CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL
Institut für Informatik · AG Software Engineering

Priv.-Doz. Henning Schnoor

Engineering Secure Software Systems Winter 2020/21
Exercise Sheet 3

issued: November 17, 2020 **due:** November 26, 2020

Exercise 3.1, DY closure and derivations (10 Points)

In the lecture, the following lemma was stated (without proof):

If S is a set with $\mathsf{IDs} \cup \{k_a \mid a \in \mathsf{IDs}\} \cup \{\epsilon\} \subseteq S$ and $m \in \mathsf{DY}(S)$, then there is a derivation of m from S: $S = S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} \ldots S_{n-1} \rightarrow_{L_{n-1}} S_n$ with $m \in S_n$.

1. Prove the above lemma.
2. State and prove an appropriate converse of the lemma.

Note: As in the lecture, you can assume that both S and m do not contain applications of hash functions, message authentication codes (MACs), or signatures.

Solution With $D(S)$, we denote the closure of a set S of terms under the application of all rules $L_d(t)$ and $L_c(t)$ as defined in the lecture. For a closure operator $O \in \{\mathsf{DY}(.), D(.)\}$, we say a set of terms S is *O-closed*, if $O(S) = S$. Clearly, $O(S)$ is the smallest O-closed set of terms containing S. To prove the claim, we therefore only need to show that a set is $\mathsf{DY}(.)$-closed if and only if it is $D(.)$-closed. This follows trivially from the definitions, since each rule from the definition of $\mathsf{DY}(.)$ can be translated into a L_d or L_c rule, and vice versa.

Therefore, the smallest $D(.)$-closed set containing S and the smallest $\mathsf{DY}(.)$-closed set containing S are one and the same, and hence $\mathsf{DY}(S) = D(S)$.

1. This follows since, due to the above, $\mathsf{DY}(S) \subseteq D(S)$.
2. This can be formalized as the converse, i.e., $D(S) \subseteq \mathsf{DY}(S)$, and also follows from the above.

Exercise 3.2, minimal derivation properties (10 Points)

In the video lecture on the computation of the Dolev-Yao closure, we proved a lemma characterizing shortest derivations.

1. Can you generalize this result to handle signatures, MACs, and hash functions?
2. Which properties does the modeling of cryptographic primitives have to satisfy for an analog of this result to hold?
3. Can you come up with a modeling of cryptographic primitives where this property does not hold?

Solution The first part of the task follows immediately from the presentation in the lecture, since the proof requires only the "one-step property" of the derivation rules. It is easy to see that the derivation rules for signatures, MACs, and hash functions follow the same pattern:

rule	prerequiste set		result set
$L_c(\mathsf{hash}(t))$	$\{t\}$	$\longrightarrow$	$\{\mathsf{hash}(t)\}$
$L_c(\mathsf{sig}_{k_A}(t))$	$\{\hat{k}_A, t\}$	$\longrightarrow$	$\{\mathsf{sig}_{k_A}(t)\}$
$L_c(\mathsf{MAC}_\mathsf{k}(t))$	$\{k, t\}$	$\longrightarrow$	$\{\mathsf{MAC}_\mathsf{k}(t)\}$
$L_d(\mathsf{sig}_{k_A}(t))$	$\{\mathsf{sig}_{k_A}(t)\}$	$\longrightarrow$	$\{t\}$
$L_d(\mathsf{MAC}_\mathsf{k}(t))$	$\{\mathsf{MAC}_\mathsf{k}(t)\}$	$\longrightarrow$	$\{t\}$

Note that there is no decomposition rule for the hash operator.

As an artificial cryptographic primitive that does not follow these rules, we introduce the MAGIC operator, which "magically" can decrypt ciphertexts encrypted with two layers of symmetric encryption. The rules for the Dolev-Yao closure are:

- if $t \in \mathsf{DY}(S)$, then $\mathsf{MAGIC}(t) \in \mathsf{DY}(S)$,
- if $\mathsf{MAGIC}(\mathsf{enc}^{\mathsf{s}}_{k_1}\left(\mathsf{enc}^{\mathsf{s}}_{k_2}(t)\right) \in \mathsf{DY}(S)$ for any terms k_1, k_2, t, then $t \in \mathsf{DY}(S)$.

The corresponding derivation rules are:

rule	**prerequiste set**		**result set**
$L_c(\mathsf{MAGIC}(t))$	$\{t\}$	$\longrightarrow$	$\{\mathsf{MAGIC}(t)\}$
$L_d(\mathsf{MAGIC}(\mathsf{enc}^{\mathsf{s}}_{k_1}\left(\mathsf{enc}^{\mathsf{s}}_{k_2}(t)\right))$	$\left\{\mathsf{MAGIC}(\mathsf{enc}^{\mathsf{s}}_{k_1}\left(\mathsf{enc}^{\mathsf{s}}_{k_2}(t)\right)\right\}$	$\longrightarrow$	$\{t\}$

This yields a counter-example to the "simplest derivations" lemma: let $S = \left\{\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A)\right\}$, $t = N_A$, and $I = \{k_{AC}\}$. Then the term t is derivable from S with the following sequence:

Clearly, t can be derived from S, but not without first constructing a term that does not appear as a subterm in $S \cup \{t\}$:

$$\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A) \longrightarrow \mathsf{enc}^{\mathsf{s}}_{k_{AC}}\left(\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A)\right) \longrightarrow \mathsf{MAGIC}\left(\mathsf{enc}^{\mathsf{s}}_{k_{AC}}\left(\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A)\right)\right) \longrightarrow N_A.$$

Exercise 3.3, DY algorithm correctness (10 Points)

Prove that the algorithm for computing the DY closure (in its decisional variant DERIVE) as stated in the lecture is correct and runs in polynomial time. As in the lecture, restrict yourself to terms without applications of hash functions, signatures, or message authentication codes (MACs).

Solution By the construction of the algorithm, it is obious that if the algorithm accepts an input (S, t), then $t \in \mathsf{DY}(S)$ holds. We prove the converse. Hence let $t \in \mathsf{DY}(S)$. Then there is a minimal derivation $S = S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} S_2 \rightarrow_{L_2} \ldots \rightarrow_{L_{n-1}} S_n$ with $t \in S_n$. Due to the lemma from the lecture about minimal derivations, we know that each decomposition step in this derivation is a decomposition of a subterm of a term in S or a composition of a subterm of $S \cup \{t\}$. Therefore, the algorithm considers all derivation rules relevant for deriving t, and is thus correct.

The algorithm runs in polynomial time since the number of qualifying terms t' is bound by the input size, and each step clearly can be performed in polynomial time.

Remark: The intuition that one can speed up the algorithm by first performing all decomposition steps and then all composition steps is false. As an example, consider the following instance:

- $S = \left\{\mathsf{enc}^{\mathsf{s}}_{[k_A, k_B]}(N_A), k_A, k_B\right\}$
- $t = N_A$

In this case, in order to obtain the term N_A, the algorithm first needs to perform the composition step $L_c([k_A, k_B])$, which adds the pair $[k_A, k_B]$ to the set S. After this, the decomposition step $L_d(\mathsf{enc}^{\mathsf{s}}_{[k_A, k_B]}(N_A))$ can be performed, which yields the target N_A.