P} \neq \phi$;

In particular, they establish the following 'initiator's guarantee' under the above assumptions:

If $s \in \Sigma_{init}$ is an initiator strand in a bundle, then there always exist primary strands $s_{resp} \in \Sigma_{resp}$ and $s_{serv} \in \Sigma_{serv}$ which agree on the initiator, responder, and M values. i.e. authentication is guaranteed w.r.t. [11, definition 2]: there is no failure of authentication if $temp1 = init, temp2 = resp, \mathbf{X} = \{A, B, M\}, h_1 = h_2 = 4$ and $\mathbf{Keys} = \{K_{AS}\}$.

We can again use our main result in section IV to show that this protocol achieves this guarantee even when all the three conditions are dropped and the tagging scheme adopted (with the appropriate ins defined on $init$ and $resp$). In addition, component numbering assures us that the protocol remains invulnerable to the type-flaw attack shown in [17] which succeeds due to replay of encrypted components inside the same run.

V. Conclusion

Many times protocols are successfully attacked when an honest agent incorrectly accepts messages, "believing" that they possess some properties (eg. freshness). Whether a technique prevents certain attacks depends upon preventing the honest user from accepting messages that do not have their claimed properties and on the inability of the penetrator to create 'valid' messages having those properties. In this paper we have introduced one such scheme to prevent replay attack and proven that it stops replay attacks. In particular, we have taken an arbitrary bundle, used a transformation on it to change it to a well tagged bundle, where no term represents replays. This was possible because, if an honest agent is willing to accept an ill-tagged fact, it should accept any value in it's place. Attacks were then shown to be existing on both or on none, indirectly proving that the attacks are not based on replays, but on some other mechanism.

VI. Acknowledgements

We would like to thank Hamid R. Arabnia and the anonymous referees of this paper. Also, special thanks go to John Clark, University of York for reviewing our paper and to Gavin Lowe for valuable suggestions and discussions. Finally, we would like to thank Joshua Guttman for some useful comments about the idea and proof.

References

[1] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):553–536, August 1981.

[2] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the Computer Security Foundations Workshop (CSFW97)*, pages 187–191, June 1994.

[3] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *5th International Working Conference on Dependable Computing for Critical Applicaitons*, pages 44–55, September 1995.

[4] T. Aura. Strategies against replay attacks. In *Proceedings of the 10th IEEE Computer Society Foundations Workshop*, pages 59 – 68, Rockport, MA, June 1997. IEEE Computer Society Press.

[5] C.A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. *ESORICS 96, LNCS 1146*, pages 351–364, 1996.

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

[7] C. Mitchell. Limitations of Challenge-Response Entity Authentication. *Electronic Letters*, 25(17):1195–1196, August 1989.

[8] J. Alves-Foss. Multi-Protocol Attacks and the Public Key Infrastructure. In *Proc. National Information System Security Conference*, pages 566–576, October 1998.

[9] J. Kelsey, B. Schneier, and D. Wagner. Protocol Interactions and the Chosen Protocol Attack. In *Proc. Security Protocols - 5th International Workshop*, pages 91–104. LNCS 1361, 1997.

[10] Li Gong. Variation on the Themes of Message Freshness and Replay or the Difficulty of Devising Formal Methods to Analyze Cryptographic Protocols. In *Proceedings of the Computer Security Workshop VI*, pages 131–136, Los Alamitos, California, 1993.

[11] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

[12] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. *Proceedings of the DARPA Information Survivability Conference and Exposition Volume I of II*, January 2000.

[13] U. Carlsen. Cryptographic protocol flaws. In *Proc. IEEE Computer Security Foundations Workshop VII*, pages 192–200. IEEE Computer Press, June 1994.

[14] Joshua D. Guttman and F. Javier THAYER. Protocol Independence through Disjoint Encryption. *13th IEEE Computer Security Foundations Workshop*, pages 24–34, July 2000.

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

[16] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[17] John A. Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0. University of York, Department of Computer Science, November 1997.

[18] P. Karn and W. Simpson. The Photuris session key management protocol. Internet draft: draft-simpson-photuris-17.txt, November 1997.

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

[20] F. J. THAYER Fábrega, J. C. Herzog, and J. D. Guttman. Honest ideals on strand spaces. *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, June 1998.

[21] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed Strand Spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, volume 27(2), pages 10–14. IEEE Computer Society Press, June 1999.