Passive intruder deduction

S. P. Suresh

Chennai Mathematical Institute
Chennai, India
spsuresh@cmi.ac.in

1 Preliminaries

Recall that we define *ground terms* $\mathscr{T}$ starting with *basic terms* $\mathscr{B} = \mathscr{A} \cup \mathscr{N} \cup \mathscr{K}$, where $\mathscr{A}$ is a countable set of agents, $\mathscr{N}$ is a countable set of nonces, and $\mathscr{K}$ is a countable set of keys (with a function $inv : \mathscr{K} \to \mathscr{K}$ such that $inv(k) = k'$ iff $inv(k') = k$). Complex terms are built according to the following syntax (where $m \in \mathscr{B}$):

$$t, t' ::= m \mid (t, t') \mid \{t\}_k$$

The set of *subterms* of a term t, denoted $\mathsf{st}(t)$, is defined to be the smallest set S such that: $t \in S$; if $(t, t') \in S$, $\{t, t'\} \subseteq S$; and if if $\{t\}_k \in S$, $\{t, k\} \subseteq S$. For a set X of terms, $\mathsf{st}(X)$ is defined to be $\bigcup\limits_{t \in X} \mathsf{st}(t)$. One can show that $|\mathsf{st}(X)| \leq \sum\limits_{t \in X} |t|$, where $|t|$ measures the size of (the syntax tree of) t.

$$\frac{}{X, t \vdash t}\,\mathit{ax}$$

$$\frac{X \vdash t_0 \quad X \vdash t_1}{X \vdash (t_0, t_1)}\,\mathit{pair} \qquad \frac{X \vdash (t_0, t_1)}{X \vdash t_i}\,\mathit{split}$$

$$\frac{X \vdash t \quad X \vdash k}{X \vdash \{t\}_k}\,\mathit{enc} \qquad \frac{X \vdash \{t\}_k \quad X \vdash \mathit{inv}(k)}{X \vdash t}\,\mathit{dec}$$

Figure 1 The Dolev-Yao system.

The Dolev-Yao system is defined by the derivation system in Figure 1. By $X \vdash_{\mathsf{dy}} t$, we mean that there is a derivation of $X \vdash t$ according to the rules of the above system. (For ease of notation, we drop the suffix and use $X \vdash t$ to refer to both the sequents and the fact that a sequent is derivable.)

▶ **Definition 1** (Derivability problem or Passive intruder deduction problem)**.** Given X and t, is it the case that $X \vdash_{\mathsf{dy}} t$?

Among the rules, *ax*, *split* and *dec* are the *elimination rules*, *pair* and *split* are the *introduction rules*. A *normal derivation* is one where the major premise of every elimination rule is the conclusion of an elimination rule. The following theorem is fundamental.

▶ **Theorem 2** (Weak normalization)**.** *If there is a derivation of $X \vdash t$ then there is a normal derivation of $X \vdash t$.*

Proof. [1] Consider a derivation π of $X \vdash t$ of minimum size. We claim that it is normal. Suppose not. Then π has a subproof ϖ which ends with an elimination rule whose major premise comes from an introduction rule. The following two cases need to be considered.

- ϖ is of the form

$$\dfrac{\dfrac{\overset{\varpi_0}{\vdots} \atop X \vdash t_0 \quad \overset{\varpi_1}{\vdots} \atop X \vdash t_1}{X \vdash (t_0, t_1)}\,\textit{pair}}{X \vdash t_i}\,\textit{split}$$

  Then ϖ can be replaced by ϖ_i, yielding a smaller proof than π of $X \vdash t$.
- ϖ is of the form

$$\dfrac{\dfrac{\overset{\varpi_0}{\vdots} \atop X \vdash t \quad \overset{\varpi_1}{\vdots} \atop X \vdash k}{X \vdash \{t\}_k}\,\textit{enc} \qquad \overset{\varpi_2}{\vdots} \atop X \vdash \textit{inv}(k)}{X \vdash t}\,\textit{dec}$$

  Then ϖ can be replaced by ϖ_0, yielding a smaller proof than π of $X \vdash t$.

Thus in both cases, we arrive at a contradiction to the fact that π is a derivation of $X \vdash t$ of minimum size. Thus it has to be the case that π is normal. ◀

▶ **Theorem 3** (Subterm property). *Let π be a normal derivation with conclusion $X \vdash t$ and last rule str. Let $X \vdash s$ occur in π. Then $s \in \mathsf{st}(X \cup \{t\})$. Furthermore, if str is an elimination rule, then $s \in \mathsf{st}(X)$.*

Proof. The proof is by induction on the structure of π, and based on a case analysis on $\mathsf{st}r$.

- Suppose $\mathsf{st}r$ is *ax*. If $X \vdash s$ occurs in π, then $s = t$. And it follows that $s = t \in X \subseteq \mathsf{st}(X)$.
- Suppose $\mathsf{st}r$ is *pair*. Then $t = (t_0, t_1)$ and π is of the following form:

$$\dfrac{\overset{\pi_0}{\vdots} \atop X \vdash t_0 \quad \overset{\pi_1}{\vdots} \atop X \vdash t_1}{X \vdash t}\,\textit{pair}$$

  Clearly $t_0 \in \mathsf{st}(t)$ and $t_1 \in \mathsf{st}(t)$. Now either $s = t$ or $X \vdash s$ occurs in π_0 or π_1. In the second and third cases, $s \in \mathsf{st}(X \cup \{t_0, t_1\})$, by induction hypothesis. But $\mathsf{st}(X \cup \{t_0, t_1\}) \subseteq \mathsf{st}(X \cup \{t\})$, and hence we are done.
- Suppose $\mathsf{st}r$ is *enc*. Then $t = \{u\}_k$ and π is of the following form:

$$\dfrac{\overset{\pi_0}{\vdots} \atop X \vdash u \quad \overset{\pi_1}{\vdots} \atop X \vdash k}{X \vdash t}\,\textit{enc}$$

  Clearly $u \in \mathsf{st}(t)$ and $k \in \mathsf{st}(t)$. Now either $s = t$ or $X \vdash s$ occurs in π_0 or π_1. In the second and third cases, $s \in \mathsf{st}(X \cup \{u, k\})$, by induction hypothesis. But $\mathsf{st}(X \cup \{u, k\}) \subseteq \mathsf{st}(X \cup \{t\})$, and hence we are done.

[1] This is a relatively simple proof, since our rules are particularly well-behaved. The proof of weak normalization in general is much more involved.

- Suppose $\mathsf{st}r$ is *split*. Then $t = t_i$ and π is of the following form:

$$\dfrac{\begin{matrix}\pi' \\ \vdots \\ X \vdash (t_0, t_1)\end{matrix}}{X \vdash t_i}\,\mathit{split}$$

  Again, either $s = t_i$ or $X \vdash s$ occurs in π'. Since π is normal, π' ends in an elimination rule. Therefore for any $X \vdash s$ occurring in π', $s \in \mathsf{st}(X)$. In particular, $(t_0, t_1) \in \mathsf{st}(X)$ and so $\{t_0, t_1\} \subseteq \mathsf{st}(X)$. Thus for all $X \vdash s$ occurring in π, $s \in \mathsf{st}(X)$.
- Suppose $\mathsf{st}r$ is *dec*. Then π is of the following form:

$$\dfrac{\begin{matrix}\pi_0 \\ \vdots \\ X \vdash \{t\}_k\end{matrix} \quad \begin{matrix}\pi_1 \\ \vdots \\ X \vdash \mathit{inv}(k)\end{matrix}}{X \vdash t}\,\mathit{dec}$$

  Again, either $s = t$ or $X \vdash s$ occurs in π_0 or π_1. Since π is normal, π_0 ends in an elimination rule. Therefore for any $X \vdash s$ occurring in π_0, $s \in \mathsf{st}(X)$. In particular, $\{t\}_k \in \mathsf{st}(X)$ and so $\{t, k\} \subseteq \mathsf{st}(X)$. Further, since $\mathit{inv}(k) \in \mathscr{B}$, π_2 cannot end in an introduction rule (else $\mathit{inv}(k)$ would be a pair or an encryption, which is absurd). Therefore for any $X \vdash s$ occurring in π_1, $s \in \mathsf{st}(X)$. Thus for all $X \vdash s$ occurring in π, $s \in \mathsf{st}(X)$.

◀

2 Polynomial time algorithm for derivability

Fix X_0 and t_0 and consider whether $X_0 \vdash t_0$. Let $\mathsf{st} = \mathsf{st}(X_0 \cup \{t_0\})$, and let $N = |\mathsf{st}|$. We know that if $X_0 \vdash t_0$ is derivable, there is a derivation which only has terms from st. Further, the proof of Theorem 2 even guaranteed that a minimum size proof of $X_0 \vdash t_0$ is normal. A minimum size proof has the further property that for each $s \in \mathsf{st}$, $X_0 \vdash s$ occurs at most once on each branch of the derivation (for otherwise we could short-circuit and obtain a proof of smaller size). Thus minimum size proofs have a bound of N on the length of each branch. Thus the height of each minimum size proof is $\leq N$. This means that we can try to calculate the set of all $t \in \mathsf{st}$ derivable at each level of a proof, starting from the leaves at level 0, and do not need to go beyond level N. This inspires the following algorithm, where $\mathit{derive}(X_0) = \{t \in \mathsf{st} \mid X_0 \vdash t\}$. Lines 4–7 in the algorithm update Y by adding all terms

Algorithm 1 Cubic algorithm for computing $\mathit{derive}(X_0)$

```
1: $i \leftarrow 0$;
2: $Y \leftarrow X_0$;
3: **while** $i <= N$ **do**
4:     $Z \leftarrow Y \cup \{(t_0, t_1) \in \mathsf{st} \mid t_0, t_1 \in Y\} \cup \{\{t\}_k \in \mathsf{st} \mid t, k \in Y\}$;
5:     $Z \leftarrow Z \cup \{t \mid \exists r : (t, r) \in Y \text{ or } (r, t) \in Y\}$;
6:     $Z \leftarrow Z \cup \{t \mid \exists k : \{t\}_k \in Y \text{ and } \mathit{inv}(k) \in Y\}$;
7:     $Y \leftarrow Z$;
8: **end while**
9: **return** $Y$;
```

that can be got from Y by the application of one rule. This operation is split across four lines for ease of formatting. It is clear from height bounds and the subterm property that the algorithm indeed computes $\mathit{derive}(X_0)$. The while loop runs for N iterations and each iteration takes at most $O(N^2)$ time, since we ensure that Y and Z are always subsets of st. Thus the algorithm takes $O(N^3)$ time.

3 Linear time algorithm for derivability

Fix a set of terms X_0 and a formula t_0 for the rest of the section. Let $\mathsf{st} = \mathsf{st}(X_0 \cup \{t_0\})$. Let $N = |\mathsf{st}|$. For any $X \subseteq \mathsf{st}$, $\mathit{derive}(X) = \{t \in \mathsf{st} \mid X \vdash t\}$.

Checking if $X_0 \vdash_{\mathsf{dy}} t_0$ amounts to checking if $t_0 \in \mathit{derive}(X_0)$. In the rest of this section, we describe how to compute $\mathit{derive}(X)$ for any $X \subseteq \mathsf{st}$. We compute $\mathit{derive}(X)$ by a marking procedure which initially marks all elements of X and propagates the marking in a clever manner. Its working is best understood when contrasted with the cubic algorithm from the previous section. The propagation step in that algorithm was to look at each pair of marked terms and mark the result of combining them via some rule. This propagation step itself is repeated many times till no new formula can be marked. In the course of this, the same term t may be "touched" many times – in deriving (t, t'), $\{t\}_k$, etc. The marking in the linear algorithm proceeds differently. When we "process" a marked t, we mark all of its consequences that we can determine at that stage, and do not process it again. For this to work, we need information about t being already marked when we process some other marked t' (so that we can mark (t, t'), for instance, without revisiting t). Towards this, we maintain some auxiliary lists. For instance, for each term t there is a list of pairs whose first component is t. While processing t, we mark each (t, t') in this list such that t' is also marked. We maintain similar lists for other operators and the position of t, as is made clear below.

For every $t \in \mathsf{st}$, we define the following sets.

- $\mathsf{P}_\ell(t) = \{(u, v) \in \mathsf{st} \mid u = t\}$.
- $\mathsf{P}_r(t) = \{(u, v) \in \mathsf{st} \mid v = t\}$.
- $\mathsf{E}_\ell(t) = \{\{u\}_k \in \mathsf{st} \mid u = t\}$.
- $\mathsf{E}_r(k) = \{\{u\}_{k'} \in \mathsf{st} \mid k' = k\}$.
- $\mathsf{D}_r(k) = \{\{u\}_{k'} \in \mathsf{st} \mid k' = \mathit{inv}(k)\}$.

The procedure to compute $\mathit{derive}(X)$ is described in Algorithm 2. For each $t \in \mathsf{st}$ it maintains a variable $\mathit{status}(t) \in \{\mathit{raw}, \mathit{pending}, \mathit{processed}\}$. It also uses a queue Q of formulas, with the corresponding *enqueue* and *dequeue* functions. The correctness of the algorithm is presented below.

▶ **Lemma 4** (Soundness). *If $\mathit{status}(t) = \mathit{processed}$, then $u \in \mathit{derive}(X)$.*

Proof. Initially, the status of every $t \in \mathsf{st}$ is either *raw* or *pending*. It is clear from the code that $\mathit{status}(t)$ becomes *processed* only after becoming *pending*. It is also clear that any formula is enqueued only after it becomes *pending*. We prove by induction that if $\mathit{status}(t)$ becomes *pending* at any stage of the *while* loop, $t \in \mathit{derive}(X)$.

The base case is when $\mathit{status}(t)$ becomes *pending* before the start of the loop. This means that $t \in X$ and hence $t \in \mathit{derive}(X)$.

Suppose $\mathit{status}(t)$ becomes *pending* in some iteration of the loop. We consider a few sample cases that might occur.

***status*(t) changes at line 8:** This means that there is a (t, s) that has just been dequeued, and therefore $\mathit{status}((t, s))$ was *pending* in an earlier iteration, and hence $(t, s) \in \mathit{derive}(X)$. Since one can apply the *split* rule to get t from (t, s), $t \in \mathit{derive}(X)$.

Algorithm 2 Linear time algorithm for $\mathit{derive}(X)$

```
$Q \leftarrow \varnothing$;
$\mathit{for\ all}\ t \in X : \mathit{status}(t) \leftarrow \mathit{pending}; \mathit{enqueue}(Q, t)$;
$\mathit{for\ all}\ t \in \mathsf{st} \setminus X : \mathit{status}(t) \leftarrow \mathit{raw}$;
while $Q \neq \varnothing$ do
    $t \leftarrow \mathit{dequeue}(Q)$;

    if $t = (u, v)$ and $\mathit{status}(u) = \mathit{raw}$ then        ▷ $t$ is the premise of split.
        $\mathit{status}(u) \leftarrow \mathit{pending}$;    $\mathit{enqueue}(Q, u)$;
    end if
    if $t = (u, v)$ and $\mathit{status}(v) = \mathit{raw}$ then        ▷ $t$ is the premise of split.
        $\mathit{status}(v) \leftarrow \mathit{pending}$;    $\mathit{enqueue}(Q, v)$;
    end if
    for all $(t, u) \in \mathsf{P}_\ell(t)$ s.t $\mathit{status}(u) \neq \mathit{raw}$ and $\mathit{status}((t, u)) = \mathit{raw}$ do
        $\mathit{status}((t, u)) \leftarrow \mathit{pending}$;        ▷ $t$ is left premise of pair.
        $\mathit{enqueue}(Q, (t, u))$;
    end for
    for all $(u, t) \in \mathsf{P}_r(t)$ s.t $\mathit{status}(u) \neq \mathit{raw}$ and $\mathit{status}((u, t)) = \mathit{raw}$ do
        $\mathit{status}((u, t)) \leftarrow \mathit{pending}$;        ▷ $t$ is right premise of pair.
        $\mathit{enqueue}(Q, (u, t))$;
    end for

    if $t = \{u\}_k$ and $\mathit{status}(u) = \mathit{raw}$ and $\mathit{status}(\mathit{inv}(k)) \neq \mathit{raw}$ then
        $\mathit{status}(u) \leftarrow \mathit{pending}$;        ▷ $t$ is the left premise of dec.
        $\mathit{enqueue}(Q, t_0)$;
    end if
    for all $\{t\}_k \in \mathsf{E}_\ell(t)$ s.t $\mathit{status}(k) \neq \mathit{raw}$ and $\mathit{status}(\{t\}_k) = \mathit{raw}$ do
        $\mathit{status}(\{t\}_k) \leftarrow \mathit{pending}$;        ▷ $t$ is left premise of enc.
        $\mathit{enqueue}(Q, \{t\}_k)$;
    end for
    if $t = k \in \mathscr{K}$ then
        for all $\{u\}_{\mathit{inv}(k)} \in \mathsf{D}_r(k)$ s.t $\mathit{status}(\{u\}_{\mathit{inv}(k)}) \neq \mathit{raw}$ and $\mathit{status}(u) = \mathit{raw}$ do
            $\mathit{status}(u) \leftarrow \mathit{pending}$;        ▷ $k$ is right premise of dec.
            $\mathit{enqueue}(Q, u)$;
        end for
        for all $\{u\}_k \in \mathsf{E}_r(k)$ s.t $\mathit{status}(u) \neq \mathit{raw}$ and $\mathit{status}(\{u\}_k) = \mathit{raw}$ do
            $\mathit{status}(\{u\}_k) \leftarrow \mathit{pending}$;        ▷ $k$ is right premise of enc.
            $\mathit{enqueue}(Q, \{u\}_k)$;
        end for
    end if

    $\mathit{status}(t) \leftarrow \mathit{processed}$;
end while
return $\{t \in \mathsf{st} \mid \mathit{status}(t) = \mathit{processed}\}$;
```

***status*(t) changes at line 18:** This means that $t = (u, v)$ and v was dequeued in this iteration. Thus $\mathit{status}(v) = \mathit{pending}$ in an earlier iteration, and hence $v \in \mathit{derive}(X)$. The condition on line 17 says that $\mathit{status}(u) \neq \mathit{raw}$, which means that $\mathit{status}(u)$ became *pending* in an earlier iteration, or earlier in this iteration. In either case, it follows that $u \in \mathit{derive}(X)$. We can now apply the *pair* rule to conclude that $t = (u, v) \in \mathit{derive}(X)$.

***status*(t) changes at line 32:** This means that there is a k that was dequeued in this iteration and a $\{t\}_{\mathit{inv}(k)} \in \mathsf{D}_r(k)$ whose *status* is not *raw*. Thus $\mathit{status}(k) = \mathit{pending}$ in an earlier iteration, and hence $k \in \mathit{derive}(X)$. Also $\mathit{status}(\{t\}_{\mathit{inv}(k)}) \neq \mathit{raw}$, so the *status* changed to *pending* in an earlier iteration, or earlier in this iteration. In either case, it follows that $\{t\}_{\mathit{inv}(k)} \in \mathit{derive}(X)$. We can now apply the *dec* rule to conclude that $t \in \mathit{derive}(X)$.

The other cases proceed similarly, and we have the required claim. ◀

▶ **Lemma 5** (Completeness). *If $t \in \mathit{derive}(X)$, eventually $\mathit{status}(t)$ is assigned the value 'processed'.*

Proof. We first prove by induction on size of proofs that if $t \in \mathit{derive}(X)$ then eventually $\mathit{status}(t)$ becomes *pending* (it is enough to show this – if t ever becomes *pending*, it is enqueued, and upon dequeue it gets assigned the status *processed*).

Suppose π is a proof of $X \vdash t$, and let $\mathsf{st}r$ be its last rule. We consider a few sample cases.

$\mathsf{st}r$ is *ax*: In this case, $t \in X$ and $\mathit{status}(t)$ becomes *pending* in line 2.

$\mathsf{st}r$ is *split*: Let the premise of $\mathsf{st}r$ be $X \vdash (u, t)$. Since $X \vdash (u, t)$ has a smaller proof, $\mathit{status}((u, t))$ become *pending* at some point of time, and added to the queue. Eventually it is dequeued. In that iteration of the **while** loop, eventually the condition in line 10 is checked. Either $\mathit{status}(t) \neq \mathit{raw}$ (which means it became *pending* earlier), or it becomes *pending* in line 11.

$\mathsf{st}r$ is *pair*: In this case, $t = (u, v)$ and there are smaller proofs of $X \vdash u$ and $X \vdash v$. So eventually both $\mathit{status}(u)$ and $\mathit{status}(v)$ become *pending*, and they are both enqueued. Eventually they are both dequeued. Without loss of generality, let v be the last dequeued term among u and v. At the time when it is dequeued, $\mathit{status}(u) \neq \mathit{raw}$. In that iteration of the **while** loop, eventually we will check the status of $t = (u, v) \in \mathsf{P}_r(v)$ in line 17. Either $\mathit{status}(t) \neq \mathit{raw}$ (which means it became *pending* earlier), or it becomes *pending* in line 18.

The other cases proceed similarly, and we have the required claim. ◀

▶ **Lemma 6** (Running time). *The algorithm terminates in $O(N)$ time.*

Proof. Each element enters Q at most once (when its status changes from *raw* to *pending*). We process each element of Q exactly once, after dequeuing it and before marking it *processed*. Processing an element involves setting the status of some of its immediate subterms and perhaps enqueueing them (all these operations take constant time). It also involves going through each element of the five sets $\mathsf{P}_\ell(t)$, $\mathsf{P}_r(t)$, $\mathsf{E}_\ell(t)$, $\mathsf{E}_r(t)$, and $\mathsf{D}_r(t)$. Each of these **for all** loops takes at most $O(N)$ time. So it would appear that the algorithm takes $O(N^2)$ overall. But we need to use the following crucial property. For distinct u and v and $f \in \{\mathsf{P}_\ell, \mathsf{P}_r, \mathsf{E}_\ell, \mathsf{E}_r, \mathsf{D}_r\}$: $f(u) \cap f(v) = \varnothing$.

Thus the *total* time spent processing the sets $\mathsf{P}_\ell(t)$ is $O(N)$ *across all* $t \in \mathsf{st}$. And similarly for P_r, E_ℓ, etc. From this it follows that the algorithm terminates in $O(N)$ time. ◀