# How to prevent type-flaw attacks on security protocols under algebraic properties

Sreekanth Malladi[1]* and Pascal Lafourcade[2]**

[1] Dakota State University
Madison, SD 57042, USA
Sreekanth.Malladi@dsu.edu
[2] Université Grenoble 1, CNRS,Verimag,
2 avenue de Vignate 38000 Gières France
pascal.lafourcade@imag.fr

June 16, 2009

**Abstract.** In this paper, we prove that type-tagging prevents type-flaw attacks on security protocols that use the Exclusive-OR operator as our main contribution. Our proof method is general and can be easily extended to other monoidal operators that possess properties such as Inverse and Idempotence. We also discuss how tagging could be used to prevent type-flaw attacks under other properties such as associativity of pairing, commutative encryption, prefix property and homomorphic encryption.

**Key words:** Cryptographic protocols, Type-flaw attacks, Tagging, Algebraic properties, Equational theories, Constraint solving, Decidability.

## 1 Introduction

A *type-flaw attack* on a protocol is an attack where a message variable of one type is essentially substituted with a message of a different type, to cause a violation of a security property.

Heather et al. proved that pairing constants called "tags" with each message prevents type-flaw attacks in their pioneer work [HLS03].

Does preventing type-flaw attacks have advantages?

– As Heather et al. pointed out, besides the obvious advantage to security in preventing these commonly and frequently reported attacks, preventing them also allows many unbounded verification approaches (e.g. [THG98,Coh00,HS00]) to be meaningful, since they assume the absence of type-flaw attacks;

* Supported by a doctoral SEED grant from the graduate school at DSU in 2007.
** Supported by ANR SeSur SCALP, SFINCS, AVOTE projects.

– Further, Ramanujam-Suresh found that the absence of any type-flaw attacks allows us to restrict analysis to well-typed runs only [RS03], which is a decidable problem; i.e., security can be decided with analyzing just a single session.

Thus, prevention of type-flaw attacks is a crucial and significant result toward protocol analysis and verification.

However, Heather et al.'s work only considered a basic protocol model with a free message algebra and perfect encryption. Operators such as Exclusive-OR and ciphers such as CBC possess algebraic properties that violate these assumptions. Recent focus in research projects world-wide has been to extend protocol analysis with algebraic properties to accommodate "real-world" protocols (e.g. [KT08,EMM07]). Naturally, a corresponding study into type-flaw attacks would be both crucial and interesting.

With this motivation, we examined several algebraic properties described in the survey by Cortier et al. [CDL06] such as:

– Associative pairing, Commutative encryption, and Monoidal theories that violate the free algebra assumption;
– the Prefix property, Homomorphic encryption, and Low-exponent RSA weakness that violate the perfect encryption assumption.

We report our observations in this paper. As our main contribution, we prove that type-tagging prevents all type-flaw attacks under XOR that possesses ACUN properties (Associativity, Commutativity, existence of Unity and Nilpotence). The proof approach is quite general and can be easily extended to other monoidal theories such as Inverse and Idempotence as well. We also advocate some prudent tagging practices to prevent type-flaw attacks under the other algebraic properties mentioned above.

*Organization.* In Section 2, we show how type-tagging can prevent type-flaw attacks under XOR using an example. In Section 3, we give a formal treatment of type-flaw attacks in a symbolic model and provide a simpler proof compared to [HLS03] that tagging prevents type-flaw attacks under XOR. In Section 4, we examine how the result withstands each algebraic property and suggest remedies in the form of prudent engineering principles. We sum up with a Conclusion.

## 2 Tagging prevents type-flaw attacks under XOR - Example

Consider the adapted Needham-Schroeder-Lowe protocol ($\mathsf{NSL}_\oplus$) by Chevalier et al. [CKRT03]:

$$
\begin{array}{lllll}
\textbf{Msg 1.} & A & \rightarrow & B & : [1,\ N_A,\ A]_{pk(B)}^{\rightarrow} \\
\textbf{Msg 2.} & B & \rightarrow & A & : [2,\ N_A \oplus B,\ N_B]_{pk(A)}^{\rightarrow} \\
\textbf{Msg 3.} & A & \rightarrow & B & : [3,\ N_B]_{pk(B)}^{\rightarrow}
\end{array}
$$

($A$ and $B$ are agent variables; $N_A$, $N_B$ are nonce variables; $[X]_Y^{\rightarrow}$ represents $X$ encrypted with $Y$ using an asymmetric encryption algorithm.).

A type-flaw attack is possible on this protocol even in the presence of component numbering (recently presented in [MH08]):

$$
\begin{array}{l}
\text{Msg } \alpha.\mathbf{1}.\quad a \rightarrow i : [1,\ n_a,\ a]_{pk(i)} \\
\quad \text{Msg } \beta.1.\quad i(a) \rightarrow b : [1,\ n_a \oplus b \oplus i,\ a]_{pk(b)} \\
\quad \text{Msg } \beta.2.\quad b \rightarrow i(a) : [2,\ n_a \oplus b \oplus i \oplus b,\ n_b]_{pk(a)} \\
\text{Msg } \alpha.\mathbf{2}.\quad i \rightarrow a : [2,\ n_a \oplus i,\ n_b]_{pk(a)} \quad (\text{replaying Msg } \beta.2) \\
\text{Msg } \alpha.\mathbf{3}.\quad a \rightarrow i : [3,\ n_b]_{pk(i)} \\
\quad \text{Msg } \beta.3.\quad i(a) \rightarrow b : [3,\ n_b]_{pk(b)}
\end{array}
$$

Notice the type-flaw in the first message ($n_a \oplus b \oplus i$ substituted for the claimed $N_A$) that induces a type-flaw in the second message as well. This is strictly a type-flaw attack since without the type-flaw and consequently without exploiting the algebraic properties, the same attack is not possible.

Component numbering cannot also prevent type-flaw attacks under the Inverse property that allows cancellation much like Nilpotence. Consider operators $\{+, -\}$, where $+$ is binary addition, $-$ a unary operator, and $0$ a constant. Then, if we change the $\oplus$ operator in the $\mathsf{NSL}_{\oplus}$ protocol to $+$, variable $N_A$ could be substituted with $n_a + i - b$ to form the same attack as with $\oplus$.

The above attack can be avoided if type-tagging were to be adopted for the elements of the `XOR` operator:

$$
\begin{array}{lllll}
\textbf{Msg 1.} & A & \rightarrow & B & : [1,\ N_A,\ A]_{pk(B)}^{\rightarrow} \\
\textbf{Msg 2.} & B & \rightarrow & A & : [2,\ [\mathsf{nonce}, N_A] \oplus [\mathsf{agent}, B],\ N_B]_{pk(A)}^{\rightarrow} \\
\textbf{Msg 3.} & A & \rightarrow & B & : [3,\ N_B]_{pk(B)}^{\rightarrow}
\end{array}
$$

**Msg** $\beta.\mathbf{2}$ is then not replayable as **Msg** $\alpha.\mathbf{2}$ even when $i(a)$ sends **Msg** $\beta.\mathbf{1}$ as $i(a) \rightarrow b : [1,\ [\mathsf{nonce}, n_a] \oplus [\mathsf{agent}, b] \oplus [\mathsf{agent}, i],\ a]_{pk(b)}^{\rightarrow}$, since **Msg** $\beta$**.2** then becomes $b \rightarrow i(a) : [2, [\mathsf{nonce}, [\mathsf{nonce}, n_a] \oplus [\mathsf{agent}, b] \oplus [\mathsf{agent}, i]] \oplus [\mathsf{agent}, b], n_b]^{\rightarrow}_{pk(a)}$.

This is not replayable as the required **Msg** $\alpha$**.2**: $i \rightarrow a : [2, [\mathsf{nonce}, n_a] \oplus [\mathsf{agent}, i], n_b]^{\rightarrow}_{pk(a)}$ because, inside **Msg** $\beta$**.2**, one occurence of $[\mathsf{agent}, b]$ is in $[\mathsf{nonce}, [\mathsf{nonce}, n_a] \oplus [\mathsf{agent}, b] \oplus [\mathsf{agent}, i]]$ and the other is outside. Hence, they cannot be canceled.

A similar reasoning applies to Inverse property for $\oplus$ instead of Nilpotence. We leave this for the reader to verify.

In the next subsection, we will prove these claims formally.

## 3 Type-tagging prevents type-flaw attacks: Proof

In this section, we present a formal proof extending an approach presented in [Mal04] that non-unifiability of encryptions (which can be ensured by tagging with component numbers) prevents type-flaw attacks with free operators and a more detailed type-tagging will prevent them under the monoidal `XOR` operator. Our proof is much simpler than [HLS03], and more importantly, allows us to easily study extrapolating the result to operators with algebraic properties. Furthermore, being a symbolic protocol model, the framework is quite flexible to include the much needed equational unification for additional equational theories.

### 3.1 Term Alegbra

We start off with a term algebra with mostly free operators except for the `XOR` operator.

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

**Variable** *(can be an Agent, Nonce etc., that are all subsets of* $Var$*);* **Constant** *(numbers 1,2, ...; name of the attacker* $\epsilon$ *etc.);* **Atom** *(split into sets agents, nonces etc.);* **Concatenation** *denoted* $[t_1, \ldots, t_n]$ *if* $t_1, \ldots, t_n$ *are terms;* **Public-Key** *denoted* $pk(A)$ *with* $A$ *of type Agent;* **Shared-Key** *denoted* $sh(A, B)$ *with* $A$ *and* $B$ *of type Agent;* **Asymmetric encryption** *denoted* $[t]^{\rightarrow}_{k}$ *where* $t$ *and* $k$ *are terms;* **Symmetric encryption** *denoted* $[t]^{\leftrightarrow}_{k}$ *where* $t$ *and* $k$ *are terms;* **Hash** *denoted* $h(t)$ *where* $t$ *is a term;* **Signature** *denoted* $Sig_k(t)$ *where* $t$ *is a term to be validated using the key* $k$*;* **XOR** *denoted* $t_1 \oplus \ldots \oplus t_n$ *where* $t_1, \ldots, t_n$ *are terms.*

We will drop the superscript $\rightarrow$ and $\leftrightarrow$ if the mode of encryption is irrelevant.

We will call terms with no atoms (but only constants and variables) as *parametric terms*. We will call a parametric term in which the variables were substituted with variables and/or atoms as a *semi-term*.

We will assume that the reader is familiar with the standard definitions of syntactic unification, and the most general unifier (mgu). We will write $t \approx t'$ if $t$ and $t'$ are unifiable.

As usual, *subterms* are defined to capture parts of messages:

**Definition 2.** **[Subterm]**
*Term $t$ is a* **subterm** *of $t'$ (denoted $t \sqsubset t'$) if*

- $t = t'$, *or*
- $t' = [t_1, \ldots, t_n]$ *with* $t \sqsubset t''$, *where* $t'' \in \{t_1, \ldots, t_n\}$ *or*
- $t' = [t'']_{k'}$ *with* $t \sqsubset t''$, *or*
- $t' = h(t'')$ *with* $t \sqsubset t''$, *or*
- $t' = Sig_k(t'')$ *with* $t \sqsubset t''$; *or*
- $t' = t_1 \oplus \ldots \oplus t_n$ *with* $t \sqsubset t''$ *where* $t'' \in \{t_1, \ldots, t_n\}$.

We will call encrypted subterms, hashes and signatures as *Compound Terms*; we will denote them as $CT(T)$ for a set of terms, $T$.

We will denote the type of a variable or atom $t$ as $type(t)$. We overload this to give the type of other terms. For instance, $type([t_1, \ldots, t_n]) = [type(t_1), \ldots, type(t_n)]$ and $type([t]_k) = [type(t)]_{type(k)}$.

We will call a substitution of a term $t$ to a variable $V$ a "well-typed" substitution, if $type(t) = type(V)$. We will call a set of substitutions $\sigma$ well-typed and write $\mathsf{well\text{-}typed}(\sigma)$ if all its members are well-typed; otherwise, we call $\sigma$ ill-typed.

We will assume that all operators in the term algebra except the `XOR` operator are free of equations of the form $t = t'$ where $t$ and $t'$ are two different terms. Thus, every equation between two terms that were not constructed with the `XOR` operator is of the form $t = t$. We will denote this theory, $E_{\mathsf{STD}}$.

On the other hand, we will assume that terms created with the `XOR` operator to contain the following equational theory denoted $E_{\mathsf{ACUN}}$ corresponding to it's ACUN algebraic properties: $t_1 \oplus (t_2 \oplus t_3) = (t_1 \oplus t_2) \oplus t_3$ (**A**ssociativity; $t_1 \oplus t_2 = t_2 \oplus t_1$ (**C**ommutativity); $t_1 \oplus 0 = t_1$ (existence of **U**nity); $t_1 \oplus t_1 = 0$ (**N**ilpotence)

We will denote the unification algorithms for terms constructed purely with the standard operators and purely with the `XOR` operator as $A_{\mathsf{STD}}$ and $A_{\mathsf{ACUN}}$ respectively.

Terms constructed using both the standard operators and the `XOR` operator can be unified using $A_{\mathsf{STD}}$, $A_{\mathsf{ACUN}}$ and the combination algorithm of Baader & Schulz [BS96] resulting in a finite number of most general unifiers.

## 3.2 Strands and Semi-bundles

The protocol model is based on the strand space framework of [THG98].

**Definition 3.** **[Node, Strand, Protocol]**
*A* **node** *is a tuple $\langle Sign, Term\rangle$ denoted $+m$ when it sends a term $m$, or $-m$ when it receives $m$. The* **sign** *of a node $n$ is denoted* $\mathrm{sign}(n)$ *that can be '+' or '−' and its* **term** *as* $\mathrm{term}(n)$ *derived from the term algebra. A* **strand** *is a sequence of nodes denoted $\langle n_1, \ldots, n_k\rangle$ if it has k nodes. Nodes in a strand are related by the edge $\Rightarrow$ defined such*

*that if* $n_i$ *and* $n_{i+1}$ *belong to the same strand, then we write* $n_i \Rightarrow n_{i+1}$*. A* **parametric strand** *is a strand with all parametric terms on its nodes. A* **protocol** *is a set of parametric strands.*

Protocol roles (or parametric strands) can be partially instantiated to produce *semi-strands* containing semi-terms on nodes obtained instantiating their parametric terms, depending on the knowledge of agents concerning the variables being instantiated: A variable is instantiated to an atom if the agent to which the strand corresponds to, either creates the atom according to the protocol or knows the value (*e.g.* being public such as an agent name). Variables may also be replaced with new variable substitutions in order for different semi-strands of the same parametric strand to be distinguishable. This is done if more than one instance of a role is visualized in an execution scenario.

We will denote the substitution to a parametric strand '$p$' by an honest agent leading to a semi-strand '$s$' as $\sigma_s^h p$.

For instance, role '$A$' in the $\mathsf{NSL}_{\oplus}$ protocol is the parametric strand

$$\mathbf{role_A} \;=\; \langle \;\; +[1,\ N_A,\ A]^{\rightarrow}_{pk(B)},\quad -[2,\ [\mathsf{nonce},\ N_A] \oplus [\mathsf{agent},\ A],\ N_B]^{\rightarrow}_{pk(A)},\quad +[3,\ N_B]^{\rightarrow}_{pk(B)} \;\; \rangle$$

and an agent '$a$' that plays the role could be the semi-strand

$$\sigma_s^h\ \mathbf{role_A} \;=\; \langle \;\; +[1,\ n_a,\ a]^{\rightarrow}_{pk(B)},\quad -[[\mathsf{nonce},\ n_a] \oplus [\mathsf{agent},\ a],\ N_B]^{\rightarrow}_{pk(a)},\quad +[3,\ N_B]^{\rightarrow}_{pk(B)} \;\; \rangle$$

where $\sigma_s^h = \{a/A,\ n_a/N_A\}$.

A set of semi-strands is a *semi-bundle.* We will denote the set of all substitutions to a protocol by honest agents leading to a semi-bundle $S$ as $\sigma_S^H$.

We will assume that honest agent substitutions leading to semi-strands are always well-typed:

**Assumption 1** *Let* $P$ *be a protocol and* $S$ *be a semi-bundle such that* $S = \sigma_S^H P$*. Then,* $(\forall \sigma \in \sigma_S^H)(\mathsf{well\text{-}typed}(\sigma))$*.*

We will use the relation '**precedes**' $(\preceq)$ on stand-alone strands in semi-bundles: Let $s$ be a strand in a semi-bundle $S$. Then,
$(\forall n_i,\ n_j \in s)(i \leq j \Rightarrow n_i \preceq n_j)$.

We will abuse the notation of $CT()$ on strands, protocols and semi-bundles as well. We will write $t \in S$ even if $t$ is a term on some node of some strand of a semi-bundle $S$.

## 3.3 Constraints and Satisfiability

We use the constraint solving model of Millen-Shmatikov [MS01] that was later modified by Chevalier [Che04] to model the penetrator[3].

[3] Heather et al. [HLS03] used classical penetrator strands of [THG98], but the basic penetrator capabilities are equal in both models.

The main constraint satisfaction procedure, namely $\mathbf{P}_\oplus$, first forms a constraint sequence from an interleaving of nodes belonging to strands in a semi-bundle:

**Definition 4.** **[Constraint sequence]**
*A* **constraint sequence** $C = \langle\ term(n_1) : T_1, \ \ldots, \ term(n_k) : T_k\rangle$ *is from a semi-bundle* $S$ *with* $k$ *'$-$' nodes if* $(\forall n)(\forall n')((term(n') : T \in C) \wedge (term(n) \in T) \Rightarrow (n \preceq n'))$*. Further, if* $i < j$ *and* $n_i$*,* $n_j$ *belong to the same strand, then* $n_i \preceq n_j$ *and* $(\forall i = 1 \text{ to } k)(T_i \subseteq T_{i+1})$*.*

A symbolic reduction rule applied to a constraint $m : T$ is said to "reduce" it to another constraint $m : T'$ or $m' : T$. $\mathbf{P}_\oplus$ applies a set of such rules $R_\oplus$ (Table 1) in any order to the first constraint in a sequence that does not have a variable as it's target, called the "**active constraint**". It is worth mentioning that $\mathbf{P}_\oplus$ eliminates any stand-alone or free variables in the term set of a constraint before applying any rule.

| | | | | | |
|---|---|---|---|---|---|
| concat | $[t_1, \ldots, t_n] : T$ | $t_1 : T, \ldots, t_n : T$ | split | $t : T \cup [t_1, \ldots, t_n]$ | $t : T \cup t_1 \cup \ldots \cup t_n$ |
| penc | $[m]_k^{\rightarrow} : T$ | $k : T, \ m : T$ | pdec | $m : [t]_{pk(\epsilon)}^{\rightarrow} \cup T$ | $m : t \cup T$ |
| senc | $[m]_k^{\leftrightarrow} : T$ | $k : T, \ m : T$ | sdec | $m : [t]_k^{\leftrightarrow} \cup T$ | $k : T, \ m : T \cup \{t, k\}$ |
| $\mathsf{XOR_R}$ | $m : T \cup \{t_1, \ldots, t_n\}$ | $m : T \cup t_1 \oplus \ldots \oplus t_n$ | $\mathsf{XOR_L}$ | $t_1 \oplus \ldots \oplus t_n : T$ | $t_1 : T, \ldots, t_n : T$ |
| Sig | $Sig_k(f(t)) : T$ | $t : T$ | Hash | $h(t) : T$ | $t : T$ |

**Table 1.** Set of reduction rules, $R_\oplus = R \cup \{\ \mathsf{XOR_L}, \ \mathsf{XOR_R}\ \}$

The rules in Table 1 do not affect the attacker substitution. There are two other rules that involve unification, and generate a new substitution that is to be applied to the whole sequence before applying the next rule. It is worth giving a more detailed account of those rules including the transformation to the constraints before the active constraint ($C_<$) and the ones after ($C_>$):

$$\frac{C_<, \ m : T \cup t, \ C_>; \ \sigma}{\tau C_<, \ \tau C_>; \ \tau \cup \sigma} \text{ where } \tau = \text{mgu}(m, t) \quad (\mathsf{un})$$

$$\frac{C_<, \ m : [t]_k^{\rightarrow} \cup T, \ C_>; \ \sigma}{\tau\ C_<, \ \tau\ m : \tau\ [t]_k^{\rightarrow} \cup \tau\ T, \ \tau\ C_>; \ \tau \cup \sigma}, \text{where } \tau = \text{mgu}(k, pk(\epsilon)), k \neq pk(\epsilon) \quad (\mathsf{ksub})$$

A sequence of applications of reduction rules on a constraint sequence can transform it into a "simple" constraint sequence:

**Definition 5.** **[Simple constraint sequence]**
*A constraint* $m : T$ *is a* **simple constraint** *if* $m$ *is a variable. A constraint sequence* $C$ *is a* **simple constraint sequence** *if every constraint in* $C$ *is a simple constraint.*

The possibility of forming bundles from a given semi-bundle can be determined by testing if constraint sequences from it are satisfiable. Satisfiability is usually defined in terms of attacker operations on ground terms; however, Chevalier [Che04] proved that $\mathbf{P}_{\oplus}$ is terminating, sound and complete with respect to the attacker capabilities. Hence, we define satisfiability directly in terms of the decision procedure:

**Definition 6.** **[Satisfiability]**
*A constraint is* **satisfiable** *if a sequence of reduction rule application from* $R$ *result in a simple constraint. A constraint sequence* $C$ *is* **satisfiable** *if every constraint in the sequence is satisfiable. Further, the initially empty substitution* $\sigma$ *is said to* **satisfy** $C$*, denoted* $\sigma \vdash C$.

It is useful to characterize "normal" constraint sequences which are those that do not contain pairs on the left and right sides of any constraint:

**Definition 7.** **[Normal Constraint Sequence]**
*A constraint sequence* $C$ *is* **normal** *iff for every constraint* $m : T \in C$, $m$ *is not a pair and for every* $t \in T$, $t$ *is not a pair.*

It has been proven in [Che04] that any constraint sequence can be "normalized" such that if a substitution satisfies the original sequence, it can also satisfy the normalized sequence.

Violations of trace properties such as secrecy and authentication can be embedded in a semi-bundle so that a satisfiable constraint sequence from the semi-bundle points to an attack. Using this concept, we define a type-flaw attack:

**Definition 8.** **[Type-flaw attacks]**
*A* **type-flaw attack** *exists on a semi-bundle* $S$ *if a constraint sequence* $C$ *from* $S$ *is satisfiable with an ill-typed substitution, but not with a well-typed substitution. i.e.* $(\exists\sigma)(\sigma \vdash C) \wedge (\nexists\sigma^{'})((\sigma' \vdash C) \wedge (\mathsf{well\text{-}typed}(\sigma^{'})))$.

## 3.4 Main requirement — Non-Unifiability of Terms

We will now state our main requirement on protocol messages which states that textually distinct compound terms should be non-unifiable and that all XORed terms must be type-tagged:

**Definition 9.** [NUT]
*Let $P$ be a protocol. Then $P$ is* NUT-Satisfying *iff*

– $(\forall t_1 \in CT(P))(\forall t_2 \in CT(P))(t_1 \neq t_2 \Rightarrow t_1 \not\approx t_2)$.;
– $(\forall t)(\forall t')((t \in P) \wedge (t = t_1 \oplus \ldots \oplus t_n) \wedge (t' \in \{t_1, \ldots, t_n\}) \Rightarrow (\exists t^{''})(t' = [type(t^{''}), t^{''}]))$.

It can be easily seen that NUT for terms constructed with standard operators is achieved by placing component numbers as the beginning element of concatenations inside all distinct compound terms in a protocol. E.g. $[1,\ N_A,\ A]^{\rightarrow}_{pk(B)}$, $[2,\ N_A,\ [3,\ N_B,\ A]^{\leftrightarrow}_{sh(A,B)}]^{\leftrightarrow}_{sh(B,S)}$, etc. Further, for terms that are XORed together, type tags must be included. For instance, $N_A \oplus B \oplus [1,\ N_A,\ A]_K$ should be transformed into $[\mathsf{nonce},\ N_A] \oplus [\mathsf{agent},\ B] \oplus [[\mathsf{nonce},\ \mathsf{agent}]^{\rightarrow}_{\mathsf{key}},\ [1,\ N_A,\ A]^{\rightarrow}_{K}]$.

The tagged $\mathsf{NSL}_\oplus$ protocol in Section **??** clearly conforms to these stipulations and hence is a NUT-Satisfying protocol.

### 3.5 Main result

We will now prove that NUT-Satisfying protocols are not vulnerable to type-flaw attacks.

The main idea is to show that every unification when applying $\mathbf{P}_\oplus$ to a constraint sequence from a NUT-Satisfying protocol results in a well-typed unifier.

The intuition behind showing that unifiers are necessarily well-typed is as follows: informally, the problem of unification of two terms under the combined theory of $(E_{\mathsf{STD}} \cup E_{\mathsf{ACUN}})$ must first result in subproblems that are purely in $E_{\mathsf{STD}}$ or purely in $E_{\mathsf{ACUN}}$ according to Baader-Schulz algorithm.

Now $E_{\mathsf{ACUN}}$ problems will have a unifier only if the XOR terms contain variables. However, according to our extended requirement of NUT above, no protocol term has an XOR term with an untagged variable. Further, the XOR terms produced by $\mathbf{P}_\oplus$ in the term set of a constraint cannot contain variables either since like in $\mathbf{P}$, the rule (*elim*) eliminates any stand-alone variables in a term set before applying any other rule. Thus, algorithm $A_{\mathsf{ACUN}}$ returns an empty unifier. Unification of $E_{\mathsf{ACUN}}$ problems only happens when two standard terms that were replaced by variables belong to the same equivalence class, can be unified with $A_{\mathsf{STD}}$ and could thus be canceled.

In summary, the unifier for a problem in $(E_{\mathsf{STD}} \cup E_{\mathsf{ACUN}})$ under the extended requirement on NUT is only from applying $A_{\mathsf{STD}}$. We show that these problems always produce well-typed unifiers.

For instance, consider the unification problem

$$[1,\ n_a]_{pk(B)} \ \overset{?}{\approx}_E \ [1,\ N_B]_{pk(a)} \ \oplus \ [2,\ A] \ \oplus \ [2,\ b]$$

Following Baader & Schulz method, we first purify this to sub-problems:

$$W \ \overset{?}{\approx}_{E_{\mathsf{STD}}} \ [1,\ n_a]_{pk(B)},\ X \ \overset{?}{\approx}_{E_{\mathsf{STD}}} \ [1,\ N_B]_{pk(a)},\ Y \ \overset{?}{\approx}_{E_{\mathsf{STD}}} \ [2,\ A],\ Z \ \overset{?}{\approx}_{E_{\mathsf{STD}}} \ [2,\ b],$$

$$\text{and } W \ \overset{?}{\approx}_{E_{\mathsf{ACUN}}} \ X \ \oplus \ Y \oplus \ Z.$$

Now, the new variables $W$, $X$, $Y$, and $Z$ are treated as constants during $A_{\mathsf{ACUN}}$. In that case, the problem $W = X \oplus Y \oplus Z$ is not unifiable. However, there is a step we missed: we need to form equivalence classes from the variables $W$, $X$, $Y$, and $Z$ such that variables from one class can be replaced with just one representative element. In this case, if we partition the variables into $\{ \{ W \}, \{ X \}, \{ Y, Z \} \}$, then we can change the problem $W \overset{?}{\approx}_{E_{\mathsf{ACUN}}} X \oplus Y \oplus Z$ into $W \overset{?}{\approx}_{E_{\mathsf{ACUN}}} X \oplus Y \oplus Y$ with an additional problem of $Y \overset{?}{\approx}_{E_{\mathsf{STD}}} Z$. This is obviously equivalent to $W \overset{?}{\approx}_{E_{\mathsf{STD}}} X$ since the $Y$'s cancel out leading to another sub-problem.

Now all the sub-problems are purely in the $\mathsf{STD}$ theory (terms on either sides do not involve the $\oplus$ operator):

$$[1,\ n_a]_{pk(B)} \overset{?}{\approx}_{E_{\mathsf{STD}}} [1,\ N_B]_{pk(a)},\quad [2,\ A] \overset{?}{\approx}_{E_{\mathsf{STD}}} [2,\ b].$$

It can be easily seen that $A_{\mathsf{STD}}$ outputs a well-typed unifier $(\{n_a/N_B, b/A\})$ for these problems resulting in a well-typed unifier for a combination of $A_{\mathsf{STD}}$ and $A_{\mathsf{ACUN}}$, since $A_{\mathsf{ACUN}}$ outputs an empty unifier.

**Theorem 1. [NUT prevents type-flaw attacks]**
*Let $P$ be a* NUT*-Satisfying protocol and $S = \sigma^H_S P$. Let $C$ be a normal constraint sequence from $S$. Then,* $(\sigma \vdash C) \Rightarrow (\exists \sigma')((\sigma' \vdash C) \wedge (\mathsf{well\text{-}typed}(\sigma')))$.

*Proof.* If $\sigma$ satisfies $C$, then from Def. 6, rules from $R_{\oplus}$ have been used to reduce it to a simple constraint sequence. The only rules that can change $\sigma$ are $(un)$ and $(ksub)$. $(ksub)$ makes a well-typed substitution since it unifies a term with the attacker's public-key which is of the same type.

We prove below that if $m\ :\ T \cup t \in C$, and $m \approx t$ then for each $\mathrm{mgu}(m, t) = \tau$, $\mathsf{well\text{-}typed}(\tau)$. Since initially $\sigma$ is empty, using induction on each constraint of the sequence, we can then conclude that $\sigma$ is well-typed.

Following the combination algorithm of [BS96] described in [Tue06], let the initial problem of $\Gamma = \{m \overset{?}{\approx}_E t\}$ be reduced to $(\Gamma', <, p)$ where

– $\Gamma'$ is a set of unification problems $\{m_1 \overset{?}{\approx} t_1, \ldots, m_n \overset{?}{\approx} t_n\}$;
– Let $\Gamma'$ be pure with every $m \approx_E t \in \Gamma'$ have $m$, $t$ formed purely from operator $\oplus$ on 0, constants and variables or from the standard theory in the term algebra defined in Def. 1;
– $<$ is a linear ordering on variables such that if $X < Y$ then $Y$ does not occur as a subterm of the instantiation of $X$;
– $p$ is a partition $\{V_1, V_2\}$ on the set of all variables $V$ such that $V_2$ are treated as constants when $A_{\mathsf{STD}}$ is applied and $V_1$ are constants when $A_{\mathsf{ACUN}}$ is applied;
– Let another partition $p'$ of variables identifies equivalence classes of $V$ where every class in a partition is replaced with a representative and where members of the class are unifiable;

Let the combined unifier of $\sigma_{\mathsf{STD}}$ and $\sigma_{\mathsf{ACUN}}$ denoted $\sigma_{STD} \odot \sigma_{ACUN} = \sigma$ which is obtained by applying [Tue06, Def. 9]; i.e., by induction on $<$. Our aim is to prove that every $\sigma$ obtained for different combinations of $<, p, p'$ is well-typed.

Let us examine the possible forms of problem elements in $\Gamma'$:

**ACUN theory:** $m \overset{?}{\approx}_{\mathsf{ACUN}} t$ exist where $m = a_1 \oplus a_2 \oplus \ldots \oplus a_i$ and $b = b_1 \oplus b_2 \oplus \ldots \oplus b_j$ where each of $a \in \{a_1, \ldots, a_n\}$ and $b \in \{b_1, \ldots, b_j\}$ is a constant or a new variable in $V$. The reason is as follows: according to the requirement on protocol messages for a NUT-Satisfying protocol, none of $\{a_1, \ldots, a_i\}$ can be an untagged variable. Also, none of $\{b_1, \ldots, b_j\}$ is a variable, since $\mathbf{P}_{\oplus}$ applies rule (*elim*) eliminating all stand-alone variables before applying any other rule. Lastly, the new variables in $m$ and $t$ would be other problems in $\Gamma'$ of the form $X = [\mathsf{tag}, x]$ where $X$ is the new variable, $\mathsf{tag}$ is a constant and $x$ is any term. These new variables have to be treated as constants when applying $A_{\mathsf{ACUN}}$ (they cannot be substituted with 0's which is the only substitution that $A_{\mathsf{ACUN}}$ can return). With all constants in $m$ and $t$, $A_{\mathsf{ACUN}}$ that would normally return a set of '0' substitutions for some variables, returns an empty set of substitutions answering that $m$ and $t$ are equivalent (if they are);

**STD theory:** $m \approx_{\mathsf{STD}} t$ where

1. either $m$ or $t$ is a new variable belonging to $V$; there is no unifier to existing variables here;
2. $m, t$ are tagged terms of the form, $[\mathsf{tag}, x]$ and $[\mathsf{tag}, x']$ where $\mathsf{tag}$ is a constant. In this case, $m$ unifies with $t$ only if $x$ unifies with $x'$ and the proof can be applied recursively;
3. $m, t \in CT(S)$; In this case, again the proof applies recursively. For instance, if $m = h(m')$ and $t = h(t')$ then we need to unify $m'$ and $t'$; Suppose $m' = [\mathsf{tag}, x_1, \ldots, x_n]$ and $t' = [\mathsf{tag}, y_1, \ldots, y_n]$. The constant $\mathsf{tag}$ guarantees that $m'$ and $t'$ have the same number of elements ($n$). Now we need to unify every $x_i$ with $y_i$ for $i = 1$ to $n$. Firstly, if one of $x_i$ and $y_i$ is a variable, then:
   - If $x_i$ or $y_i$ is a new variable, there is no substitution to existing variables;
   - If both are existing variables, then they are both of the same type by Def. 9 and Assumption 1; similarly if one of them is an atom;

   If $x_i, y_i \in CT(S)$, then the proof proceeds recursively to each subterm in turn.
4. $m$ and $t$ are two new variables in a subset of the variable identification partition $p'$. However, since both are problems in $\Gamma'$ are such that they map to a tagged pair or compound term in the standard theory, their unifier is once again well-typed from above. Note that $m$ and $t$ cannot be existing variables since these variables are from ACUN problems and ACUN problems contain necessarily new variables as explained previously in the case for ACUN theory.

## 4 More algebraic properties

We now consider some more algebraic properties of message operators. The first set breaks the free algebra assumption for protocol messages like `XOR`. The second set breaks the perfect encryption assumption.

### 4.1 Algebraic properties with equational theories

**Monoidal theories.** Following the definition of monoidal theories from [CD07], we can determine that

- the theory $\mathsf{ACU}$ over $\{+, 0\}$ where $\mathsf{A}$ stands for associativity, $\mathsf{C}$ for commutativity and $\mathsf{U}$ for the existence of Unity is a monoidal theory;
- the theories $\mathsf{ACUIdem}$ and $\mathsf{ACUN}$ where $\mathsf{Idem}$ stands for Idempotence and $\mathsf{N}$ for Nilpotence are also monoidal theories over $\{+, 0\}$ and $\{\oplus, 0\}$ respectively;
- the theory of Abelian Groups ($\mathsf{AG}$ or $\mathsf{ACUInv}$) over $\{+, -, 0\}$ where $\mathsf{Inv}$ stands for Inverse is also monoidal where $-$ is a unary operator.

If we replace or overload the $\oplus$ operator in Section **??** with $\mathsf{Idem}$ or $\mathsf{Inv}$, we can make a similar reasoning as made for $\mathsf{ACUN}$ properties in Theorem 1:

When the combination algorithm of Baader & Schulz is applied for $E_{\mathsf{STD}} \cup E_{\mathsf{T}}$ where $T$ is a theory with any, some or all of $\mathsf{A}$, $\mathsf{C}$, $\mathsf{U}$, $\mathsf{N}$, $\mathsf{Idem}$, $\mathsf{Inv}$, the algorithm for $\mathsf{T}$, say $A_{\mathsf{T}}$ will return an empty substitution when the operator with theory is so used in the protocol such that every term is type-tagged. Consequently, the unifier for the combined unification problem will only have substitutions from $A_{\mathsf{STD}}$ which will be well-typed as explained in Theorem 1.

However, we must note that the procedure $\mathbf{P}_{\oplus}$ in [Che04] that we followed only considered $\mathsf{ACUN}$ properties. We conjecture that if a suitable constraint solving algorithm is developed for other monoidal theories as well, then the above concept of necessarily well-typed unifiers could be used to extend Theorem 1 under those theories.

**Associativity of Pairing.** This property allows the equation $[a, [b, c]] = [[a, b], c]$. Denote this as the theory $\mathsf{Assoc}$.

Component numbering cannot prevent ill-typed unifiers. A simple example can prove this: $[1, A, [b, c], d]$ can be unified using $[1, [a, B], C, d]$, with $\sigma = \{[a, B]/A, [b, c]/C, d/D\}$. Obviously, $\sigma$ is ill-typed.

However, type-tagging prevent ill-typed unifiers. If we consider the same example,
$[[\mathsf{agent}, A], [\mathsf{pair}, [[\mathsf{nonce}, b], [\mathsf{agent}, c]]], [\mathsf{key}, d]]$ cannot be unified with
$[[\mathsf{pair}, [[\mathsf{agent}, a], [\mathsf{nonce}, B]]], [\mathsf{agent}, C], [\mathsf{key}, D]]$ even under associativity, due to the "$\mathsf{pair}$" tag for pairs.

It would be straightforward to prove this claim formally:

- Following Baader-Schulz algorithm again, we can first purify the main unification problem into sub problems that are either purely in the $\mathsf{STD}$ theory and through the introduction of new variables, to those that resemble $m \approx t$ where all subterms of $m$ and $t$ are variables, atoms or pairs for the $\mathsf{Assoc}$ theory;
- The $\mathsf{STD}$ theory returns well-typed unifiers as described in the proof of Theorem 1;

– The unifiable problems in the Assoc theory will resemble $[[\mathsf{tag_1}, x_1], \ldots, [\mathsf{tag_n}, x_n]] \approx [[\mathsf{tag_1}, y_1], \ldots, [\mathsf{tag_n}, y_n]]$. This returns a well-typed unifier if all $x_i \approx y_i$ ($i = 1$ to $n$) return well-typed unifiers which they do if at least one of $x_i$ or $y_i$ are variables from Def. 9 and Assumption 1. If they are both compound terms, the proof proceeds recursively.

**Associativity and Commutativity of a general operator** The concepts above can easily be extrapolated to associativity of a general operator, say '.' as well. For instance, $[1, [a.b].C]$ and $[2, A.[b.c]]$ return an ill-typed unifier, but $[[[\mathsf{agent}, a].[\mathsf{nonce}, b]].[\mathsf{key}, C]]$ and $[[\mathsf{agent}, A].[[\mathsf{nonce}, b].[\mathsf{key}, c]]]$ do not.

These concepts can be extrapolated to commutativity as well: Consider $[1, n_a.B.a]$ unified with $[1, A.b.N_A]$ that results in an ill-typed unifier $\{n_a/A, b/B, a/N_A\}$ but type-tagging does not allow such a unification and ensures well-typed unification. Consider the same example: $[\mathsf{nonce.agent.agent}, [\mathsf{nonce}, n_a].[\mathsf{agent}, B].[\mathsf{agent}, a]]$ cannot be unified with $[\mathsf{nonce.agent.agent}, [\mathsf{agent}, A].[\mathsf{agent}, b].[\mathsf{nonce}, N_A]]$.

It should be straightforward to extend the formal proof that we outlined for associativity of pairing to the cases of associativity and commutativity of a general operator.

## 4.2 Algebraic properties with cipher weaknesses

Some algebraic properties violate the perfect encryption assumption, without altering the freeness of the message algebra. If they produce subterms, like the following inference rule due to Coppersmith [CFPR96], the main theorem still stands tall since unification in the STD theory will still be well-typed (recall the steps of STD theory unification in the proof of Theorem 1 that handles the case of $m \approx_{\mathsf{STD}} t$ – they consider $m$ and $t$ being subterms of the semi-bundle; In particular, step 3 considers the case of $m, t \in CT(S)$):

$$\{\ [a,\ [x,\ b]]_k^{\rightarrow},\ \ [c,\ [x,\ d]]_k^{\rightarrow},\ \ a,\ \ b,\ \ c,\ \ d\ \} \ \vdash\ x, \text{ where } a \neq c\ \vee\ b \neq d.$$

Clearly, since this inference produces a subterm ('$x$'), the main result stands tall in its presence and no type-flaw attacks can be possible if the protocol obeys NUT.

Some others produce non-subterms such as the Prefix property and homomorphic encryption discussed in [CDL06]. Let us examine if and how prudent tagging could be adopted to prevent type-flaw attacks under these properties:

**Prefix property.** The Prefix property is obeyed by block ciphering techniques such as CBC and ECB. This property leads the attacker to infer $[m]_k^{\leftrightarrow}$ (a non-subterm) from $[m, n]_k^{\leftrightarrow}$ thereby invalidating Theorem 1.

Consider the Woo and Lam $\pi_1$ protocol modified by inserting component numbers inside each encrypted component[4]:

[4] Heather et al. [HLS03] do not specify the exact position where component numbers need to be inserted, although they inserted numbers at the beginning of encryptions in their examples

Msg 1. $a \rightarrow b : a$
Msg 2. $b \rightarrow a : n_b$
Msg 3. $a \rightarrow b : [a, b, n_b, 1]^{\leftrightarrow}_{sh(a,s)}$
Msg 4. $b \rightarrow s : [a, b, [a, b, n_b, 1]^{\leftrightarrow}_{sh(a,s)}, 2]^{\leftrightarrow}_{sh(b,s)}$
Msg 5. $s \rightarrow b : [a, b, n_b, 3]^{\leftrightarrow}_{sh(b,s)}$

$sh(x, y)$ represents a shared-key between agents $x$ and $y$. We presented a type-flaw attack on this protocol in [MAF03] even when it uses component numbering if the Prefix property is exploited, and if pairing is associative:

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. $b \rightarrow I(s) : [a, b, [n_b, 3], 2]^{\leftrightarrow}_{sh(b,s)}$
Msg 5. $I(s) \rightarrow b : [a, b, n_b, 3]^{\leftrightarrow}_{sh(b,s)}$ /* using Prefix property on Msg 4. */

This attack works because, an attacker can infer $[a, b, n_b, 3]^{\leftrightarrow}_{sh(b,s)}$ from Msg 4 ($[a, b, [n_b, 3], 2]^{\leftrightarrow}_{sh(b,s)}$) exploiting the Prefix property and associativity of pairing.

This attack can be easily prevented by adopting type-tagging since it eliminates associativity of pairing as explained previously. It can also be prevented by simply inserting component numbers at the beginning of encryptions, instead of at the end.

**Homomorphism of Encryptions.** With this property, it would be possible to infer the non-subterms $[m]_k$, and $[n]_k$ from $[m, n]_k$. Obviously, this is stronger than the Prefix property.

The "pair" tag assumed to contain within parentheses cannot void this inference. For instance, a term $[[\mathsf{type}_1, t_1], [\mathsf{type}_2, t_2]]_k$ can still yield the non-subterms $[\mathsf{type}_1, t_1]_k$, and $[\mathsf{type}_2, t_2]_k$. Even with component numbering, a term such as $[1, [t_1, t_2]]_k$ can broken down into $[1]_k$, and $[t_1, t_2]_k$.

With a range of such non-subterm encryptions to infer, it can be easily seen that neither component numbers, nor type-tags, no matter how they are placed, can prevent the attack on the Woo and Lam protocol above under this inference.

In particular, if the plaintext block length equals the length of a nonce or agent, then the attacker can infer $[a]_{sh(b,s)}$, $[b]_{sh(b,s)}$, $[n_b]_{sh(b,s)}$ easily from Msg 4 under any tagging. He can then replay Msg 5 by stitching these together.

However, this inference is only possible under an extremely weak system such as ECB, so a realistic threat in real-world situations is unlikely.

## 5 Conclusion

In this paper, we provided a proof that adopting type-tagging for message fields in a protocol prevents type-flaw attacks under the ACUN properties induced by the most popular Exclusive-OR operator. We also extrapolated those results to many other interesting and commonly encountered theories.

We did not find a single property under which component numbering prevents type-flaw attacks that type-tagging cannot, although we presented several examples where the opposite could be true. However, we advocate the use of component numbering in addition to type-tagging since they prevent the replay of different terms with the same type as well.

The most significant advantage of being able to prevent type-flaw attacks is that analysis could be restricted to well-typed runs only. This has been shown to be a decidable problem in the standard, free theory but not for monoidal theories. We are currently in this pursuit. We previously made an attempt at this under the assumption that tagging might not prevent type-flaw attacks [CM07]. We intend to reattempt it by taking advantage of the results in this paper.

*Acknowledgments.* We benefited greatly from the following people's help and guidance: Gavin Lowe provided detailed and insightful observations into our proofs in [Mal04] that were helpful in this effort. Cathy Meadows suggested adopting of Baader & Schulz algorithm when dealing with XOR-unification. Jon Millen clarified numerous concepts about constraint solving and some crucial aspects of `XOR` unification.

## References

[BS96] F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. of Symbolic Computation*, 21:211–243, 1996.

[CD07] Véronique Cortier and Stéphanie Delaune. Deciding knowledge in security protocols for monoidal equational theories. In *LPAR*, pages 196–210, 2007.

[CDL06] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

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

[Che04] Y. Chevalier. A simple constraint solving procedure for protocols with exclusive-or. *Presented at Unif 2004 workshop*, 2004. available at http://www.lsv.ens-cachan.fr/unif/past/unif04/program.html.

[CKRT03] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18$^{th}$ Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 261–270. IEEE Computer Society Press, 2003.

[CM07] Y. Chevalier and S. Malladi. Decidability of "real-world" context-explicit security protocols. *Unpublished, incomplete draft*, 2007.

[Coh00] E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *Computer Security Foundations Workshop (CSFW)*, pages 144–158, 2000.

[EMM07] Santiago Escobar, Catherine Meadows, and José Meseguer. Equational cryptographic reasoning in the maude-nrl protocol analyzer. *Electr. Notes Theor. Comput. Sci.*, 171(4):23–36, 2007.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[HS00] J. Heather and S. Schneider. Towards automatic verification of security protocols on an unbounded network. In *Proc. 13th Computer Security Foundations Workshop*, pages 132–143. IEEE Computer Society Press, 2000.

[KT08] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *ACM Conference on Computer and Communications Security*, pages 129–138, 2008.

[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)*, Ottawa, Canada, June 2003.

[Mal04] S. Malladi. Phd dissertation - formal analysis and verification of password protocols. *ACM Portal, University of Idaho*, 2004.

[MH08] S. Malladi and G. S. Hura. What is the best way to prove a cryptographic protocol correct? (position paper). In *Workshop on Security in Systems and Networks (SSN 2008), IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008)*, pages 1–7, 2008.

[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.

[RS03] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *23rd FST&TCS, Lecture Notes in Computer Science*, volume 2914, pages 323–374, December 2003.

[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.

[Tue06] M. Tuengerthal. Implementing a Unification Algorithm for Protocol Analysis with XOR. Technical Report 0609, Institut für Informatik, CAU Kiel, Germany, 2006.