Non-Unifiability of Encrypted Terms in Cryptographic Protocol Specifications

Sreekanth Malladi
Dept of Math and Computer Science
University of Wisconsin-Superior
Superior, WI - 54880
msskanth@hotmail.com

Carol Taylor
Center for Secure and Dependable Systems
University of Idaho
Moscow, ID - 83843
ctaylor@cs.uidaho.edu

Abstract

Non-Unifiability of Encrypted Terms (NUET) has been continuously advocated over the past decade for cryptographic protocol security. In this paper, we extend the definition of NUET to suit newer protocol design features, and newer scenarios. We present implementation guidelines to achieve NUET in the context of the new features and environments. We also prove that previous results established assuming NUET such as type-flaw attack prevention (and thus decidability) are also valid for the newer considerations when our redefined NUET is properly implemented in a protocol.

Keywords – *Cryptographic protocols, Passwords, Constructed keys, Inference systems, Non-unifiability, Type-flaw attacks, Completeness.*

1 Introduction

Suppose we have two encryptions $[X]_K$ and $[Y]_K$ in a protocol specification[1]. Then, placing unique constants (say 1 and 2) inside them, as $[1, X]_K$ and $[2, Y]_K$ will make it impossible to replay one claiming as another. That means, regardless of what is substituted for X and Y, the encryptions can be uniquely identified and they are said to be *non-unifiable*.

This property of *Non-Unifiability of Encrypted Terms* (NUET) in protocols was suggested as a prudent engineering practice for protocol design in the classical work by Abadi and Needham [AN94, Principle 10]. Following Abadi and Needham, NUET was advocated or used at many places in the literature (e.g. [Low96b, Aur97, HLS00, GT00b, RS03, BP04, RS05]). The most significant of these papers is recently by Ramanujam and Suresh in which they prove that NUET ensures decidability of protocol security for trace-based security properties, and prevents type-flaw attacks [RS05].

However, all the above works including [RS03] have considered basic cryptographic protocol features such as atomic keys and assume perfect protocol run environments where encryptions are unbreakable and unforgeable. "Real-world" protocols such as SSL and SET employ newer features such as non-atomic keys and weak human passwords [FKK, MV97]. Also, with the knowledge of newer cryptographic vulnerabilities, attackers are endowed with new powers to break encryptions and create fake encryptions.

[1] $[t]_k$ denotes t encrypted with k.

These are the issues we consider in this paper: we extend the definition of NUET to suit the newer features and environments. We suggest implementation guidelines for protocol design to satisfy the extended definition. We prove that the so satisfied requirement is sufficient to prevent type-flaw attacks on cryptographic protocols and consequently, ensures the decidability of protocol security.

This paper is structured as follows: In **Section 2**, we expand on the above points and present a background. In **Section 3**, we present our framework to be used in subsequent sections and redefine NUET. In **Section 4**, we prove our main result on type-flaw attacks. In **Section 5**, we provide implementation guidelines to achieve the new NUET in the standard model, and in a model that allows weak human passwords. In the same section we also present a study of the effect of NUET in the presence of non-standard attacker inference rules. We sum up with a **Conclusion** that includes a discussion of non-free term algebras in protocol specifications and how they could influence the results established thus far.

2 Background

Consider the Woo and Lam π_1 protocol [WL94]:

Msg 1. $A \rightarrow B : A$
Msg 2. $B \rightarrow A : N_B$
Msg 3. $A \rightarrow B : [A, B, N_B]^{\leftrightarrow}_{sh(A,S)}$
Msg 4. $B \rightarrow S : [A, B, [A, B, N_B]^{\leftrightarrow}_{sh(A,S)}]^{\leftrightarrow}_{sh(B,S)}$
Msg 5. $S \rightarrow B : [A, B, N_B]^{\leftrightarrow}_{sh(B,S)}$

(*Notation*: We use uppercase letters for variables and lowercase for ground terms; $[X_1, \ldots, X_n]$ represents concatenation of terms X_1 to X_n; $[X]^{\leftrightarrow}_Y$ represents X encrypted with Y using a symmetric keying algorithm. $sh(X, Y)$ represents a shared-key between agents X and Y.)

We will abbreviate the Woo and Lam π_1 protocol to WL π_1 and use it as a running example in this paper. This protocol belongs to a special class of protocols that contain message receptions by participants that they cannot decrypt (in Msg 3 above, B is not expected to decrypt the message, but to simply use it to construct Msg 4). Ramanujam and Suresh in [RS05] explicitly state that their results are *not* valid for this class of protocols. They in fact exemplify the invalidity on the WL π_1 protocol. To demonstrate the strength of our results, we use this *very* example in our paper to show that our results are also applicable for these class of protocols.

Below we explain two important features in "real-world" protocols that fall outside the scope of Ramanujam *et al.*'s (and many other's such as [HLS00]'s) results.

2.1 Constructed keys, and Passwords

Ramanujam *et al.*'s results state that their results are only applicable to protocols that employ atomic keys, but not composed keys. *Composed* (or *Constructed* or *non-atomic*) keys are keys that are formed using shared secrets and other data exchanged as part of the protocol itself. For example, $[a, [n_b]^{\leftrightarrow}_{k'}]$ can be used as a key. Many times, the "authentication force" in terms encrypted with constructed keys lies not just in the fields inside the encryptions, but also in the terms inside the keys.

In **Section 3**, we redefine NUET by requiring constants to be placed inside keys as well (if it is a constructed key). Our protocol model is based on Millen-Shmatikov's constraint solving technique that explicitly employs constructed keys. Hence unlike previous results, they are valid even in presence of constructed keys. In **Section 5** we present an example protocol that employs constructed keys and show how it can be attacked if our newly redefined NUT is not followed.

Messages should also not be tagged with constants when a protocol is being implemented with a weak password [CMAFE03, Low04]. For example, consider the following message:

$$[21, n_a]^{\leftrightarrow}_{passwd(a,s)}$$

Now if an attacker guesses $passwd(a, s)$, he can decrypt the message using that guess and check the tag "21" to verify that guess. He does not even need to know n_a! Thus, tags facilitate such direct guessing attacks.

In **Section 5** we show that NUET can also be adopted by rearranging terms, without using artificial constants that aid guessing attacks.

2.2 Non-standard attacker inference rules

All published works that suggested NUET only considered standard attacker inference rules that adopt the perfect encryption assumption. Perfect encryption assumption implies that an attacker can neither break existing encryptions nor create fake encryptions.

However, nothing is perfect in this world and encryption is no exception. For example, the rule

$$[m, n]^{\leftrightarrow}_k \vdash [m]^{\leftrightarrow}_k \text{ (} \vdash \text{ denotes the relation } \textit{infer} \text{).}$$

holds when using CBC (Cipher Block Chaining) method for symmetric key encryption provided m is a whole number of blocks [Boy90, SG92]. Here, an attacker is able to create $[m]^{\leftrightarrow}_k$ without actually possessing m or k. In **Section 5** we show how an attacker can attack the WL π_1 protocol using this rule even when the protocol implements NUET as suggested in previous works. We then present alternative ways to achieve NUET depending on the inference rules in effect to avoid such attacks.

3 The Framework

Our formal protocol model is based on the constraint solving model of Millen-Shmatikov [MS01]. The model uses symbolic terms which is ideal for our results which involve unification. We start off by introducing the term algebra, protocol model, NUET, and constraint solving.

3.1 Term Algebra

Messages are constructed using operators that are applied freely on terms.

Definition 1. A *term* is one of the following:

Variable	We assume that every variable appearing in a protocol specification and thus part of a term, can be attributed to a particular type. We denote the type of a variable V as $type(V)$.
Constant	Atomic values and numbers. E.g., name of the attacker ϵ, nonce n_a, numbers 1, 2, 3 etc.
$[t_1, t_2]$	Pair of terms t_1 and t_2, using the pairing operator. We will write $[t_1, t_2, t_3]$ for $[t_1, [t_2, t_3]]$, to avoid notational clutter.
$pk(X)$	Public-key of agent X. It is assumed that the corresponding secret key is not sent as part of any message in the protocol. Therefore, the attacker can only decrypt messages encrypted with his public-key.

$h(t)$	Hash of t, where h is a one-way hash function.
$[t]_k^{\rightarrow}$	t encrypted with k using an asymmetric encryption algorithm. k is any term, not necessarily a public-key.
$[t]_k^{\leftrightarrow}$	t encrypted with k using a symmetric encryption algorithm. k can be any term.
$sig_k(t)$	Public-key signature of term t validated using key k. According to assumptions, private keys are never revealed, so the attacker can only construct signatures with his own private key.

A *ground term* is any term that does not contain a variable. We use uppercase letters for variables and lowercase letters for constants and ground terms. Note that every protocol can possibly use special variables, such as *Msg* 3 in the `WL` π_1 protocol. Since B cannot decrypt the term, this is represented as a variable of a special type (for the proofs and results in this paper, it suffices to simply assume that such variables have a special type. We do not need to formalize those types).

We assume that the attacker possesses an initial set of terms such as his own public-key, $pk(\epsilon)$, the names of some principals, nonces he can generate and so on.

When we refer to encryption in general (not specifically asymmetric or symmetric), we will ignore the superscripts '$\leftrightarrow$' and '$\rightarrow$'. So, t encrypted with k will be written simply as $[t]_k$. We now define the *subterm* relation on terms:

Definition 2. Let t, t_1, t_2 and k be terms. Then, the *subterm* relation denoted as $\sqsubset$ is defined as the smallest relation on terms such that,

1. $t \sqsubset t$;
2. $t \sqsubset [t_1, t_2]$ if $t \sqsubset t_1 \vee t \sqsubset t_2$;
3. $t \sqsubset [t']_k$ if $t \sqsubset t' \vee t \sqsubset k$;
4. $t \sqsubset pk(t')$ if $t \sqsubset t'$;
5. $t \sqsubset h(t')$ if $t \sqsubset t'$;
6. $t \sqsubset sig_k(t')$ if $t \sqsubset t' \vee t \sqsubset k$;

We will write $t \in T \;\; if \;\; t \sqsubset t' \wedge t' \in T$.

In point 3 above, $t \sqsubset k$ might sound unusual, but it is crucial for our proofs since our term algebra allows constructed keys (See Lemma 3).

3.1.1 Unification

We now talk about unification.

Definition 3. A *substitution* σ is denoted as $\sigma = \{x_1/V_1, \ldots, x_n/V_n\}$ such that σt represents the substitution of x for all occurrences of V in a term t whenever $x/V \in \sigma$. Further, $\forall x/V \in \sigma$, V is a variable and x is either a variable, ground term or a constant.

Definition 4. Let t, t' be two terms. Then t, t' are said to be *unifiable*, denoted $t \approx t'$ (read t *unifies with* t') iff $\sigma t = \sigma t'$ for some substitution σ. σ is called the *unifier*. A unifier ρ is called the *most general unifier* (mgu) if every other unifier σ, is an extension of ρ; specifically $\sigma = \rho\sigma$.

Further,

$$
\begin{array}{lll}
[t_1, t_2] & \approx [t'_1, t'_2] & \text{if } t_1 \approx t'_1 \wedge t_2 \approx t'_2, \\
pk(t) & \approx t' & \text{if } (t' = pk(t'') \wedge t \approx t'') \vee (t' \text{ is a variable}), \\
[t]_k & \approx t' & \text{if } (t' = [t'']_{k''} \wedge t \approx t'' \wedge k \approx k'') \vee (t' \text{ is a variable}), \\
h(t) & \approx t' & \text{if } (t' = h(t'') \wedge t \approx t'') \vee (t' \text{ is a variable}), \\
sig_k(t) & \approx t' & \text{if } (t' = sig_{k''}(t'') \wedge t \approx t'' \wedge k \approx k'') \vee (t' \text{ is a variable}).
\end{array}
$$

3.2 Protocol Model

In this section we introduce the protocol model. The basic model is derived from the idea of semi-bundles introduced in [Son99] with the underlying framework derived from the strand space model [THG98].

Definition 5. A *strand* is a sequence of communication points called *nodes* denoted as $\langle n_1, \ldots, n_k \rangle$. A node consists of a *term* and a *sign*. The term of a node n (denoted as $\mathrm{term}(n)$) can be any term constructed from the term algebra. The sign of a node (denoted as $\mathrm{sign}(n)$) is either '$+$' or '$-$'. Thus, strand $s = \langle n_1, \ldots, n_k \rangle$ can be represented as $\langle \pm t_1, \ldots, \pm t_k \rangle$ where t_i is the term on the node n_i for $i = 1$ to k.

Roles in protocols are modeled as *parametric strands*. For example, role 'B' in the WL π_1 protocol can be represented in the parametric strand form as:

$$role_B = \langle -A, +N_B, -X, +[A, [B, X]]^{\leftrightarrow}_{sh(B,S)}, -[A, [B, N_B]]^{\leftrightarrow}_{sh(B,S)} \rangle.$$

A parametric strand is instantiated into a *semi-strand* using a substitution for some of the variables in the parametric strand, by an honest agent. These substitutions indicate that the agent knows the value for that term. For example, above, $role_B$ can be instantiated by an agent 'b' using

$$\sigma^h_b = \{n_b/N_B, b/B, s/S\}$$
and
$$\sigma^h_b role_B = \langle -A, +n_b, -X, +[A, [b, X]]^{\leftrightarrow}_{sh(b,s)}, -[A, [b, n_b]]^{\leftrightarrow}_{sh(b,s)} \rangle.$$

A typical honest substitution is denoted σ^h_x where x is the agent's identity. A set of semi-strands is a *semi-bundle*; the set of all honest substitutions for a semi-bundle is denoted σ^H_S. We will denote the set of encryptions in a semi-bundle S as $Encr(S)$, hashes as $Hash(S)$ and signatures as $Sig(S)$. We will tend to ignore the subscripts in σ^h_x and σ^H_S when we refer to substitutions in general, not of a specific strand or semi-bundle respectively.

We will say that a substitution is "well-typed" if variables in all the elements of the substitution are instantiated with ground terms of their corresponding types; for example, a nonce variable N_A being instantiated with a nonce value 'n_a' where $n_a \in Nonce$. More generally,

$$\text{well-typed}(\sigma) \;\text{ iff }\; \forall x/V \in \sigma \,.\, x \;\text{ is ground } \;\Rightarrow x \in type(V).$$

We assume that an honest agent always follows a well-typed substitution:

Assumption 1.

$$\forall \sigma^h \in \sigma^H \,.\, \text{well-typed}(\sigma^h)$$

3.3 NUET — Non-Unifiability of Encrypted Terms

We now explain our notion of Non-Unifiability of Encrypted Terms and define it formally. The requirement is quite simple. In a given protocol specification (set of parametric strands), *no two encrypted*[2] *terms should be unifiable unless they are textually equal*. In other words, there is no common substitution for variables in two textually distinct terms, that will make the terms equal. Formally,

Definition 6. Let S be a semi-bundle such that $S = \sigma^h St$ where St is a set of parametric strands. Then St is NUET-*satisfying* and S is NUET-*complying* iff

$$\forall t_1, t_2 \in St \,.\, ((t_1 = [t'_1]_{k_1} \wedge t_2 = [t'_2]_{k_2}) \vee (t_1 = h(t'_1) \wedge t_2 = h(t'_2)) \vee (t_1 = sig_{k_1}(t'_1) \wedge t_2 = sig_{k_1}(t'_2))$$

$$\wedge (t_1 \neq t_2) \Rightarrow t_1 \not\approx t_2. \tag{1}$$

$$\forall t \in St \,.\, t = [t']^{\rightarrow}_{k} \Rightarrow k \text{ is not a variable.} \tag{2}$$

Condition 1 is sufficient to prevent type-flaw attacks in presence of atomic keys. In presence of constructed keys, we would need the additional Condition 2 (See Theorem 1, Case 3).

NUET is quite easy to achieve in protocols. Since the requirement is that terms should be non-unifiable unless they are textually equal, it is possible that this requirement is naturally achieved by a protocol without any special modifications. However, even in protocols that do not satisfy NUET naturally, they can be modified slightly to satisfy NUET. A simple technique is to use distinct constants (such as numbers) inside textually distinct terms of parametric strands. This would ensure that the protocol will be NUET-*satisfying*. This is explained in more detail with examples in **Section 5**.

3.4 Attacker Capabilities

Attacker capabilities are modeled as operators on sets of terms to analyze, synthesize or hide terms. Let Φ be the set of attacker term set operators. Φ captures standard inference rules, namely pairing, splitting, encryption, decryption as well as hashing, and signatures.

Definition 7. The set of attacker term set operators, Φ is defined as $\Phi = \{\Phi_{analz}, \Phi_{synthz}\}$ with $\Phi_{analz} = \{\phi_{split}, \phi_{pdec}, \phi_{sdec}\}$, $\Phi_{synthz} = \{\phi_{pair}, \phi_{penc}, \phi_{senc}, \phi_{hash}, \phi_{sig}\}$, and,

$$\begin{array}{ll}
\phi_{split}(S) = S \cup x \cup y \text{ if } [x,y] \in S, & \phi_{pair}(S) = S \cup [x,y] \text{ if } x,y \in S, \\
\phi_{pdec}(S) = S \cup x \quad \text{ if } [x]^{\rightarrow}_{pk(\epsilon)} \in S, & \phi_{penc}(S) = S \cup [x]^{\rightarrow}_{y} \text{ if } x,y \in S, \\
\phi_{sdec}(S) = S \cup x \quad \text{ if } [x]^{\leftrightarrow}_{y}, y \in S, & \phi_{senc}(S) = S \cup [x]^{\leftrightarrow}_{y} \text{ if } x,y \in S, \\
\phi_{hash}(S) = S \cup h(x) \text{ if } x \in S, & \phi_{sig}(S) = S \cup sig_{pk(\epsilon)}(x) \text{ if } x \in S.
\end{array}$$

The question as to whether a term t can be generated by the attacker using Φ is formulated using the *fake* operation, $\mathcal{F}$:

Definition 8. The *fake* operation $\mathcal{F}$ is defined such that $\mathcal{F}(T)$ is the smallest set formed from a set of ground terms T, closed under Φ. Further, $t \in \mathcal{F}(T)$ iff there exists $\phi_1, \ldots, \phi_n \in \Phi$ such that $t \in \phi_n(\ldots(\phi_1(T))\ldots)$.

[2]Here, by "encryption" we mean any cryptographic function application, not necessarily symmetric-key and public-key encryptions.

3.5 Constraints and Satisfiability

As stated before, a semi-bundle is a collection of partially instantiated strands. All the nodes in a semi-bundle can be merged into an interleaving. Since a semi-bundle has finite number of nodes, there can be finitely many such interleavings from a semi-bundle.

A *constraint sequence* can be drawn from each interleaving such that every element in a sequence represents a *constraint* stating that a message expected by an honest agent has to be computed by the attacker using his initial knowledge, messages sent by honest agents until that point and his capabilities.

A typical constraint is represented as $m : T$ denoting that $\sigma^\epsilon m$ should be obtainable using $\mathcal{F}$ on T where σ^ϵ is the attacker's substitution. i.e., $m : T \Leftrightarrow \sigma^\epsilon m \in \mathcal{F}(\sigma^\epsilon T)$.

A constraint sequence is *satisfiable* if there is a consistent substitution of ground terms to variables such that the attacker can generate every message on every constraint in the sequence. A semi-bundle is said to be *completable* to a *bundle* if a constraint sequence drawn from one of the interleavings of the semi-bundle is satisfiable using some consistent substitution.

Definition 9. Let C be a constraint sequence and σ^ϵ be the attacker's substitution. Then, we say that σ^ϵ *satisfies* C, or $\sigma^\epsilon \vdash C$, iff $\forall m : T \in C \centerdot \sigma^\epsilon m \in \mathcal{F}(\sigma^\epsilon T)$.

Definition 10. If $\forall m : T \in C \centerdot m$ is a variable, then C is called a *simple* constraint sequence.

A simple constraint sequence is always solvable as long as the attacker has at least one constant. We, assume that the attacker possesses at least one ground term for every variable occurring in a protocol, with the same type. Therefore, the attacker can always do a well-typed substitution for such a sequence:

Assumption 2. Let C be a simple constraint sequence. Then, $\exists \sigma \centerdot \sigma \vdash C \ \wedge$ well-typed(σ).

3.6 Constraint Satisfaction

We now introduce the *constraint satisfaction procedure* by Millen-Shmatikov [MS01] to determine if a constraint sequence is satisfiable. The algorithm for the procedure (denoted **P**) is given in Appendix A. The algorithm uses a set of *reduction rules*, R.

A reduction rule is always applied by **P** to the first constraint of a constraint sequence that does not have a variable on the left side, called the *active constraint* of **P**. An application of a rule $r \in R$ on a constraint sequence C with the current substitution σ^ϵ is represented as,

$$r(C; \sigma^\epsilon) = \langle C'; \sigma^{\epsilon'} \rangle,$$

where C' is the new constraint sequence and $\sigma^{\epsilon'}$, the new substitution. A list of all reduction rules in R considered by **P** is in the Appendix[3]. **P** applies rule $elim$ to eliminate all stand-alone variables on the right side, before applying any other rule.

Definition 11. Let C be a constraint sequence and $C_{\mathbf{P}} = \{C_1, \ldots, C_n\}$, a set of constraint sequences generated by $\mathbf{P}$ from C. Then C is *solvable* iff there exists a sequence of reduction rules that reduce the sequence such that the resultant sequence is a simple constraint sequence. Formally,

$$\sigma^\epsilon \vdash C \textit{ iff } \exists r^n, \ldots, r^1 \in R \centerdot r^n(\ldots(r^1(C_1; \sigma_1^\epsilon))\ldots) = \langle C_n; \sigma_n^\epsilon \rangle \wedge \exists \tau \centerdot \tau \vdash C_n \wedge \sigma^\epsilon = \sigma_n^\epsilon \cup \tau,$$

where $\sigma_1^\epsilon = \{\ \}$ and C_n is a simple constraint sequence.

[3]Note that we denote the application of rules in a functional notation, although in reality they are only relations.

Note that the above definition assumes that solving a constraint sequence C using **P** is equivalent to testing if $\sigma^\epsilon m \in \mathcal{F}(\sigma^\epsilon T), \forall m : T \in C$. This has been formally justified by Millen-Shmatikov in [MS01] who proved that **P** is terminating, sound and complete.

Definition 12. Two constraint sequences C and C' are *equivalent* iff whenever $\sigma^\epsilon \vdash C$, $\sigma^\epsilon \vdash C'$. A constraint sequence is *normal* for **P** iff the active constraint is not of the form $[t_1, t_2] \ : \ T$ for some t_1 and t_2.

Lemma 1. Every constraint sequence has an equivalent normal constraint sequence.

Proof. Let C be a constraint sequence. Let $m \ : \ T \in C$ be the active constraint of **P** for C such that $m = [t_1, t_2]$ for some t_1, t_2 and let σ^ϵ be the attacker's substitution. Then apply r^{pair} on C such that $r^{pair}(C; \sigma^\epsilon) = \langle C'; \sigma^\epsilon \rangle$. Then, because r^{pair} is sound [MS01, A.3], $\sigma^\epsilon \vdash C \Rightarrow \sigma^\epsilon \vdash C'$. Therefore, C and C' are equivalent. Apply r^{pair} as many times as possible on C to obtain a normal constraint sequence C''. By induction, C and C'' are equivalent.

□

3.7 Security properties and Attacks

An attack is a violation of a security property, not known to exist in a normal run of the protocol. We assume that a security property can be "embedded" into a semi-bundle and testing for violation of the property is equivalent to finding out if a constraint sequence from the semi-bundle is satisfiable. This is particularly true for trace properties such as secrecy and authentication. For example, violation of secrecy can be tested by embedding a "test strand" containing a node with a secret term (unencrypted) and a '$+$' sign into a semi-bundle and then testing the semi-bundle for solvability.

Our main result is general and is valid for any trace property that can be embedded into a semi-bundle as above since it is based around the general definition of solvability of such a "test" semi-bundle.

We however define a *type-flaw attack* using the property of well-typed introduced in Section 2.2. We will say that a type-flaw attack exists on a semi-bundle if a constraint sequence from the semi-bundle was solved with an ill-typed substitution, and the constraint sequence cannot possibly be solved using a well-typed substitution. Formally,

Definition 13. Let C be a constraint sequence from a semi-bundle S and σ^ϵ be the attacker's substitution. We say that a *type-flaw attack* is possible on S iff

$$(\sigma^\epsilon \vdash C \wedge \neg\text{well-typed}(\sigma^\epsilon)) \wedge (\nexists \sigma^{\epsilon'} \centerdot \sigma^{\epsilon'} \vdash C \wedge \text{well-typed}(\sigma^{\epsilon'})).$$

4 Main Result

We now prove that if a semi-bundle is NUT-*complying* then there are no type-flaw attacks on the semi-bundle.

Proof Outline. Our proof utilizes the reduction procedure **P**. We consider a possible constraint sequence C that can be generated from the given semi-bundle S. We then examine the reduction rules that could be possibly applied to satisfy C and which affect the current attacker substitution. We show that when S is NUT-*complying*, the application of those rules implies that the attacker substitution remains well-typed, if it was initially well-typed. This implies that if C was solved using an ill-typed substitution, the ill-typing only occurred after all the reduction rules were applied on C by **P**.

The resultant constraint sequence, when no more rules can be applied, is a simple constraint sequence. According to Assumption 2, such a sequence can always be satisfied by a well-typed attacker substitution.

Therefore, whenever C was satisfied with an ill-typed substitution, it can also be satisfied using a well-typed substitution. This proves that there is no type-flaw attack on S.

There are two other crucial parts in our proof. Firstly, we prove that whenever two (non-variable) terms of a NUT-*complying* semi-bundle are unified, the unifier is a well-typed substitution. This is rather straightforward since terms in a NUT-*complying* semi-bundle are unifiable only when obtained from the same corresponding term in their parametric strands. Further, both of those terms would be instantiated with honest instantiations, which are always well-typed from Assumption 1. Therefore, a unifier for those would be well-typed. This is proved in the following Lemma:

Lemma 2.

Let $S = \sigma_S^H St$ such that St is NUT-*satisfying* and S is NUT-*complying*. Then $\forall t, t' \in Encr(S) \cup Hash(S) \cup Sig(S)$, $\exists \sigma \centerdot \sigma t = \sigma t' \Rightarrow$ well-typed(σ).

Proof. We prove through induction.

1. Since S is NUT-*complying*, let $t_1 = \sigma_1^h X$ and $t_2 = \sigma_2^h X$. Then, $\exists \sigma \centerdot \sigma t_1 = \sigma t_2 \Rightarrow \exists \sigma \centerdot \sigma\sigma_1^h X = \sigma\sigma_2^h X$. From Assumption 1, $\forall \sigma^h \in \sigma^H \centerdot$ well-typed(σ^h). Let $\sigma_1^h X$ be ground and $\sigma_2^h X = X$. Then $\sigma\sigma_1^h X = \sigma_1^h X$, and $\sigma\sigma_2^h X = \sigma X$. Therefore, $\sigma \subset \sigma_1^h$ and well-typed(σ). Similarly, if $\sigma_2^h X$ is ground and $\sigma_1^h X = X$, then well-typed(σ).

2. Let $t_1 = \sigma_1^h[X, Y] \wedge t_2 = \sigma_2^h[X, Y]$. Then, $\exists \sigma \centerdot \sigma t_1 = \sigma t_2$
   $\Rightarrow \exists \sigma \centerdot (\sigma\sigma_1^h X = \sigma\sigma_2^h X) \wedge (\sigma\sigma_1^h Y = \sigma\sigma_2^h Y)$ (from def 4)
   $\Rightarrow \exists \sigma \centerdot (\sigma\sigma_1^h X = \sigma\sigma_2^h X) \wedge \exists \sigma \centerdot (\sigma\sigma_1^h Y = \sigma\sigma_2^h Y)$
   $\Rightarrow$ well-typed$(\sigma) \wedge$ well-typed(σ) (From (1))
   $\Rightarrow$ well-typed(σ).

3. Let $t_1 = [t_1']_{k_1} \wedge t_2 = [t_2']_{k_2}$. Then, $\exists \sigma \centerdot \sigma t_1 = \sigma t_2$
   $\Rightarrow \exists \sigma \centerdot (\sigma t_1' = \sigma t_2') \wedge \exists \sigma \centerdot (\sigma k_1 = \sigma k_2)$ (from def 4)
   $\Rightarrow$ well-typed$(\sigma) \wedge$ well-typed(σ) (From (1))
   $\Rightarrow$ well-typed(σ).

4. Let $t_1 = h(t_1') \wedge t_2 = h(t_2')$. Then, $\exists \sigma \centerdot \sigma t_1 = \sigma t_2$
   $\Rightarrow \exists \sigma \centerdot (\sigma t_1' = \sigma t_2')$ (from def 4)
   $\Rightarrow$ well-typed(σ) (From (1))

5. Let $t_1 = sig_{k_1}(t_1') \wedge t_2 = sig_{k_2}(t_2')$. Then, $\exists \sigma \centerdot \sigma t_1 = \sigma t_2$
   $\Rightarrow \exists \sigma \centerdot (\sigma t_1' = \sigma t_2') \wedge \exists \sigma \centerdot (\sigma k_1 = \sigma k_2)$ (from def 4)
   $\Rightarrow$ well-typed$(\sigma) \wedge$ well-typed(σ) (From (1))
   $\Rightarrow$ well-typed(σ).

□

Secondly, note that NUT guarantees that terms are non-unifiable *only* when they belong to the semi-bundle (or subterms of the semi-bundle). Therefore, the above result on the unifier remaining well-typed is true only under the condition that attacker inference rules (here the reduction rules) always decompose a term set to add subterms of the existing terms. For example, a reduction rule $[t_1, t_2]_k \vdash [t_1]_k$. would reduce the right side of the active constraint T, by adding a term $[t_1]_k$ if T contained some $[t_1, t_2]_k$. Here, $[t_1]_k \not\subset [t_1, t_2]_k$. Therefore, if $[t_1]_k$ is unified with the left side of the constraint, the resultant substitution is not necessarily well-typed, since $[t_1]_k$ does not belong to the semi-bundle and was not created by an honest agent. Hence, our scheme is

valid only for those inference rules which reduce a term set to obtain subterms. i.e. reduce a term set T to T' so that, $\forall t \in T'$, $t \sqsubset t''$ for some $t'' \in S$.

The following Lemma states that all the reduction rules in R reduce the term set of a constraint so that the resultant term set contains only subterms of the original term set:

Lemma 3.

$$\forall r \in R \,.\, r(C_<, m : T, C_>; \sigma) = \langle C_<, m : T', C_>; \sigma\rangle \wedge T \subset T' \Rightarrow \forall t \in T' \setminus T, \exists t' \in T \,.\, t \sqsubset t'.$$

Proof. The only rules where $T \subset T'$ are r^{split}, r^{pdec}, and r^{sdec}.

1. In the case of r^{split} where $[t_1, t_2] \in T$, $T' \setminus T = \{t_1, t_2\}$ and $t_1 \sqsubset [t_1, t_2]$ and $t_2 \sqsubset [t_1, t_2]$;
2. For r^{pdec}, $T' \setminus T = t$, where $t \sqsubset [t]_{pk(\epsilon)}$ and $[t]_{pk(\epsilon)} \in T$;
3. Similarly, for r^{sdec}, we have two constraints replacing the active constraint, $m : T \cup [t]_k^{\leftrightarrow}$. $k : T$ and $m : T \cup t \cup k$. For the first constraint, $T' \setminus T = \{\}$. For the second constraint, $T' \setminus T = \{t, k\}$ where $t \sqsubset [t]_k$ and $k \sqsubset [t]_k$ [4].

□

We now state and prove the main result.

Theorem 1. Let S be a semi-bundle. If S is NUT-*complying*, there cannot be a type-flaw attack on S.

Proof. Let C be a constraint sequence from S and σ^ϵ be the attacker's substitution. If S is NUT-*complying* then, we want to prove that,

$$\sigma^\epsilon \vdash C \Rightarrow (\exists \sigma^{\epsilon'} \,.\, \sigma^{\epsilon'} \vdash C \wedge \text{well-typed}(\sigma^{\epsilon'})).$$

Firstly, since $\sigma^\epsilon \vdash C$, by Definition 11, let $r^n(\ldots(r^1(C_1; \sigma_1^\epsilon))\ldots) = \langle C_n; \sigma_n^\epsilon\rangle$, for some $r^n, \ldots, r^1 \in R$, and substitution ρ where $\rho \vdash C_n$, with $\sigma^\epsilon = \sigma_n^\epsilon \cup \rho$.

We first prove that σ_n^ϵ is well-typed. We show that $\forall r \in R \,.\, r(C; \sigma) = \langle C'; \sigma'\rangle$, well-typed$(\sigma) \Rightarrow$ well-typed(σ'). Since $\sigma_1^\epsilon = \{\ \}$ is well-typed, it follows inductively that σ_n^ϵ is well-typed.

Case 1 $\forall r \in \{r^{pair}, r^{hash}, r^{penc}, r^{senc}, r^{sig}, r^{split}, r^{pdec}, r^{sdec}\} \,.\, r(C; \sigma^\epsilon) = \langle C'; \sigma^{\epsilon'}\rangle$, with $\sigma^\epsilon = \sigma^{\epsilon'}$. Therefore, well-typed$(\sigma^\epsilon) \Rightarrow$ well-typed$(\sigma^{\epsilon'})$.

Case 2 When $r = r^{un}$, let $r^{un}(C_<, m : T, C_>; \sigma^\epsilon) = \langle \tau C_<, \tau C_>; \tau \cup \sigma^\epsilon\rangle$ where $\tau =$ mgu$(m, t), t \in T$; A reduction rule is always applied to the active constraint. By the definition of active constraint, m is not a variable. Also, **P** eliminates all stand-alone variables from T using rule (*elim*) before applying any other rules in R. Therefore, t is not a variable. Assume without loss of generality that C is a normal constraint sequence (Lemma 1). Therefore, from Definition 12, m is not a pair. From Lemma 3, $\forall t \in T \,.\, t \sqsubset t'$, for some $t' \in S$ and since S is NUT-*complying*, $\tau m = \tau t \Rightarrow$ well-typed(τ) from Lemma 2. Now since, $\sigma^{\epsilon'} = \sigma^\epsilon \cup \tau$, well-typed$(\sigma^\epsilon)$ and well-typed(τ), it follows that well-typed$(\sigma^\epsilon) \Rightarrow$ well-typed$(\sigma^{\epsilon'})$.

Case 3 When $r = r^{ksub}$, let $r^{ksub}(C_<, m : [t]_k^{\rightarrow} \cup T, C_>; \sigma^\epsilon) = \langle \tau C_<, \tau m : \tau[t]_k^{\rightarrow} \cup \tau T, \tau C_<; \tau \cup \sigma^\epsilon\rangle$. Here $\tau =$ mgu$(k, pk(\epsilon))$. From Condition 2 of NUT, k is not a variable. Therefore, $k \approx pk(\epsilon) \Rightarrow \exists X \,.\, k = pk(X)$ for some X. From the term algebra, $\forall pk(X)$, X is a principal name. Since ϵ is a principal name (the name of the attacker), well-typed(τ). Hence, well-typed$(\sigma^\epsilon) \Rightarrow$ well-typed$(\sigma^{\epsilon'})$.

[4] Recall that $k \sqsubset [t]_k$ by Definition 2.

Finally, since C_n is a simple constraint sequence, from Assumption 2, $\exists \rho' \centerdot$ well-typed$(\rho') \wedge \rho' \vdash C_n$. Let $\sigma^{\epsilon'} = \sigma_n^{\epsilon} \cup \rho'$. From above, $\exists r^1, \ldots, r^n \in R \centerdot r^n(\ldots r^1(C_1; \sigma_1^{\epsilon}) \ldots) = (C_n; \sigma_n^{\epsilon})$ and $\rho' \vdash C_n$. Therefore, from Definition 11, $\sigma^{\epsilon'} \vdash C$. Further, since well-typed(σ_n^{ϵ}) and well-typed(ρ'), it follows that well-typed$(\sigma^{\epsilon'})$. Therefore, $\sigma^{\epsilon} \vdash C \Rightarrow (\exists \sigma^{\epsilon'} \centerdot \sigma^{\epsilon'} \vdash C \wedge$ well-typed$(\sigma^{\epsilon'}))$.

Hence, the result.

□

5 Implementing NUET

As suggested in previous works, NUET can be achieved by inserting constants inside encryptions. However, this has to be done carefully and consistently. Proper placement of constants is crucial since arbitrary placements will not guarantee NUET. For example, $[a, 1, N_B]_K$ and $[B, n_a, 2]_K$ both contain unique constants but are still unifiable[5]! In the following Lemma we prove that placing constants at the beginning of every concatenation ensures NUET.

Lemma 4. Let S be a semi-bundle and let $f_1(t_1)$ and $f_2(t_2)$ be two terms in S. Let $t_1 = [c_1, t_1']$ and $t_2 = [c_2, t_2']$. Then $c_1 \neq c_2 \Rightarrow f_1(t_1) \not\approx f_2(t_2)$.

Proof. Let us suppose $f_1(t_1) \approx f_2(t_2)$ is true. Then, by definition, $t_1 \approx t_2$ is also true. i.e,
$\exists \sigma \centerdot \sigma t_1 = \sigma t_2 \Rightarrow \exists \sigma \centerdot \sigma[c_1, t_1] = \sigma[c_2, t_2] \Rightarrow \exists \sigma \centerdot [c_1, \sigma t_1] = [c_2, \sigma t_2] \Rightarrow c_1 = c_2$, a contradiction.
Therefore, $f_1(t_1) \not\approx f_2(t_2)$.

□

With this result, we now demonstrate the placement of tags and the application of our results on the following protocol:

Msg 1. $A \rightarrow B : [A, B]$
Msg 2. $B \rightarrow A : [1, N_B]_{\overrightarrow{pk(A)}}$
Msg 3. $A \rightarrow B : [2, N_A]_{\overrightarrow{pk(B)}}$
Msg 4. $B \rightarrow A : [3, A, N_B]_{\overrightarrow{[4, N_A]}}$
Msg 5. $A \rightarrow B : [5, A, N_B]_{\overrightarrow{[6, A, N_B]}}$.

We have created this artificial protocol by incorporating constructed keys in messages 4 and 6 ($[4, N_A]$ and $[6, A, N_B]$). Notice the use of tags '4' and '6' inside the encryption keys which prevents the attack below:

Msg 1. $a \rightarrow b : [a, b]$
Msg 2. $b \rightarrow a : [1, n_b]_{\overrightarrow{pk(a)}}$
Msg 3. $I(a) \rightarrow b : [2, pk(\epsilon)]_{\overrightarrow{pk(B)}}$
Msg 4. $b \rightarrow I(a) : [3, a, n_b]_{\overrightarrow{pk(\epsilon)}}$
Msg 5. $I(a) \rightarrow b : [4, a, n_b]_{\overrightarrow{[a, n_b]}}$.

The attack works by using a type-flaw in Msg 3 ($pk(\epsilon)$ in place of N_A) and extracting n_b in Msg 4 by the attacker since it is now encrypted with his own public-key. This attack cannot be avoided by excluding tags inside constructed keys and this observation was never made in previous works. Ramanujam *et al.*'s paper explicitly states that their condition only mandates non-unifiability of encrypted subterms in a protocol

[5] with a unifier $\sigma = \{[a, 1]/B, [n_a, 2]/N_B\}$.

which is sufficient only for atomic keys. As the attack above demonstrates, when a constructed key contains no ground terms, the attacker can unify the key with his own public key and extract the contents inside the encryption.

The above representation for the protocol is in the form of parametric strands. Let both the roles in the protocol be named $roleA$ and $roleB$. Changing them to semi-strands using a consistent substitutions σ_a^h and σ_b^h where $\sigma_a^h = \{a/A, b/B, n_a/N_A\}$ and $\sigma_b^h = \{b/B, n_b/N_B\}$ we obtain,

$$roleA = \langle +[a,b], -[1,N_B]_{\overrightarrow{pk(a)}}, +[2,n_a]_{\overrightarrow{pk(B)}}, -[3,a,N_B]_{\overrightarrow{h(4,n_a,N_B)}}, +[5,a,N_B]_{\overrightarrow{h(6,a,N_B)}}\rangle,$$
$$roleB = \langle -[A,b], +[1,n_b]_{\overrightarrow{pk(A)}}, -[2,N_A]_{\overrightarrow{pk(B)}}, +[3,A,n_b]_{\overrightarrow{h(4,N_A,n_b)}}, -[5,A,n_b]_{\overrightarrow{h(6,A,n_b)}}\rangle.$$

Let $St = \{roleA, roleB\}$ and $S = \{\sigma_a^h roleA, \sigma_b^h roleB\}$. Assuming well-typed($\sigma_a^h$) and well-typed($\sigma_b^h$), we can now conclude from our main result that S is devoid of type-flaw attacks, since St is NUET-*satisfying* and therefore S is NUET-*complying*.

5.1 Password protocols

As explained earlier, constants should not be placed inside password encryptions as they enable direct guessing attacks. However, constants are not required to achieve NUET, as terms may as well be arranged so as to satisfy NUET. A simple solution is to make sure that every concatenation in a semi-bundle starts with a (secret) ground atomic value such as a nonce.

Consider the terms $[n_a, K_A]_{passwd(a,b)}$ and $[K_B, n_b]_{passwd(a,b)}$. These terms are unifiable, although they are textually distinct. However, a simple rearrangement of these terms as $[n_a, K_A]_{passwd(a,b)}$ and $[n_b, K_B]_{passwd(a,b)}$ would ensure that they are non-unifiable. Essentially, the condition which is achieved by placing constants at the beginning is now achieved by replacing them with ground atomic terms.

The existence of at least one such atomic value inside password encryptions has been demonstrated to be unavoidable for password protocols [Gon93]. The reason being that in the absence of such a requirement (and in the absence of any other redundancies), it would be impossible to confirm successful decryption.

5.2 Non-standard inference rules

The above theory on placement of constants is only valid when considering standard inference rules. For example, consider the WL π_1 protocol tagged with constants to be NUET-*satisfying*:

Msg 1. $A \rightarrow B : A$
Msg 2. $B \rightarrow A : N_B$
Msg 3. $A \rightarrow B : [A, B, N_B, 1]_{\overleftrightarrow{sh(A,S)}}$
Msg 4. $B \rightarrow S : [A, B, [A, B, N_B, 1]_{\overleftrightarrow{sh(A,S)}}, 2]_{\overleftrightarrow{sh(B,S)}}$
Msg 5. $S \rightarrow B : [A, B, N_B, 3]_{\overleftrightarrow{sh(B,S)}}$

Although tags have been placed at the end of every concatenation, the protocol still satisfies NUET going by a similar argument in the proof of Lemma 4. However, a type-flaw attack is still possible on this protocol:

Msg 1. $a \rightarrow b : a$
Msg 2. $b \rightarrow a : n_b$
Msg 3. $i(a) \rightarrow b : [n_b, 3]$ /* In place of $[a, b, n_b, 1]_{sh(a,s)}$ */
Msg 4. $b \rightarrow i(s) : [a, b, [n_b, 3], 2]_{sh(b,s)}$
Msg 5. $i(s) \rightarrow b : [a, b, [n_b, 3]]_{sh(b,s)}$ /* using CBC inf rule on Msg 4. */

The attack works because, the attacker induces a type-flaw in Msg 3 and reduces Msg 4 ($[a, b, [n_b, 3], 2]_{sh(b,s)}$) to obtain Msg 5 ($[a, b, [n_b, 3]]_{sh(b,s)}$).

The attack works despite our proof on absence of type-flaw attacks in presence of NUET, since it involves a decomposition rule that decomposes a term set to create a non subterm of the semi-bundle. Consequently, it affects the attacker substitution by introducing an ill-typed substitution. It falls outside the scope of Lemma 2 since it involves a term that was not present in the original semi-bundle. The term was produced by a rule not considered in Lemma 3 to prove that all rules only produce subterms that originally existed in the semi-bundle.

This attack can be easily avoided by placing constants placed at the beginning of every concatenation. We would like to point out however, that such tagging is not an epicure for all inference rules. For example, a similar rule for CBC encryptions is possible by taking advantage of the *suffix property* (as opposed to prefix property used by the previous rule):

$$[m, n]_k \vdash [n]_k$$

In presence of such a rule, placing constants at the end of every concatenation ensures type-flaw attack protection as terms produced by the rule can never be unified with existing terms to produce ill-typed unifiers.

Generally speaking, constants have to be so chosen so as to make unification with existing terms impossible when non-standard inference rules are in effect. Note that no such special care is needed when considering rules as below:

$$\{[a, [x, b]]_{\overrightarrow{k}}, [c, [x, d]]_{\overrightarrow{k}}, a, b, c, d\} \vdash x \text{ where } a \neq c \vee b \neq d$$

This rule was shown to be possible by Coppersmith *et al.* when using RSA method for asymmetric key encryption [CFPR96].

Such a rule does not affect our results in any way since the rule only produces a subterm of the existing semi-bundle (essentially retaining the validity of Lemmas 2 and 3). Therefore, no special care needs to be taken when such rules are in effect.

6 Conclusion

In this paper we redefined NUT and formally proved that it prevents type-flaw attacks in broader scenarios. We have shown example protocols, scenarios and attacks that fall outside the scope of the previous results. Of particular interest is the demonstration that our results are equally valid for protocols such as the WL π_1 protocol that fall outside Ramanujam *et al.'s* and other results.

Note that our result is also essentially a decidability result for protocols and can be considered as an extension of Ramanujam *et al.*'s results to broader scenarios. Ramanujam *et al.* follow a different approach for decidability by modeling protocol execution traces using finite state automaton and two counter machines. In the end they show that, in presence of tagging, analyses can only be restricted to well-typed runs (which is known to be decidable). Thus, they prove that tagging ensures attacks are possible only in a well-typed system, demonstrating that they are not type-flaw attacks. In other words, a proof of absence of type-flaw attacks is sufficient to conclude decidability of security for protocols on unbounded scenarios. This is exactly what we proved in this paper. Our results are also stronger in another sense — they clearly demonstrate under what class of attacker inference rules the decidability property holds. This was never considered by Ramanujam *et al.*

Decidability results on unbounded scenarios seal the problem of protocol security for trace based properties. The results enable the protocol designer to check whether a protocol is correct by simply following the condition for decidability and verifying the protocol (with single instance per role) using an analysis tool such as the constraint solver. This is a matter of few seconds.

The use of the constraint solving model as our framework allows us to reason our concepts (such as placement of constants) clearly, since it is based on a symbolic analysis approach that correlates replay of terms with unification. This is not easy in other approaches based on the strand space framework that start their analysis with ground instances of the protocol (e.g. [THG98, GT00a]).

Previous works based on the constraint solving model only used it to develop tools to analyze protocols on bounded scenarios [CE02, BMV03, CMAFE03]. However, the model can also be used as a general framework for proofs on unbounded scenarios, as is done in this paper. The basic modeling of protocol roles is the same in both the standard strand space framework and in the constraint solving model. They differ in the attacker model — While the strand space framework models the attacker using penetrator strands, the constraint solving method models them using term set operators and equates them to a symbolic reduction procedure. The net capability of the attacker is the same in both the approaches.

There are two important extensions to our theory that we are currently investigating at this point. The first concerns relaxing the free term algebra assumption by allowing for operations such as `XOR` and `Products` that contain algebraic properties such as cancellation (e.g. $a \oplus a \oplus b = b$). Our proof strategy in presence of those operations is the same as in this paper with the additional investigation as to how the attacker substitution is affected when a constraint is eliminated using unification. Our aim is to find out if for every attacker substitution, there exists an equivalent well-typed substitution that satisfies a constraint even when cancellation operations are considered. This remains to be seen.

Secondly, we have only considered trace based properties such as secrecy and authentication but would like to examine other properties such as anonymity and non-repudiation as well. The results in this paper are still more general than previous works for trace based properties as we have a single proof regardless of the properties under consideration[6], while other works require separate proofs for each property.

References

[AN94] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 122–136. IEEE Computer Society Press, 1994.

[Aur97] 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.

[BMV03] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Proc. 10th ACM Conference on Computer and Communication Security*, pages 335–344. ACM press, 2003.

[Boy90] C. Boyd. Hidden assumptions in cryptographic protocols. In *Proc. of the IEE, Part E*, pages 433–436, 1990.

[BP04] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Theoretical Computer Science. Special issue FoSSaCS'03. To appear*, 2004.

[CE02] R. Corin and S. Etalle. An Improved Constraint-based system for the verification of security protocols. *9th Int. Static Analysis Symp. (SAS)*, LNCS 2477:326–341, september 2002.

[6] including newly discovered properties against type-flaw and multi-protocol guessing attacks [MAFM00, MAF03]

[CFPR96] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. *Lecture notes in computer science*, 1070, 1996.

[CMAFE03] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks. In *Workshop in the Issues of Theory of Security (WITS03), Poland, Warsaw*, April 2003.

[FKK] A. Freider, P. Karlton, and P. Kocher. The SSL protocol Version 3.0. `http://home.netscape.com/eng/ssl3/`.

[Gon93] L. 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.

[GT00a] J. D. Guttman and F. J. Thayer. Authentication tests. In *Proceedings, 2000 IEEE Symposium on Secuirty and Privacy*, pages 96 – 108. IEEE Computer Society Press, May 2000.

[GT00b] J. D. Guttman and F. J. Thayer. Protocol Independence through Disjoint Encryption. *13th IEEE Computer Security Foundations Workshop*, pages 24–34, July 2000.

[HLS00] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. 13th Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society Press, July 2000.

[Low96a] 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.

[Low96b] G. Lowe. Some new attacks on cryptographic protocols. In *Proceedings of 9th Computer Security Foundations Workshop*. IEEE, 1996.

[Low04] G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12:83–98, 2004.

[MAF03] S. Malladi and J. Alves-Foss. How to prevent type-flaw guessing attacks on password protocols. In *Workshop on Foundations of Computer Security (FCS03)*, June 2003.

[MAFM00] S. Malladi, J. Alves-Foss, and S. Malladi. What are multi-protocol guessing attacks and how to prevent them. In *11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2002)*, pages 77–82. IEEE Computer Society, june 2000.

[Mea96] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[Mea04] C. Meadows. Towards a hierarchy of cryptographic protocol models. In *DIMACS Workshop on Security Analysis of Protocols*. DIMACS Center, CoRE Building, Rutgers University, Piscataway, NJ, June 2004.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. ACM Conference on Computer and Communication Security*, pages 166–175. ACM press, 2001.

[MV97] Mastercard and VISA. SET Secure Electronic Transaction Specification: Business Description. May 1997. Available electronically at `http://setco.org/set_specifications.html`.

[RS03] R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In *Workshop in the issues of theory of security (WITS03)*, 2003.

[RS05] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[SG92] S. Stubblebine and V. Gligor. On message integrity in cryptographic protocols. In *Proc. IEEE Symposium on research in security and privacy*, pages 85–104, 1992.

[Son99] D. X. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 192–22. IEEE Computer Society Press, 1999.

[THG98] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 160–171. IEEE Computer Society Press, 1998.

[THG99] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.

[WL94] T.Y.C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

A Reduction Procedure P [MS01]:

$C :=$ initial constraint sequence
$\sigma := 0$
`repeat`
  `let` $c^* = m : T$ be the constraint in C
    s.t. m is not a variable
  `if` c^* not found
    `output` **Satisfiable!**
  apply rule *(elim)* to c^* until no longer applicable
  $\forall r \in R$
    `if` r is applicable to C
      $\langle C'; \sigma' \rangle := r(C; \sigma)$
      create node with C'; add $C \rightarrow C'$ edge
  $\langle C; \sigma \rangle :=$ `pop`
  `push` $\langle C'; \sigma' \rangle$
`until emptystack`

B Set of Reduction Rules, R

$$
\begin{array}{lcl}
r^{un}(C_<, m : T, C_>; \sigma^\epsilon) & = & \langle \tau C_<, \tau C_>; \tau \cup \sigma^\epsilon \rangle \text{ where } \tau = \text{mgu}(m, t) \wedge t \in T; \\
r^{pair}(C_<, [m_1, m_2] : T, C_>; \sigma^\epsilon) & = & \langle C_<, m_1 : T, m_2 : T, C_>; \sigma^\epsilon \rangle; \\
r^{hash}(C_<, h(m) : T, C_>; \sigma^\epsilon) & = & \langle C_<, m : T, C_>; \sigma^\epsilon \rangle; \\
r^{penc}(C_<, [m]_k^{\rightarrow} : T, C_>; \sigma^\epsilon) & = & \langle C_<, k : T, m : T, C_>; \sigma^\epsilon \rangle; \\
r^{senc}(C_<, [m]_k^{\leftrightarrow} : T, C_>; \sigma^\epsilon) & = & \langle C_<, k : T, m : T, C_>; \sigma^\epsilon \rangle; \\
r^{sig}(C_<, \text{sig}_{pk(\epsilon)}(m) : T, C_>; \sigma^\epsilon) & = & \langle C_<, m : T, C_>; \sigma^\epsilon \rangle; \\
r^{split}(C_<, m : [t_1, t_2] \cup T, C_>; \sigma^\epsilon) & = & \langle C_<, m : t_1 \cup t_2 \cup T, C_>; \sigma^\epsilon \rangle; \\
r^{pdec}(C_<, m : [t]_{pk(\epsilon)}^{\rightarrow} \cup T, C_>; \sigma^\epsilon) & = & \langle C_<, m : t \cup T, C_>; \sigma^\epsilon \rangle; \\
r^{ksub}(C_<, m : [t]_k^{\rightarrow} \cup T, C_>; \sigma^\epsilon) & = & \langle \tau C_<, \tau m : \tau [t]_k^{\rightarrow} \cup \tau T, \tau C_>; \tau \cup \sigma^\epsilon \rangle, \\
 & & \quad \text{where } \tau = \text{mgu}(k, pk(\epsilon)), k \neq pk(\epsilon); \\
r^{sdec}(C_<, m : [t]_k^{\leftrightarrow} \cup T, C_>; \sigma^\epsilon) & = & \langle C_<, k : T, m : T \cup t \cup k, C_>; \sigma^\epsilon \rangle.
\end{array}
$$

Justification for NSPW

This paper focusses on decidability and completeness results for cryptographic protocols with considerations of newer protocol features and newer protocol environments.

Cryptographic protocols are being vigorously studied over the last decade. Only recently the area has attained a level of maturity and consolidation. Many useful tools have been built such as model checkers, theorem provers and special purpose analyzers to analyze cryptographic protocols [Low96a, Mea96, Son99, MS01]. Useful frameworks such as the strand space framework have been developed which aid in both tool construction and proofs by hand [THG99]. In this process, some well-known problems have been "solved". For example, security properties such as secrecy which were proven to be undecidable on unbounded scenarios have been shown to be decidable when protocols follow simple restrictions on message construction such as NUET.

The latest trend in the literature has been research into tools and results with extended protocol features and relaxed assumptions about the environment, such as cryptographic vulnerabilities [Mea04]. Tools are being developed, and decidability results are being published with these considerations. Following this new shift of paradigm in protocol analysis, we have presented a largely useful result for protocols, redefining a previously used condition plus identified several issues in its implementation. We redefined the previously advocated condition of NUET on protocols to suit the new features and environments. We have proven that under the redefined condition, the protocol security problem for trace based security properties remains decidable in the context of new features and some of the environments. We obtained this result by demonstrating the absence of type-flaw attacks under NUET.

We would like to emphasize through our work that this is the direction in which protocol analysis should proceed. Tools such as model checkers are useful to find attacks (and only in bounded models), but cannot be used to prove that a protocol is correct on unbounded scenarios.

To quote Dijkstra's famous observation,

> *Program testing can be used to find errors, but not to prove the absence of errors.*

Protocols are programs too. An end user would naturally be more satisfied to use a protocol that is provably correct, not merely an assurance of its security by demonstrating the lack of discovered flaws by tools. Using our results, a protocol designer can design a protocol following NUET and verify a bounded instance of the protocol using a tool and conclude its security. As mentioned earlier, this is a matter of few seconds.

Thus, we beleive that our paper is well qualified for consideration in to the New Security Paradigms Workshop. It presents a clear shift in the analysis paradigm through consideration of new features and environments for cryptographic protocol analysis as well as a shift of focus from analyzing protocols for attacks using tools towards evolving provably secure protocols. We believe our results will encourage active discussion on the application of our results to other features such as non-free term algebras and non-trace based security properties. We also hope that the discussion will clarify any concerns about our results and our implementation guidelines.

On a final note, we would like to thank the organizers of the NSPW 2005 for kind consideration of our paper for includsion in the workshop.

Sincerely
Sreekanth Malladi, PhD, Assistant Professor, University of Wisconsin-Superior
Carol Taylor, PhD, Instructor, University of Idaho.

Attendance Statement. To participate in the lively and extensive discussions at the New Security Paradigms Workshop, one of us will attend the workshop. One of the authors has already attended a past NSPW (2002)

and is well aware of the intensity with which the discussions take place.