A simple procedure for finding guessing attacks (Extended Abstract)

Ricardo Corin[1] and Sandro Etalle[1,2]
[1] Dept. of Computer Science, University of Twente, The Netherlands
[2]CWI, Center for Mathematics and Computer Science Amsterdam
{corin,etalle}@cs.utwente.nl

Abstract

A novel procedure for finding guessing attacks in security protocols is presented. The procedure enjoys a simple and intuitive definition, and is easily implementable.

1 Introduction

Security protocols that use weak passwords (e.g. human chosen) can be subject to *guessing attacks* [GLMS93]. Guessing attacks exist in two flavours: online and offline. In *online* guessing attacks the intruder is allowed to generate fake messages and to supply them to the honest agents, for instance for checking whether a certain guess is correct. In *offline* guessing attacks, on the other hand, the intruder *first* gathers some knowledge K from the protocol execution, and *then* proceeds offline to perform a password search. Let us immediately see an example, using the following (contrived) protocol:

$$\begin{array}{lcl} 1.\ a \rightarrow b & : & (a, na) \\ 2.\ b \rightarrow a & : & \{na\}_{pab} \end{array}$$

Here, na is a's nonce, pab is a shared weak password between a and b, $(\cdot,\cdot)$ denotes message pairing and $\{na\}_{pab}$ is nonce na symmetrically encrypted with pab. After a protocol execution, the attacker (who has eavesdropped all the traffic) has gathered a knowledge set $K = \{(a, na), \{na\}_{pab}\}$. With this knowledge, the intruder can mount an offline guessing attack on pab, by proceeding as follows:

(0) obtain na by splitting (a, na)
(1) REPEAT
(2) generate a guess p % using a dictionary
(3) let na' be the result of decrypting $\{na\}_{pab}$ using p
(4) UNTIL na' matches na

In step (4), the knowledge K allows the intruder to check whether a given guess of pab is correct. In this case we say that pab is *guessable* wrt K. Intuitively, a password p is guessable in K if it is possible to infer a term v (the *verifier*) from $K \cup \{p\}$ in two *different ways* with the understanding that p plays a crucial role in at least one of these ways of deriving v. In our example, the verifier was na, which is derivable from $\{na\}_{pab}$ using the guess and from (a, na) by projection.

Contributions. In this paper, we propose a simple procedure to find offline guessing attacks, given a knowledge set K and a guess p. It is based on the idea of obtaining the *minimal seed* of a knowledge K, as presented in [CDSV03]. Calculating the minimal seed before looking for guessing attacks allow us to automatically obtain a knowledge that has no "false attacks" (ie. two different derivations that were possible without guessing). Then, a procedure for looking for guessing attacks can be easily derived.

We have implemented the procedure in Prolog[1]; The prototype allowed us to test the security of several protocols (including the examples discussed on Section 3.2).

Related Work. The first definition of guessing attacks in a formal setting is given by Lowe [Low02]. The definition basically aims at capturing the notion of deriving some value in two ways.

However, Lowe's definition is not constructive, and its proposed implementation is based on a CSP model which is difficult to embed in other verification tools. Moreover, the procedure is tangled with the protocol verification process.

Moreover, it is not clear how to implement it in a verification procedure: a straightforward implementation of Lowe's definition would not terminate (in the Appendix, we report Lowe's definition).

Recently, another approach to guessing attacks was presented in [Del03], were also the complexity of guessing attacks was studied. It is worth mentioning that it is unclear whether the definition of guessing attack of [Del03] is equivalent to Lowe's. This work was subsequently extended in [DJ04], where an algorithm for finding guessing attacks is described, for a finite number of participants.

One last approach was presented in [CMAFE03] (partly authored by the present authors). There, guessing attacks are defined by using a "masking function" σ, that hides a possible verifier v with a special mark, say "$\mathbf{x}$", before checking its derivability. Then, the checking of whether v is derivable in two ways can be reduced to checking whether v *and* "$\mathbf{x}$" are derivable. Although the approach is intuitively sound, there are some cases in which the definition is not equivalent to Lowe's.

The presented approach seems to be compatible with the one of Lowe [Low02], although this is not formally proved. Indeed, it would be interesting to unify that work with Lowe's, and also with our setting.

2 Preliminaries

In this section we introduce the basic elements we need in the rest of the paper: the term algebra representing the exchanged messages, and the intruder model, based upon the Dolev Yao model [DY83].

Term Algebra. The set of terms $\mathcal{T}$ contains a set constants $\mathcal{C}$, representing the agent identities, nonces (ie. random values) and keys. We use a special constant $e \in \mathcal{C}$ to denote the intruder's identity. The set of terms is defined by the grammar defined in the left side of Table 1. This term algebra is quite standard: we have pairing, public keys and (symmetric/asymmetric) encryption. Finally, we assume that private keys are never part of messages, and so they are never leaked.

$t_1, t_2 ::=$	c	constant in $\mathcal{C}$	$\{t_1, t_2\}$	$\rightarrow_{pair} \{(t_1, t_2)\}$
	$pk(t_1)$	public key	$\{(t_1, t_2)\}$	$\rightarrow_{first} \{t_1\}$
	(t_1, t_2)	pair	$\{(t_1, t_2)\}$	$\rightarrow_{second} \{t_2\}$
	$h(t_1)$	hash	$\{t\}$	$\rightarrow_{hash} \{h(t)\}$
	$\{t_1\}_{t_2}$	symmetric encryption	$\{t_1, t_2\}$	$\rightarrow_{senc} \{\{t_1\}_{t_2}\}$
	$\{t_1\}^{\rightarrow}_{t_2}$	asymmetric encryption	$\{\{t_1\}_{t_2}, t_2\}$	$\rightarrow_{sdec} \{t_1\}$
			$\{t_1, t_2\}$	$\rightarrow_{penc} \{\{t_1\}^{\rightarrow}_{t_2}\}$
			$\{t_1\}$	$\rightarrow_{intpenc} \{\{t_1\}^{\rightarrow}_{pk(e)}\}$
			$\{\{t_1\}^{\rightarrow}_{pk(e)}\}$	$\rightarrow_{intpdec} \{t_1\}$

Table 1: Left: Grammar for terms. Right: Set of rules R_{std}.

Rules. Rules are used to represent the (standard) abilities of the attacker, in the style of Dolev-Yao. Let A and B be two sets of terms, and let ℓ be a rule label, representing the name of the rule. A *rule* is defined then as a triple $A \rightarrow_\ell B$. We work with the set of rules R_{std} given in the right side of Table 1.

As usual, the attacker is allowed to pair and split terms, hash, symmetrically encrypt with any (possibly non-atomic) key and decrypt symmetrically if the key is known to the attacker. Public-key

[1]There is an online demo located at `http://130.89.144.15/cgi-bin/ga/show.cgi`.

encryption is modeled by allowing to encrypt with any key (*penc*) and also encrypt with the intruder's public key $pk(e)$, which we assume the intruder always has. Then, rule *intpdec* models asymmetric decryption of a term encrypted with the attacker's public key. The attacker cannot decrypt any term encrypted with a different public key than his own, since we assume that private keys are not leaked. We now define a derivation relation over sets of terms.

Definition 1 *We write $T \rightarrow_r T'$ if there is an instance of a rule $r = \{t_1, ..., t_n\} \rightarrow_\ell \{s\} \in R_{std}$ such that $\{t_1, ..., t_n\} \subseteq T$ and $T' = T \cup \{s\}$.*

In the above definition, $1 \leq n \leq 2$ for R_{std}. Note that the application of rules to sets is monotonic in the sense that we never remove terms, and thus never "forget" information. A trace $tr = \langle r_1 \cdots r_n \rangle$, is a finite sequence of rule instances r_i, $i \in [1..n]$.

By iteratively applying rules from R_{std} to a given set, we obtain a *derivation*:

Definition 2 *Let T and T' be two sets of terms. We write $T \overset{tr}{\Rightarrow} T'$ if there is a finite path:*

$$T_0 \rightarrow_{r_1} T_1 \rightarrow_{r_2} ... \rightarrow_{r_n} T_n$$

Such that $T = T_0$, $T' = T_n$, $tr = \langle r_1 \cdot ... \cdot r_n \rangle$, and for each $i = [1..n]$, $r_i \in R_{std}$.

Note that $T \overset{\langle\rangle}{\Rightarrow} T$ for every T. We omit the trace and simply write $T \Rightarrow T'$ when we are not interested on the specific trace. We write $T \overset{tr}{\Rightarrow} t$ as a shorthand for $T \overset{tr}{\Rightarrow} T'$ for some T' s.t. $t \in T'$. Intuitively, $T \overset{tr}{\Rightarrow} t$ means that the attacker is able to produce the term t with trace tr from the knowledge T, using the rules R_{std}. Trace tr records the rules that were applied at each step of the derivation. Finally, we write $T \not\Rightarrow t$ when there is no trace tr s.t. $T \overset{tr}{\Rightarrow} t$.

Definition 3 *Given set of terms T, $\mathcal{C}(T)$ denotes the minimum set containing $\{t \in \mathcal{T} \mid T \Rightarrow t\}$.*

Given a finite set of terms T and a term t, it is easy to see that $t \in \mathcal{C}(T)$ is decidable.

3 Guessing Attacks

In this section we define our notion of guessing attacks. We first describe a procedure to obtain the *minimal seed* of a given knowledge K, in a similar vein of the work of [CDSV03]. Intuitively, this minimal seed of K will contain all the information contained in K but in compact form, with no redundant information.

Definition 4 *The minimal seed of T, $min(T)$ is the minimum subset of $\mathcal{C}(T)$ that satisfies the following properties for every $t, t_1, t_2, k \in \mathcal{T}$ and $c \in \mathcal{C}$:*

$$c \in min(T) \quad \text{iff} \quad T \Rightarrow c \tag{1}$$

$$\{(t_1, t_2)\} \quad \notin \quad min(T) \tag{2}$$

$$h(t) \in min(T) \quad \text{iff} \quad T \not\Rightarrow t \tag{3}$$

$$\{t\}_k \in min(T) \quad \text{iff} \quad T \not\Rightarrow k \tag{4}$$

$$\{t_1\}^{\rightarrow}_{t_2} \in min(T) \quad \text{iff} \quad T \not\Rightarrow t_1 \vee T \not\Rightarrow t_2 \text{ with } t_2 \neq pk(e) \tag{5}$$

$$\{t_1\}^{\rightarrow}_{pk(e)} \quad \notin \quad min(T) \tag{6}$$

Every constant that is derivable from T should also be derivable from $min(T)$ (1). Also, no pairs should be in $min(T)$, since they can always be splitted (2). The hash of a term t should only belong to $min(T)$ when t itself is not derivable from T (3). Similarly, we have that $\{t\}_k \in min(T)$ only when the key k is not derivable from T (4). For public key ciphertexts with encrypting key different than the

attacker's $pk(e)$, we require that $\{t_1\}^{\rightarrow}_{t_2} \notin min(T)$ when both t_1 and t_2 are derivable from T (5). Finally, we do not allow ciphertexts encrypted with the attacker's public key to reside in $min(T)$, since it can always be decrypted (6).

Even though this definition is not constructive, as shown in [CDSV03], we can obtain[2]:

Lemma 3.1 *Let T be a finite set of terms. Then,*

- *$min(T)$ is finite and computable.*
- *$\mathcal{C}(min(T)) = \mathcal{C}(T)$*

Example 3.2 *Let $T = \{k, \{na\}_k, h((na, k))\}$. Then $min(T) = \{na, k\}$.*

3.1 Procedure for finding guessing attacks

Now we are ready to give define a set $ga(\cdot)$, that contains the set of all verifiers corresponding to guessing attacks over a knowledge set and a guess.

In the following definition, we use $\biguplus$ to denote a disjoint union of sets.

Definition 5 *Let T be a finite set of terms, then $ga(T)$ is defined as the least set of terms satisfying:*

$$
\begin{array}{llll}
ga(T) & \ni & h(t) & \text{if } T = S \biguplus \{h(t)\} \text{ and } S \Rightarrow t \quad (r1)\\
ga(T) & \ni & t_1 & \text{if } T = S \biguplus \{(t_1, t_2)\} \text{ and } S \cup \{t_2\} \Rightarrow t_1 \quad (r2)\\
ga(T) & \ni & t_2 & \text{if } T = S \biguplus \{(t_1, t_2)\} \text{ and } S \cup \{t_1\} \Rightarrow t_2 \quad (r3)\\
ga(T) & \supseteq & ga(S \cup \{t_1, t_2\}) & \text{if } T = S \biguplus \{(t_1, t_2)\} \quad (r4)\\
ga(T) & \ni & t & \text{if } T = S \biguplus \{\{t\}_k\} \text{ and } S \Rightarrow k \text{ and } S \Rightarrow t \quad (r5)\\
ga(T) & \supseteq & ga(S \cup \{t\}) & \text{if } T = S \biguplus \{\{t\}_k\} \text{ and } S \Rightarrow k \wedge S \not\Rightarrow t \quad (r6)\\
ga(T) & \ni & \{t_1\}^{\rightarrow}_{t_2} & \text{if } T = S \biguplus \{\{t_1\}^{\rightarrow}_{t_2}\} \text{ and } t_2 \neq pk(e) \text{ and } S \Rightarrow t_2 \text{ and } S \Rightarrow t_1 \quad (r7)\\
ga(T) & \ni & t_1 & \text{if } T = S \biguplus \{\{t_1\}^{\rightarrow}_{t_2}\} \text{ and } t_2 = pk(e) \text{ and } S \Rightarrow t_1 \quad (r8)\\
ga(T) & \supseteq & ga(S \cup \{t_1\}) & \text{if } T = S \biguplus \{\{t_1\}^{\rightarrow}_{t_2}\} \text{ and } t_2 = pk(e) \text{ and } S \not\Rightarrow t_1 \quad (r9)
\end{array}
$$

Let us give the intuition behind the first rule (r_1): If the attacker can derive t from S, then there exists at least two "distinct ways" of deriving $h(t)$ from the set $S \biguplus \{h(t)\}$. Therefore, $h(t)$ is a verifier of a guess. Similarly, if there is a pair $(t_1, t_2) \in T$, t_1 is a verifier provided that from $S \cup \{t_2\}$ one derives back t_1 (analogously for t_2) (r_2) and (r_3). The pair is splitted in (r_4). In (r_5), we state that the plaintext t is a verifier if $\{t\}_k \in T$ and S allows one to derive both t and k. In (r_6), we add t to T, since we can decrypt thanks that $T \setminus \{t\}_k$ derives k. We proceed similarly for public key encrypted terms, in rules $(r7)$, $(r8)$ and $(r9)$.

We can actually *compute* the set $ga(\cdot)$:

Proposition 3.3 *If T is finite, then $ga(T)$ is computable.*

Proof - sketch. To be done. Structure: define a procedure that builds $ga(T)$, by implementing the above rules. Checking $T \Rightarrow t$ terminates. Note that for the three recursive cases ($r4$, 6 and $r9$) the standard multiset norm based on the term-depth norm is well-founded and decreasing. The thesis follows then from the fact that T is finite. Actually, we are not interested in computing the whole set $ga(T)$, but only in checking if it contains at least one element; Thus, we can implement a simple procedure to find a guessing attack (see the Appendix for the definition).

Another useful result is that when we combine $min(\cdot)$ and $ga(\cdot)$ we never obtain any verifiers. This will prevent us from finding any "fake" guessing attack.

Lemma 3.4 *Let T be a finite set of terms, then $\forall T : ga(min(T)) = \emptyset$.*

[2] In fact, our setting is *simpler* than the one of [CDSV03], which deals with message terms from a Spi calculus grammar.

Now we are ready to define guessing attacks:

Definition 6 *(guessing attack) We say that there is a guessing attack over T with guess g if $\exists v : v \in ga(min(T) \cup \{g\})$.*

In fact, any valid v is a verifier for a guessing attack; the above definition only concerns about the existence of one such verifier.

The above definition states that if $v \in ga(min(T) \cup \{g\})$, then $T \cup \{g\}$ must allow a situation which satisfies two things: First, no "fake" guessing attacks arise since anything that can be derived in two ways arise thanks to the guess g, since from no verifier can appear from $min(T)$ alone, thanks to Lemma 3.4. Second, v is derivable in two ways.

3.2 Examples

In this section we illustrate our procedure on some example protocols.

The protocol of the Introduction:

$$\begin{aligned} a \to b &: (a, na) \\ b \to a &: \{na\}_{pab} \end{aligned}$$

The intruder's knowledge at the end of the protocol is $T = \{\{na\}_{pab}, (a, na)\}$. Here, $min(T) = \{\{na\}_{pab}, a, na\}$. Suppose that pab is weak. Then we can check if a guess of pab can be verified by computing $ga(min(T) \cup \{pab\}) = ga(\{\{na\}_{pab}, a, na, pab\}) = \{na\}$. Since this set is non-empty, a guess of pab can be verified.

Gong's protocol. The previous protocol can be patched using a confounder c, to obtain the Gong exchange [GLMS93]:

$$\begin{aligned} a \to b &: \{(na, c)\}_{pk(b)} \\ b \to a &: \{na\}_{pab} \end{aligned}$$

If we assume that the intruder knows $pk(b)$ initially, then the gathered knowledge is $T = \{\{(na, c)\}_{pk(b)}, \{na\}_{pab}, pk(b)\}$. Also here we have that $min(T) = T$. To see if a guess of pab can be verified, we have to compute:

$$\begin{aligned} ga(min(T) \cup \{pab\}) &= ga(\{\{(na, c)\}_{pk(b)}, \{na\}_{pab}, pk(b), pab\}) \\ &= ga(\{\{(na, c)\}_{pk(b)}, na, pk(b), pab\}) \\ &= \emptyset \end{aligned}$$

Since this set is empty, a guess of pab is not verifiable, which confirms that using a confounder prevents the attack.

A real key exchange protocol. The EKE protocol [BM92]. The protocol consists of the following steps:

$$\begin{aligned} a \to b &: \{pk(k)\}_{pab} && \text{(EKE.1)} \\ b \to a &: \{\{r\}^{\rightarrow}_{pk(k)}\}_{pab} && \text{(EKE.2)} \\ a \to b &: \{na\}_r && \text{(EKE.3)} \\ b \to a &: \{(na, nb)\}_r && \text{(EKE.4)} \\ a \to b &: \{nb\}_r && \text{(EKE.5)} \end{aligned}$$

First, a generates a new private key k, and then it derives the public key $pk(k)$. Then, a encrypts $pk(k)$ with the shared password pab and sends it to b (EKE.1). Then, b extracts $pk(k)$, generates a fresh session key r and encrypts it with $pk(k)$. Then, b encrypts again the resulting message with pab and

sends it to a (EKE.2). The following three messages (EKE.i), $i = 3, 4, 5$, exchange nonces na and nb to perform the "hand-shaking" necessary to defend against replay attacks.

Here, $T = \{\{pk(k)\}_{pab}, \{\{r\}^{\rightarrow}_{pk(k)}\}_{pab}, \{na\}_r, \{(na, nb)\}_r, \{nb\}_r\}$. Also, $min(T) = T$. Let the guess be pab. Then,

$$\begin{aligned} ga(min(T) \cup \{pab\}) &= ga(\{\{pk(k)\}_{pab}, \{\{r\}^{\rightarrow}_{pk(k)}\}_{pab}, \{na\}_r, \{(na, nb)\}_r, \{nb\}_r, pab\}) \\ &= ga(pk(k), \{\{r\}^{\rightarrow}_{pk(k)}\}_{pab}, \{na\}_r, \{(na, nb)\}_r, \{nb\}_r, pab\}) \\ &= ga(pk(k), \{r\}^{\rightarrow}_{pk(k)}, \{na\}_r, \{(na, nb)\}_r, \{nb\}_r, pab\}) \\ &= \emptyset \end{aligned}$$

Thus confirming that EKE is secure. However, depending on the implementation, r can be strong or weak; the reader can check that if r is weak and thus guessable, we obtain an attack. Similarly, one can also find an attack if we take $pk(k)$ to be guessable. This can happen if e.g. a client uses the same k in two sessions (for details see [BM92]).

4 Conclusions

In this paper we have proposed a simple procedure for finding guessing attacks.

We have implemented our procedure in Prolog; More specifically, we implemented a procedure to calculate $min(T)$ (Definition 8) based on the work of [CDSV03], and the actual procedure to find attacks (Definition 7). Preliminary benchmarks are promising, showing a high performance (for example, verification of EKE as discussed in Section 3.2 takes 0.05s on a Pentium IV 1.8GHz with 256MB of RAM).

Moreover, the procedure presented in this paper can in fact be "plugged-in" to any tool capable of executing a security protocol, e.g. [CE02]. Briefly, we need to feed our procedure with the current intruder knowledge at each step of the verification.

At this moment we are working at proving our definition equivalent to that of Lowe [Low02].

References

[BM92] S.M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Security Conference on Research in Security and Privacy*, pages 72–84, 1992.

[CDSV03] I. Cibrario, L. Durante, R. Sisto, and A. Valenzano. A new knowledge representation strategy for cryptographic protocol analysis. In *Tools and Algorithms for the Construction and Analysis of Systems: 9th International Conference, TACAS 2003*, volume 2619 of *LNCS*, pages 284 – 298. Springer-Verlag, April 2003.

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

[CMAFE03] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks (extended abstract). In R. Gorrieri and R. Lucchi, editors, *IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, Warsaw, Poland, April 2003. Dipartamento di Scienze dell'Informazione Universita di Bologna, Italy.

[Del03] S. Delaune. Intruder deduction problem in presence of guessing attacks. In *Proc. Workshop on Security Protocols Verification (SPV'2003), Marseille, France, Sep. 2003*, pages 26–30, 2003.

[DJ04] S. Delaune and F. Jacquemard. A theory of guessing attacks and its complexity. In *Research Report LSV-04-1, Lab. Specification and Verification, ENS de Cachan*, 2004.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

[GLMS93] Li Gong, T. Mark A. Lomas, Roger M.Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[Low02] Gavin Lowe. Analyzing protocols subject to guessing attacks. *Workshop on Issues in the Theory of Security (WITS'02)*, January 2002.

A Appendix

A.1 A procedure to find guessing attacks

We define a procedure $\mathtt{ga}(\cdot)$ that implements the set-theoretic definition of $ga(\cdot)$, given in Definition 5.

Definition 7 *Procedure for searching guessing attacks of T. Let $\mathtt{ga}(T)$ be obtained as follows:*

$\mathtt{Procedure}\ \mathtt{ga}(T)$
$\mathtt{If}\ (t_1,t_2) \in T \wedge T \setminus \{(t_1,t_2)\} \cup \{t_2\} \Rightarrow t_1\ \mathtt{then}\ \mathtt{return}(t_1)$
$\mathtt{else\ if}\ (t_1,t_2) \in T \wedge T \setminus \{(t_1,t_2)\} \cup \{t_1\} \Rightarrow t_2\ \mathtt{then}\ \mathtt{return}(t_2)$
$\mathtt{else\ if}\ (t_1,t_2) \in T\ \mathtt{then}\ \mathtt{return}(\mathtt{ga}(T \cup \{t_1,t_2\}))$
$\mathtt{else\ if}\ \{t_1\}_{t_2} \in T \wedge T \setminus \{\{t_1\}_{t_2}\} \Rightarrow t_1,t_2\ \mathtt{then}\ \mathtt{return}(t_1)$
$\mathtt{else\ if}\ \{t_1\}_{t_2} \in T \wedge T \setminus \{\{t_1\}_{t_2}\} \Rightarrow t_2 \wedge T \setminus \{\{t_1\}_{t_2}\} \not\Rightarrow t_1\ \mathtt{then}$
$\mathtt{return}(\mathtt{ga}(T \setminus \{\{t_1\}_{t_2}\} \cup t_1))$
$\mathtt{else\ if}\ \{t_1\}^{\rightarrow}_{t_2} \in T \wedge t_2 = pk(e) \wedge T \setminus \{\{t_1\}^{\rightarrow}_{t_2}\} \Rightarrow t_1\ \mathtt{then}\ \mathtt{return}(t_1)$
$\mathtt{else\ if}\ \{t_1\}^{\rightarrow}_{t_2} \in T \wedge t_2 = pk(e) \wedge T \setminus \{\{t_1\}^{\rightarrow}_{t_2}\} \not\Rightarrow t_1\ \mathtt{then}\ \mathtt{return}(\mathtt{ga}(T \setminus \{\{t_1\}^{\rightarrow}_{t_2}\} \cup t_1))$
$\mathtt{else\ if}\ \{t_1\}^{\rightarrow}_{t_2} \in T \wedge t_2 \neq pk(e) \wedge T \setminus \{\{t_1\}^{\rightarrow}_{t_2}\} \Rightarrow t_2 \wedge T \setminus \{\{t_1\}^{\rightarrow}_{t_2}\} \Rightarrow t_1\ \mathtt{then}$
$\mathtt{return}(\{t_1\}^{\rightarrow}_{t_2})$
$\mathtt{else\ if}\ h(t) \in T \wedge T \setminus \{h(t)\} \Rightarrow t\ \mathtt{then}\ \mathtt{return}(h(t))$
$\mathtt{else\ if}\ \mathtt{return}(\bot)$

Note how the algorithm may generate different values, due to the non-deterministic choices in the different membership tests.

We obtain a relation between the set $ga(T)$ and the algorithm $\mathtt{ga}(T)$:

Proposition A.1 (Soundness) *. Let T be a finite set of terms. If $\mathtt{ga}(T) = v \neq \bot$, then $v \in ga(T)$.*

A.2 A procedure to calculate the minimal seed

We propose an alternative way of defininig the minimum seed of a set of terms T to the one proposed in Definition 8.

Definition 8 *Let T be a finite set of terms. The* minimizing rules *are the following rewriting rules:*

$$T \longrightarrow \begin{cases} S & \text{if } \exists S : T = S \uplus \{h(t)\} \wedge S \Rightarrow t \\ T & \text{otherwise} \end{cases} \quad (1)$$

$$T \longrightarrow \begin{cases} S \cup \{t_1, t_2\} & \exists S : \text{ if } T = S \uplus \{(t_1, t_2)\} \\ T & \text{otherwise} \end{cases} \quad (2)$$

$$T \longrightarrow \begin{cases} S \cup \{t\} & \text{if } \exists S : T = S \uplus \{\{t\}_k)\} \wedge S \Rightarrow k \\ T & \text{otherwise} \end{cases} \quad (3)$$

$$T \longrightarrow \begin{cases} S & \text{if } \exists S : T = S \uplus \{\{t_1\}^{\rightarrow}_{t_2})\} \wedge T \Rightarrow t_1 \wedge T \Rightarrow t_2 \text{ with } t_2 \neq pk(e) \\ T & \text{otherwise} \end{cases} \quad (4)$$

$$T \longrightarrow \begin{cases} S \cup \{t_1\} & \text{if } \exists S : T = S \uplus \{\{t_1\}^{\rightarrow}_{pk(e)})\} \\ T & \text{otherwise} \end{cases} \quad (5)$$

Notice that these rules are non-deterministic in the existence choice.

We define $minim(T)$ as the least fixed point obtained by starting from T and iteratively applying the minimizing rewrite rules (1)-(5).

To prove that $minim(T)$ is well-defined we have to proceed as follows: First, one has to show that none of the rewrite rules affect the $\Rightarrow$ relation (i.e. if $T \Rightarrow t$ and T is rewritten into T' then $T' \Rightarrow t$ as well). This is immediate from the definition of $\Rightarrow$, rewriting never forgets information ($T \Rightarrow T'$ implies that $T \subseteq T'$). Then one can show that the rewrite system is terminating and confluent.

Proposition A.2 *Let T be a finite set of terms. Then, $minim(T) = min(T)$.*

A.3 Lowe's definition of Guessing attacks

We report on the definition of guessing attacks given by Lowe in [Low02], adapted to our setting. We first need the definition of the *undo* relation over R_{std}:

$\{t_1, t_2\} \rightarrow_{pair} \{(t_1, t_2)\}$ *undo* $\{(t_1, t_2)\} \rightarrow_{first} \{t_1\}$
$\{t_1, t_2\} \rightarrow_{pair} \{(t_1, t_2)\}$ *undo* $\{(t_1, t_2)\} \rightarrow_{second} \{t_2\}$
$\{(t_1, t_2)\} \rightarrow_{first} \{t_1\}$ *undo* $\{t_1, t_2\} \rightarrow_{pair} \{(t_1, t_2)\}$
$\{(t_1, t_2)\} \rightarrow_{second} \{t_2\}$ *undo* $\{t_1, t_2\} \rightarrow_{pair} \{(t_1, t_2)\}$
$\{t_1, t_2\} \rightarrow_{senc} \{\{t_1\}_{t_2}\}$ *undo* $\{\{t_1\}_{t_2}, t_2\} \rightarrow_{sdec} \{t_1\}$
$\{\{t_1\}_{t_2}, t_2\} \rightarrow_{sdec} \{t_1\}$ *undo*
$\{t_1, t_2\} \rightarrow_{senc} \{\{t_1\}_{t_2}\}$
$\{t_1\} \rightarrow_{intpenc} \{\{t_1\}^{\rightarrow}_{pk(e)}\}$ *undo* $\{\{t_1\}^{\rightarrow}_{pk(e)}\} \rightarrow_{intpdec} \{t_1\}$
$\{\{t_1\}^{\rightarrow}_{pk(e)}\} \rightarrow_{intpdec} \{t_1\}$ *undo* $\{t_1\} \rightarrow_{intpenc} \{\{t_1\}^{\rightarrow}_{pk(e)}\}$

Now, Lowe's guessing attacks can be defined:

Definition 9 *There is a* Lowe *guessing attack over T with verifier v of guess g, if there exists trace tr, sets T_1, S, S', and labels ℓ, ℓ' s.t.:*

1. $T \cup \{g\} \stackrel{tr}{\Rightarrow} T_1$.
2. $S \rightarrow_\ell \{v\} \in tr$.
3. $\nexists T_2 \cdot T \Rightarrow T_2$ *and* $S \subseteq T_2$
4. $S' \rightarrow_{\ell'} \{v\} \in tr$, $(S, \ell) \neq (S', \ell')$ *or* $v \in T \cup \{g\}$.
5. $S \rightarrow_\ell \{v\}$ *and* $S' \rightarrow_{\ell'} \{v\}$ *do not undo any rule in* tr.