

# Extending constraint solving for cryptographic protocol analysis with non-standard attacker inference rules \*

## Abstract

In this paper, we extend constraint solving for protocol analysis with two non-standard attacker inference rules that are possible when using CBC method for symmetric key encryption and RSA method for asymmetric key encryption. We explain our technique and implementation using some examples which include previously unknown attacks on popular protocols. We also simplify the framework of constraint solving as presented by Millen and Shmatikov.

**Keywords.** Communication Security, Cryptographic Protocols

## 1 Introduction

Constraint solving for bounded process cryptographic protocol analysis was first presented by Millen and Shmatikov [MS01]. Since then, the technique has gained popularity due to its simplicity, clean representation of the problem, and ease of use [CE02, BMV03, CMAFE03, MS03]. In this paper, we present an extension of constraint solving with two non-standard attacker inference rules:

1.  $[m, n]_k^{\leftrightarrow} \models [m]_k^{\leftrightarrow}$ .

$[m, n]$  denotes pairing  $m$  and  $n$ .  $[t]_k^{\leftrightarrow}$  denotes  $t$  encrypted with  $k$  using a symmetric keying algorithm.  $\models$  denotes the relation *infer*.

This rule was long known to hold when using CBC (Cipher Block Chaining) method for symmetric key encryption provided  $m$  is a whole number of blocks [Boy90, SG92].

2. Similarly, Coppersmith *et al.* have shown that the rule

$$\{[a, [x, b]]_k^{\leftrightarrow}, [c, [x, d]]_k^{\leftrightarrow}, a, b, c, d\} \models x$$

where  $a \neq c \vee b \neq d$

\*This work was partially supported by the National Science Foundation under Grant No. 0313727

$[t]_k^{\rightarrow}$  denotes  $t$  encrypted with  $k$  using an asymmetric keying algorithm) would hold, under certain assumptions, when using RSA method for asymmetric key encryption [CFPR96].

We implemented both inference rules in the constraint solver and tested them on several examples<sup>1</sup>. Interestingly, the seemingly simple CBC rule presented unforeseen difficulties in both theory and implementation. On the other hand, the RSA rule, which has resisted all attempts at incorporation into unbounded analysis, was straightforward to incorporate in the constraint solver and has a satisfactory theory. The implementation using examples is explained in **Section 3**.

Our implementations preserve the termination, soundness and completeness properties of the original constraint solving algorithm. Due to space limitations, proofs for these claims are given separately in a paper to be submitted.

We also simplify the framework of the original Millen-Shmatikov constraint solving paper by observing that the notion of “encryption hiding” is in fact redundant.

## 2 Constraint Solving

The aim of constraint solving is to answer the following question:

*Given a scenario with a finite number of protocol participants, is an attack possible in this scenario?*

The term algebra is comprised of simple terms, namely, variables and constants as well as compound terms such as pairs, encryptions, hashes and signatures formed by using corresponding operators on terms. A ground term is any term that does not contain a variable.

Variables are denoted using uppercase letters (e.g.  $N_A, B$ ) while constants using lowercase letters (e.g.  $a, n_b$ ).

Pairing of terms  $t_1$  and  $t_2$  is denoted  $[t_1, t_2]$ . We represent  $t$  encrypted with  $k$  using a symmetric keying algorithm

<sup>1</sup>On-line demo at <http://constraintsolver.org>

by  $[t]_k^{\leftrightarrow}$  ( $k$  can be any term); if the superscript is  $\rightarrow$  instead of  $\leftrightarrow$ , it represents  $t$  encrypted with  $k$  using an asymmetric keying algorithm. Public-key of agent  $X$  is denoted  $pk(X)$ . The hash of  $t$  is written  $h(t)$ , and  $sig_k(t)$  denotes signature of term  $t$  validated using key  $k$ . The constant  $\epsilon$  denotes the attacker.

## 2.1 Protocol Model

Constraint solving relies on the strand space framework [THG98] for its protocol model. Each role in a protocol is represented using a parametric strand, which is a series of communication nodes, sending or receiving messages. A protocol, which is a collection of roles, is thus a set of parametric strands called a *semi-bundle* (a term originally coined by Song [Son99]).

## 2.2 Standard Attacker Capabilities

We capture standard attacker capabilities as inference rules on the set of terms known to the attacker, namely pairing, splitting, encryption, and decryption, as well as hashing and signatures.

**Definition 1.** *The set of attacker term set operators is defined as  $\Phi = \Phi_{\text{analz}} \cup \Phi_{\text{synth}}$  with  $\Phi_{\text{analz}} = \{\phi_{\text{split}}, \phi_{\text{pdec}}, \phi_{\text{sdec}}\}$  and  $\Phi_{\text{synth}} = \{\phi_{\text{pair}}, \phi_{\text{penc}}, \phi_{\text{send}}, \phi_{\text{hash}}, \phi_{\text{sig}}\}$ , where*

$$\begin{aligned}\phi_{\text{split}}(S) &= S \cup \{x\} \cup \{y\} \text{ if } [x, y] \in S, \\ \phi_{\text{pdec}}(S) &= S \cup \{x\} \text{ if } [x]_{pk(\epsilon)}^{\rightarrow} \in S, \\ \phi_{\text{sdec}}(S) &= S \cup \{x\} \text{ if } [x]_y^{\rightarrow}, y \in S,\end{aligned}$$

$$\begin{aligned}\phi_{\text{pair}}(S) &= S \cup \{[x, y]\} \text{ if } x, y \in S, \\ \phi_{\text{penc}}(S) &= S \cup \{[x]_y^{\rightarrow}\} \text{ if } x, y \in S, \\ \phi_{\text{send}}(S) &= S \cup \{[x]_y^{\rightarrow}\} \text{ if } x, y \in S, \\ \phi_{\text{hash}}(S) &= S \cup \{h(x)\} \text{ if } x \in S, \\ \phi_{\text{sig}}(S) &= S \cup \{sig_{pk(\epsilon)}(x)\} \text{ if } x \in S.\end{aligned}$$

Note that we use functional notation for the  $\phi$  operators (following [MS01]), although they are actually only relations, since in general we have a choice of terms on which we can act. Usually this will cause no problems, but when the situation demands, we will specify the term or terms on which our operator acts.

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

**Definition 2.** *Given a set of ground terms  $T$ , then  $\mathcal{F}(T)$  is the closure of  $T$  under  $\Phi$ : that is,  $t \in \mathcal{F}(T)$  iff there exist  $\phi_1, \dots, \phi_n \in \Phi$  such that  $t \in \phi_n(\dots(\phi_1(T))\dots)$ .*

## 2.3 Satisfiability

As explained before, in constraint-based analysis, the goal is to find an attack on a protocol scenario. The idea is to first include a violation of a security property in the semi-bundle that is to be tested. For example, to test secrecy, an artificial test strand emitting a (supposedly) secret term is included. Next, all nodes in the semi-bundle are combined into a “node-merge”. Since there are finitely many strands with finitely many nodes, there can be only finitely many node-merges.

Now if there is an attack on the scenario, an attacker has to derive all messages that are expected to be received by the participants in at least one node-merge. These requirements are called *constraints*. To derive a message, he has in his armory all messages sent before that message.

A typical constraint in the sequence is denoted as  $m : T$ , representing that  $m$  should be derivable from  $T$ . A constraint sequence  $C$  with  $n$  constraints would be written as

$$\langle m_1 : T_1, \dots, m_n : T_n \rangle.$$

If  $C$  is satisfiable, i.e.,

$$\forall m : T \in C, \sigma m \in \mathcal{F}(\sigma T)$$

where  $\sigma$  is the attacker’s substitution of ground terms for variables in  $C$ , then there is indeed an attack on the node merge from which the constraint sequence was extracted. Further,  $\sigma$  is said to *satisfy*  $C$  (denoted  $\sigma \vdash C$ ).

Satisfiability is determined using a *constraint satisfaction procedure* explained in the next subsection.

## 2.4 Constraint Satisfaction Procedure

The goal of the constraint satisfaction procedure is to find a consistent substitution that satisfies the given constraint sequence. The procedure formulated originally in [MS01] (denoted **P**) is given in the Appendix.

The procedure handles one constraint at a time—the first constraint in the sequence, say  $m : T$ , such that  $m$  is not a variable (called the *active constraint*). The procedure applies reduction rules that simplify the constraint. A typical reduction rule  $r$  is represented as:

$$\frac{C_{<}, m : T, C_{>}; \sigma}{C'_{<}, m' : T', C'_{>}; \sigma'} (r).$$

Here,  $C_{<}$  is the sequence in  $C$  before the active constraint  $m : T$  and  $C_{>}$  is the sequence after the active constraint.  $m' : T'$  is the reduced constraint, while  $\sigma$  and  $\sigma'$  are the original and modified attacker substitutions respectively.

If more than one rule can be applied on the active constraint, or if the same rule can be applied in more than one

way, the reduction tree branches. Indeed, the procedure generates a tree with nodes corresponding to the constraint sequences and the branches representing the applied rules.

If reduction rules are applied on a constraint to the extent that the left side of the constraint ( $m$  in  $m : T$ ) has been reduced to a variable (whereupon it is called a *simple constraint*), then the constraint is deemed solvable and the next constraint in the sequence is handled. If the attacker substitution has been changed, it is carried on to the next step. This process is continued until no more constraints are left (in which case the sequence is deemed satisfiable), or the process halts because one of the constraints in the sequence cannot be reduced to a simple constraint.

## 2.5 Perfect Encryption and Non-standard Inference Rules

The perfect encryption assumption means that encryption is assumed to be unforgeable and unbreakable. Formally, this takes two forms:

1. It should be impossible to create  $[t]_k$  without knowing *both*  $t$  and  $k$ . The Cipher Block Chaining cryptosystem for symmetric key encryption presents a potential weakness that violates this assumption. Here, given a term  $[t_1, t_2]_k^{\leftrightarrow}$ , the attacker can infer  $[t_1]_k^{\leftrightarrow}$  without actually possessing  $t_1$  or  $k$ .
2. If  $[t]_k$  is an encryption, then it should be impossible to gain any knowledge of  $t$  without possessing the inverse-key of  $k$ . Coppersmith et al. in [CFPR96] present a weakness in the RSA public-key cryptosystem, that enables an attacker to violate this condition. Here, an attacker can infer part of the plain-text,  $x$  from two encryptions,  $[a, [x, b]]_k^{\leftrightarrow}$  and  $[c, [x, d]]_k^{\leftrightarrow}$  without knowing the corresponding inverse of  $k$ .

Standard inference rules adopt the perfect encryption assumption. Hence, the above two rules that violate the assumption fall into the category of “non-standard inference rules”. Apart from the fact that these rules break the assumption in two different ways, they contain some interesting properties and challenges:

1. They arise out of weaknesses in cryptosystems that follow different modes of encryption (symmetric and asymmetric).
2. One creates a term that may have never existed in the original term set, while the other extracts a subterm of the original term set.
3. They look at deeper levels of term structure, causing problems with unification creating new variables (this was not the case with standard rules considered

in [MS01]). This complicates the question of the termination of the constraint satisfaction procedure.

In the next section we explain how we implemented these rules in the constraint solver. Before proceeding however, the reader is advised that the practical aspects of these rules (i.e. the possibility of their existence, the properties of the underlying cryptosystems and the mathematics which enable these rules) are not of primary importance here. Our aim is to study the feasibility of extending constraint-based analysis assuming such rules are possible, without exhaustively exploring known variants of these weaknesses. For more details about the weaknesses themselves, the reader is referred to [PQ01, CFPR96].

## 3 Extension and Implementation

### 3.1 Simplifying the Framework

We start off with a simplification of the original framework of [MS01] by eliminating encryption hiding.

The protocol model presented in [MS01] included terms of the form  $[x]_y$ , so-called “hidden” terms, as well as two additional operators,  $\phi_{open}$  and  $\phi_{hide}$ . The idea was that a symmetrically encrypted term  $[x]_y^{\leftrightarrow}$  could be taken out of play in a certain sense by hiding it after all applicable decryption rules had been applied to it. Presumably this was done to avoid the possibility of non-termination of the reduction procedure due to endless application of the decryption rule to the same term. The hidden term  $[x]_y$  could still come into play because it was treated the same as the corresponding encrypted term,  $[x]_y^{\leftrightarrow}$ , for purposes of unification. However, it turns out that for satisfiable constraints there is always a sequence of reduction rules which avoids the use of hidden terms.

Consider an example discussed in [MS01], namely the constraint  $k : [t]_k, [k]_{[t]_k}^{\leftrightarrow}$ . This constraint could have arisen by applying the (sdec) rule to the satisfiable constraint  $c = k : [t]_k^{\leftrightarrow}, [k]_{[t]_k}^{\leftrightarrow}$  to decrypt the term  $[t]_k^{\leftrightarrow}$ . However, the correct reduction step to use on  $c$ , according to the normal derivation [MS01, section A.4.1.1], is to apply (sdec) to decrypt the term  $[k]_{[t]_k}^{\leftrightarrow}$ . Thus, no hidden term is necessary in this case.

That hidden terms are not needed in general, even with our non-standard inference rules, is proved in our technical report [MR05]. Consequently, we have modified the (sdec) rule from that of [MS01] by eliminating the hidden term altogether.

### 3.2 CBC

The extension of the constraint solver with the CBC weakness starts with an extension of Definition 1. We add a

new term set operator  $\phi_{cbc}$  defined by:

$$\phi_{cbc}(S) = S \cup \{[t_1]_k^{\leftrightarrow}\} \text{ if } [t_1, t_2]_k^{\leftrightarrow} \in S$$

We then add a corresponding reduction rule to the set of rules used by procedure **P**:

$$\frac{C_{<}, m : T \cup [t_1, t_2]_k^{\leftrightarrow}, C_{>}; \sigma}{C_{<}, m : T \cup [t_1]_k^{\leftrightarrow}, C_{>}; \sigma}$$

We denote by  $\mathbf{P}_{cbc}$  the reduction procedure with this new rule included.

At first sight, it may seem that by getting rid of the term  $[t_1, t_2]_k^{\leftrightarrow}$  after the reduction, we could lose possible attacks. However, this is not the case (see completeness proofs for the above rule in [MR05]).

### 3.2.1 Example 1

We tested the rule on the following Lowe-modified Denning-Sacco shared-key protocol [Low97]:

$$\begin{aligned} \text{Msg 1. } A \rightarrow S &: A, B \\ \text{Msg 2. } S \rightarrow A &: [B, K_{AB}, T, [K_{AB}, A, T]_{sh(B,S)}^{\leftrightarrow}]_{sh(A,S)}^{\leftrightarrow} \\ \text{Msg 3. } A \rightarrow B &: [K_{AB}, A, T]_{sh(B,S)}^{\leftrightarrow} \\ \text{Msg 4. } B \rightarrow A &: [N_B]_{K_{AB}}^{\leftrightarrow} \\ \text{Msg 5. } A \rightarrow B &: [N_B - 1]_{K_{AB}}^{\leftrightarrow} \end{aligned}$$

( $sh(X, Y)$  represents a shared-key between agents  $X$  and  $Y$ . Also, we use n-ary concatenation of terms instead of binary for clarity.)

This attack is basically the same as the one reported in [CDL04] on the unmodified Denning-Sacco protocol (which consists of only the first three of the previous five messages):

$$\begin{aligned} \text{Msg 1. } i(b) \rightarrow s &: b, a \\ \text{Msg 2. } s \rightarrow i(b) &: [a, k_{ba}, T, [k_{ba}, b, T]_{sh(a,s)}^{\leftrightarrow}]_{sh(b,s)}^{\leftrightarrow} \\ \text{Msg 3. } i(k_{ba}) \rightarrow b &: [a, k_{ba}, T]_{sh(b,s)}^{\leftrightarrow} \\ \text{Msg 4. } b \rightarrow i(k_{ba}) &: [n_b]_a \\ \text{Msg 5. } i(k_{ba}) \rightarrow b &: [n_b - 1]_a \end{aligned}$$

**Explanation.** The attack works with the intruder first spoofing as  $b$  (playing role  $A$ ) to  $s$  and obtaining Msg 2. Then, she uses the CBC weakness on Msg 2 and extracts  $[a, k_{ba}, T]_{sh(b,s)}^{\leftrightarrow}$  from it. She then replays this term as Msg 3 to  $b$  (with  $b$  playing role  $B$ ) claiming to be agent  $k_{ba}$  (playing role  $A$ ). Since  $b$  cannot verify the contents inside Msg 3 against any known value, he accepts it. He then sends Msg 4 ( $n_b$  encrypted with a value that the attacker knows,  $a$ ). The attacker then simply sends Msg 5 since she can easily construct it. This attack assumes that the lengths of a key and an agent identity are the same and that it is possible to initiate communication using a key as an agent identity.

### 3.2.2 Example 2

For our next example, consider the Woo and Lam Protocol  $\pi_1$  [WL94]:

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

The solver found an attack on this protocol, previously reported by Heather *et al.* in [HLS00]. Heather *et al.* also suggest using ‘‘component numbers’’ inside all encryptions to prevent that attack. We inserted such numbers inside all the encryptions and still found basically the same attack, but this time involving the CBC weakness:

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

This attack works because an attacker can infer  $[a, b, n_b, 3]_{sh(b,s)}^{\leftrightarrow}$  from Msg 4 ( $[a, b, n_b, 3, 2]_{sh(b,s)}^{\leftrightarrow}$ ) using the CBC inference rule.

### 3.2.3 A Pitfall and its Remediation

Although the rule above suffices for many cases, it does not terminate when it is applied on terms such as  $[X]_k^{\leftrightarrow}$  where  $X$  is a variable. The reason is that the rule attempts to unify  $X$  with a pair  $[A, B]$ , succeeds and creates a new term  $[A]_k^{\leftrightarrow}$  with two new additional variables  $A$  and  $B$ . Next, the rule is applied on  $[A]_k^{\leftrightarrow}$ , creating more variables, and the process continues indefinitely. There are two possible remedies for this problem:

1. Limit the number of times the rule is applied to terms of the form  $[X]_k^{\leftrightarrow}$  where  $X$  is a variable. The completeness of the decision procedure of [CKRT03] guarantees that the maximum number of such applications of the CBC rule needed to decide the satisfiability of the constraint set is bounded in terms of the specification of the protocol under consideration.
2. Another solution seems more satisfying but needs a radical change in the underlying framework: By modifying our term algebra with the assumption that the pairing operator is associative, i.e. by identifying

$$[A, [B, C]] = [[A, B], C] = [A, B, C]$$

for all  $A, B$ , and  $C$ , we achieve a cleaner approach to extending the basic Millen-Shmatikov model with

the CBC rule. In this approach, the CBC rule handles a term  $[X]_k^{\leftrightarrow}$  as a special case if  $X$  is a variable. The idea is that the general CBC solution to the constraint  $[m]_k^{\leftrightarrow} : [X]_k^{\leftrightarrow}$ , namely the sequence of solutions  $X = [m, X1]$ ,  $X = [[m, X1], X2]$ ,  $X = [[[m, X1], X2], X3], \dots$  can now be compactly represented in the form  $X = [m, A]$ .

In our current implementation (and proofs) we have adopted the first solution. Implementation and proofs for the second are currently being investigated.

### 3.3 RSA

The extension with the RSA weakness proceeds with the addition of a new term set operator  $\phi_{rsa}$  to  $\phi_{analz}$  as:

$$\phi_{rsa}(S) = S \cup \{x\} \\ \text{if } [a, [x, b]]_k^{\leftrightarrow}, [c, [x, d]]_k^{\leftrightarrow}, a, b, c, d \in S \\ \text{with } a \neq c \vee b \neq d$$

We then add a corresponding reduction rule:

$$\frac{C_{<}, m : T \cup [a, [x, b]]_k^{\leftrightarrow} \cup [c, [x, d]]_k^{\leftrightarrow}, C_{>}; \sigma}{C_{<}, a : T, b : T, c : T, d : T, m : T \cup x, C_{>}; \sigma}$$

where  $a \neq c \vee b \neq d$ .

When we implemented this rule in the solver, the inequality tests ( $a \neq c \vee b \neq d$ ) were included by allowing the rule to be applied only when the terms  $[a, b]$  and  $[c, d]$  are textually distinct.

However, the condition that  $[a, b]$  and  $[c, d]$  are textually distinct terms is not enough to ensure the soundness of the (rsa) rule, since these terms may be unified later in the reduction process, invalidating the use of (rsa) at a higher level of the reduction tree. Therefore, we introduce another element to each node of the reduction tree, namely a list  $\mathcal{L}$  of pairs  $[a, b]$ ,  $[c, d]$  which are not allowed to become textually equal at any stage of reduction. Our implementation checks all the pairs in  $\mathcal{L}$  for distinctness after each unification is applied during the reduction process.

#### 3.3.1 Example 1

We tested the rule on the following, modified version of the Needham-Schroeder Public-Key protocol [NS78]:

$$\begin{aligned} \text{Msg 1. } A \rightarrow B &: [N_A, A]_{pk(B)}^{\leftrightarrow} \\ \text{Msg 2. } B \rightarrow A &: [N_A, [N_B, B]]_{pk(A)}^{\leftrightarrow} \\ \text{Msg 3. } A \rightarrow B &: [N_B]_{pk(B)}^{\leftrightarrow}, N_A \\ \text{Msg 4. } B \rightarrow A &: [N_A, [N_B, X]]_{pk(A)}^{\leftrightarrow} \end{aligned}$$

Above we assume that  $X$  is a publicly-known message such as a common banking request. We attempted to test if  $N_B$  could be leaked when using the RSA weakness and the solver indeed reported a compromise:

$$\begin{aligned} \text{Msg 1. } a \rightarrow b &: [n_a, a]_{pk(b)}^{\leftrightarrow} \\ \text{Msg 2. } b \rightarrow a &: [n_a, [n_b, b]]_{pk(a)}^{\leftrightarrow} \\ \text{Msg 3. } a \rightarrow b &: [n_b]_{pk(b)}^{\leftrightarrow}, n_a \\ \text{Msg 4. } b \rightarrow a &: [n_a, [n_b, x]]_{pk(a)}^{\leftrightarrow} \\ \text{Test Msg. } \rightarrow &: n_b \end{aligned}$$

**Explanation.** The solver formed a ground instance depicting a normal run of the protocol and used the RSA weakness on  $[n_a, [n_b, x]]_{pk(a)}^{\leftrightarrow}$ ,  $[n_a, [n_b, b]]_{pk(a)}^{\leftrightarrow}$  to extract  $n_b$ .

#### 3.3.2 Example 2

As a second example, consider the following faulty toy protocol created by modifying Millen's "ffgg" protocol [Mil98]:

$$\begin{aligned} \text{Msg 1. } A \rightarrow B &: A, S, C_1, C_2, N_1 \\ \text{Msg 2. } B \rightarrow A &: X, Y \\ \text{Msg 3. } A \rightarrow B &: [S, [N_2, [N_1, C_1]]]_{pk(B)}^{\leftrightarrow} \\ \text{Msg 4. } B \rightarrow A &: [X, [N_2, [Y, C_2]]]_{pk(A)}^{\leftrightarrow} \end{aligned}$$

The aim was to test the secrecy of  $N_2$ . We input this protocol in the solver with one strand per role and it did not find an attack. In a scenario with one initiator strand and two responder strands, the solver found an attack compromising  $N_2$ :

$$\begin{aligned} \text{Msg 1. } a \rightarrow b &: a, s, c_1, c_2, n_1 \\ \text{Msg 2. } b \rightarrow a &: x_1, y_1 \\ \text{Msg 2'. } b \rightarrow a &: x_2, y_2 \\ \text{Msg 3. } a \rightarrow b &: [s, [n_2, [n_1, c_1]]]_{pk(b)}^{\leftrightarrow} \\ \text{Msg 4. } b \rightarrow a &: [x_1, [n_2, [y_1, c_2]]]_{pk(a)}^{\leftrightarrow} \\ \text{Msg 4'. } b \rightarrow a &: [x_2, [n_2, [y_2, c_2]]]_{pk(a)}^{\leftrightarrow} \\ \text{Test Msg. } \rightarrow &: n_2 \end{aligned}$$

**Explanation.** In the above attack,  $b$  plays the role of  $B$  two times. It uses different values of  $X$  and  $Y$  in each run, but the same  $N_2$ , which is kept secret. The attacker simply eavesdrops on the network, grabs  $[x_1, [n_2, [y_1, c_1]]]_{pk(a)}^{\leftrightarrow}$  and  $[x_2, [n_2, [y_2, c_2]]]_{pk(a)}^{\leftrightarrow}$ , uses the RSA weakness and extracts  $n_2$  as can be seen in the final "Test Msg". With the RSA rule deactivated, the solver did not find any attack in the same scenario.

## 4 Conclusion

The relevant code for our implementations is placed in the Appendix together with a screen shot of our on-line demo.

The aim of this paper is to explain what needs to be done to extend the constraint solving procedure with a "typical" non-standard attacker inference rule. Although only two in

number, the rules we have selected serve as good illustrations, since they represent much of the variety of traits that non-standard rules can present, as explained in Section 2.5.

The CBC rule can be implemented such that  $[t_1]_k^{\leftrightarrow}$  can be derived from  $[t_1, t_2]_k^{\leftrightarrow}$  only when  $t_1$  is of fixed block length. The current implementation is general in the sense that  $t_1$  can be of any size, but we believe that it is not difficult to handle the more specific case. Other similar rules can also be easily implemented; for example, the “suffix property” in some cryptosystems that enables derivation of  $[t_2]_k^{\leftrightarrow}$  from  $[t_1, t_2]_k^{\leftrightarrow}$  [CDL04].

It is important to realize that our proofs handle the CBC and RSA rules separately, although our current implementation allows the use of both at the same time. Obtaining results for both the rules working in tandem is a topic of current research.

We have not performed a complexity analysis of the extended algorithm and intend to pursue it as future work.

**Related work.** The original constraint solving paper by Millen-Shmatikov considered only standard attacker inference rules that did not violate the perfect encryption assumption. Since then, extensions were made to the algorithm in that paper to improve its speed and efficiency [CE02, BMV03], but retaining standard inference rules. Attempts have been made to extend the protocol model by relaxing the free-algebra assumption, allowing for algebraic properties of operations such as XOR, Products and Diffie-Hellman exponentiation (e.g. [MS03, CS03]). Some complexity results were also reported for similar operations, but only gave a non-deterministic decision procedure [CKRT03]. This paper differs from those contributions by including a practical decision procedure with an implementation and sample results. We retain the free algebra assumptions and extend the standard attacker capabilities with weaknesses known to exist in popular cryptosystems. No similar extensions to the constraint solving algorithm were ever published, to the best of our knowledge.

The CBC weakness was studied in many places (e.g. [SG92, PQ01]), and extensions to other tools were reported by considering the rule. E.g. The NRL protocol analyzer [SM00], and recently by Kremer *et al.* [KR04]. However, none of them involved a symbolic protocol model which is the basis for constraint-based analysis. We provide a much more rigorous study of the CBC rule, uncover a previously unreported pitfall and establish its properties.

Relatively fewer works reported study of the RSA weakness pertaining to protocol security. Roscoe *et al.* [RB99] have shown that incorporating the weakness would be impossible without destroying the foundation on which their protocol model is built. On the same lines, Heather *et al.* [HS02] later show that two other frameworks

for unbounded protocol analysis, the Strand Space model and the Rank Functions model also suffer from similar difficulties. We believe that we are the first to consider this daunting rule in an analysis tool, let alone in symbolic models, bounded scenarios and constraint-based analysis.

**Acknowledgments.** Thanks to Dr. Jon Millen and Dr. Vitaly Shmatikov for helpful discussions.

## References

- [BMV03] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Proc. 10<sup>th</sup> 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.
- [CDL04] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Technical Report LSV-04-15, Laboratoire Spécification et Vérification, ENS Cachan, France*, 2004.
- [CE02] R. Corin and S. Etalle. An Improved constraint-based system for the verification of security protocols. *9<sup>th</sup> Int. Static Analysis Symp. (SAS)*, LNCS 2477:326–341, september 2002.
- [CFPR96] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. *Lecture notes in computer science*, 1070, 1996.
- [CKRT03] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18<sup>th</sup> Annual IEEE Symposium on Logic in Computer Science (LICS’03)*, pages 261–270. IEEE Computer Society Press, 2003.
- [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 *Proc. Workshop on Issues in the Theory of Security (WITS’03)*, 2003.
- [CS03] H. Comon and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of Exclusive or. In

- Proc. 18<sup>th</sup> Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 271–280. IEEE Computer Society Press, 2003.
- [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.
- [HS02] J. Heather and S. Schneider. Equal to the task? In *Proc. 7th European Symposium on Research in Computer Security (ESORICS'02)*, pages 162–177, 2002.
- [KR04] S. Kremer and M. D. Ryan. Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. In *Proc. 2<sup>nd</sup> International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04)*, Electronic Notes in Theoretical Computer Science, August 2004.
- [Low97] G. Lowe. A family of attacks upon authentication protocols. *Department of Mathematics and Computer Science, Technical Report 1997/5, University of Leicester*, 1997.
- [Mil98] J. Millen. A necessarily parallel attack. In *Proc. Workshop on Formal Methods and Security Protocols (FMSP'98)*, 1998.
- [MR05] S. Malladi and S. Rosenberg. To be submitted for publication. 2005.
- [MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8<sup>th</sup> ACM Conference on Computer and Communication Security (CCS'01)*, pages 166–175. ACM press, 2001.
- [MS03] J. Millen and V. Shmatikov. Symbolic protocol analysis with Products and Diffie-hellman exponentiation. In *Proc. 16<sup>th</sup> Computer Security Foundations Workshop (CSFW-16)*, pages 47–61. IEEE Computer Society Press, 2003.
- [NS78] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.
- [PQ01] O. Pereira and J.-J. Quisquater. On the perfect encryption assumption. In *Proc. Workshop on Issues in the Theory of Security (WITS'01)*, 2001.
- [RB99] A. W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security: Special Issue CSFW12*, 7(2,3):147–190, 1999.
- [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.
- [SM00] S. Stubblebine and C. Meadows. Formal characterization and automated analysis of known-pair and chosen-text attacks. *IEEE Journal on Selected Areas in Communications*, 8(4), April 2000.
- [Son99] D. X. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proc. 12<sup>th</sup> IEEE Computer Security Foundations Workshop*, pages 192–202. 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.
- [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 and Reduction Rules R

$C$  := 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  
       push  $\langle C'; \sigma' \rangle$   
        $\langle C; \sigma \rangle := \text{pop}$   
 until emptystack

Reduction Procedure P [MS01]

The entire set of reduction rules,  $R$ , considered by P is given below:

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

## B CBC weakness in Prolog

The following piece of code was added to the constraint solver to implement the CBC weakness in Section 3.2:

```

reduct(M,T,[M,T1],Num_cbc_vars,New_num_cbc_vars):-
  do_cbc(T,T1,Num_cbc_vars,New_num_cbc_vars),
  write(' Applying CBC rule on '), write(T), writeln(' decomposing to '),
  write(T1).

do_cbc([X+K|T],[A+K|T],Num_cbc_vars,Num_cbc_vars):-
  not(var(X)),X=[A,B],writeln([X+K|T]).
do_cbc([X+K|T],[A+K|T],Num_cbc_vars,New_num_cbc_vars):-
  var(X),Num_cbc_vars > MAX_TRIES,!,fail.
do_cbc([X+K|T],[A+K|T],Num_cbc_vars,New_num_cbc_vars):-
  var(X),New_num_cbc_vars is Num_cbc_vars + 1,X=[A,B].
do_cbc([H|T],[H|T1],Num_cbc_vars,New_num_cbc_vars):-
  do_cbc(T,T1,Num_cbc_vars,New_num_cbc_vars).

```

Note: MAX\_TRIES will be replaced before runtime by a constant depending on the protocol  $P$  being tested; this constant may be taken to be  $4|P|$ , following the notation of [CKRT03]. Num\_cbc\_vars and New\_num\_cbc\_vars were added as appropriate to several other predicates.

## C RSA weakness in Prolog

The following piece of code was added to the constraint solver to implement the RSA weakness in Section 3.3:

```

reduct(M,T,[A,T3],[B,T3],[C,T3],[D,T3],[M,T1],RSA_LIST,NEW_RSA_LIST):-
  member([A,[X,B]]*K,T),delete_exactly(T,[A,[X,B]]*K,T2),
  member([C,[X,D]]*K,T2),delete_exactly(T2,[C,[X,D]]*K,T3),
  K=pk(PK),
  T1=[X|T3],
  append([[A,B],[C,D]],RSA_LIST,NEW_RSA_LIST),
  checklist(NEW_RSA_LIST),
  write(' Applying RSA rule on '), write([A,[X,B]]*K), write(' and '),
  write([C,[X,D]]*K), write(' to extract '), write(X), write(' ... '),
  writeln(' ').

% delete_exactly deletes all elements of list which are equivalent to X
% (NOT all elements that unify with X)
delete_exactly([H|T],X,T1):-
  H == X,
  delete_exactly(T,X,T1).

delete_exactly([H|T],X,[H|T1]):-
  H \== X,
  delete_exactly(T,X,T1).

delete_exactly([],X,[]).

checklist([]).%checks that no pairs in the RSA list have become equivalent

checklist([[X,Y]|T]):-
  X \== Y,
  checklist(T).

```

Note: RSA\_LIST and NEW\_RSA\_LIST arguments were also added to several other predicates, and checklist is called in the rules (un) and (ksub) as well as (rsa).

Constraint Solver Project - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop <http://lflux.uwsuper.edu/security/index.php> Search Print

Home Bookmarks mozilla.org mozillaZine mozillaDev.org FinalExam.htm

Home Interactive Demo Bibliography Documentation Contact Information

UNIVERSITY of WISCONSIN  
*Superior*  
Wisconsin's Public Liberal Arts College

Wisconsin Collaborative  
Project in Cyber Security\*  
\*supported by the National Science Foundation Award #0313727

NATIONAL SCIENCE FOUNDATION

### Extension of Millen-Shmatikov Constraint Solver

This Interactive Demo can be ran in a standard mode or with one or both additional rules added.

Include CBC  
 Include RSA

Select a protocol form the list below or chose "User input" to type in your own protocol in the text area on the top.

NSL modified (F)

```
% NSL protocol modified to test RSA
strand( roleA, A, B, Na, Nb, X, [
% recv( [ A, B ] ),
send( [ Na, A ]*pk(B) ), send( X ),
recv( [ Na, [ Nb, B ] ]*pk(A) ),
send( Nb*pk(B) ), send( Na ),
recv( [ Na, [ Nb, X ] ]*pk(A) )
] ).

strand( roleB, A, B, Na, Nb, X, [
recv( [ Na, A ]*pk(B) ),
send( [ Na, [ Nb, B ] ]*pk(A) ),
recv( [ Nb*pk(B), Na ] ),
send( [ Na, [ Nb, X ] ]*pk(A) )
] ).
```

Verify Reset

```
Applying RSA rule on [na, [nb, x]]*pk(a) and [na, [nb, b]]*pk(a)
to extract nb ...
nb is in the term set
nb : [nb, na, nb*pk(b), x, [na, a]*pk(b), a, b, e]
b is in the term set
b : [na, nb*pk(b), x, [na, a]*pk(b), a, b, e]
na is in the term set
na : [na, nb*pk(b), x, [na, a]*pk(b), a, b, e]
x is in the term set
x : [na, nb*pk(b), x, [na, a]*pk(b), a, b, e]
na is in the term set
na : [na, nb*pk(b), x, [na, a]*pk(b), a, b, e]
nb : [[na, [nb, x]]*pk(a), na, nb*pk(b), [na, [nb, b]]*pk(a), x,
[na, a]*pk(b), a, b, e]

Simple constraints:

RSA List:[[na, x], [na, b]]

Trace:
send([na, a]*pk(b))
```

Email: webmaster@constraintsolver.org ... Terms of Use ... University of Wisconsin-Superior, Superior, WI 54880

Figure 1. On-line demo for the Constraint Solver