CHRISTIAN-ALBRECHTS-UNIVERSITÄT ZU KIEL Priv.-Doz. Henning Schnoor
Institut für Informatik · AG Software Engineering

Engineering Secure Software Systems Winter 2020/21
Exercise Sheet 10

issued: January 19, 2021 **due:** January 28, 2021

Exercise 10.1, indistinguishability (10 Points)

For the following pairs of terms, determine whether they are I-distinguishable, where $I = \{k_A, k_C, \hat{k}_C, \mathsf{yes}, \mathsf{no}\}$ contains the initial adversary knowledge.

$\mathbf{t_1}$	$\mathbf{t_2}$
$[N_A, \mathsf{enc}^{\mathsf{s}}_{N_A}(N_B)]$	$[N_B, \mathsf{enc}^{\mathsf{s}}_{N_B}(N_A)]$
$[N_B, \mathsf{enc}^{\mathsf{s}}_{N_A}(N_B)]$	$[N_A, \mathsf{enc}^{\mathsf{s}}_{N_B}(N_A)]$
$[N_A, \mathsf{enc}^{\mathsf{s}}_{N_A}(N_B)]$	$[N_A, \mathsf{enc}^{\mathsf{s}}_{N_B}(N_B)]$
$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{yes})$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_B, \mathsf{yes})$
$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{yes})$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{no})$
$[N_A, \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{hash}(N_A), \mathsf{yes})$	$[N_B, \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{hash}(N_B), \mathsf{yes})$
$[N_A, \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{hash}(N_A), \mathsf{yes})$	$[N_B, \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{hash}(N_A), \mathsf{yes})$

Exercise 10.2, strong secrecy and derivation-based secrecy (10 Points)

For an equational theory E, a term t is E-derivable from a set of terms I, if there is a term M built from E-constructors (e.g., encryption functions), E-deconstructors (e.g., decryption functions) and elements from I with $M \equiv_E t$.

Example: Let E model symmetric encryption and pairing, let $I = \{k_{AC}, \underbrace{\mathsf{enc}^{\mathsf{s}}_{k_{AC}}(\mathsf{yes}, N_A)}_{=:u}\}$. Then $t = N_A$ is E-derivable from I via $M = \mathsf{proj}_2(\mathsf{dec}^{\mathsf{s}}_{k_{AC}}(u))$.

Now, the *(nonce) derivation problem* for E is to determine, given a set I of terms and a term (a nonce) t, whether t is E-derivable from I.

Show that if static equivalence for E is decidable, then the nonce derivation problem for E is also decidable.

Note: It suffices to state the (simple) algorithm deciding nonce derivation problem, which may apply the decision algorithm for static equivalence.

Exercise 10.3, secrecy properties and events (10 Points)

In the lecture, two different kinds of (trace) properties were discussed:

- secrecy properties, modeled with derivability of the constant FAIL and in ProVerif using the statement

  query attacker(FAIL),

- event properties, modeled in ProVerif using the specification **event** and queries like

  query x:key; **event(termServer(x))** ⇒ **event(acceptsClient(x))**.

Is one of these concepts more powerful than the other? In other words, can you "translate" any secrecy query into an event quary and/or vice versa? Which, if any, extensions would our theoretical model require to be able to handle event properties?

Note: The point of this exercise is not for you to actually specify a (rather cumbersome) translation, but to conceptually consider the relationships and differences between these two types of properties.

Solution As mentioned in the note, the point of the exercise is more to have some interesting discussion in class than to formally specify a "translation" (i.e., in TCS terms, a reduction). Some points to keep in mind are:

- One can "simulate" events with emitting special messages that contain the keyword **event** and some parameters. The messages must be signed by an honest principal, obviously.
- It is then possible to reason about events that have "happened" using derivability of the corresponding terms. (Note that we only talk about whether the adversary **can** derive a term, it is not required that he actually does that or "wants to" in an actual protocol run.)
- To then check whether an implication "event A is always preceeded by event B" is satisfied, one can proceed as follows:
  - rewrite the protocol as follows:
    1. each time when event B happens, emit a special message (as argued above)
    2. before a participant performs event A, wait for the B-message from the adversary
    3. if event A happens, release the FAIL constant.
  - If this protocol is secure, but the version without waiting for the B-message is insecure, then the implication is not satisfied: Since the "non-waiting version" is insecure, there is a protocol run in which A occurs. Since the "waiting version" is secure, there is no protocol run in which A is preceeded by B. In particular, there is a protocol run in which A occurs, but is not preceeded by B.
  - If the protocol without waiting for the B-message is secure, then event A simply cannot happen and the implication is trivially satisfied.
  - If both versions of the protocol are insecure, then there is a protocol run in which B happens before A, however we do not know whether A is always preceeded by B. This modeling does not allow us to decide implications between events.

However, the implication is not satisfied if and only if there is a a protocol run in which the B-message is not derivable, but the FAIL-message is. Note that the Rusinowitch-Turuani algorithm can be extended to cover this case, since the algorithm can also check, after finding a candidate attack, whether some term (i.e., the B-message) is **not** derivable. So, the algorithm still runs in nondeterministic polynomial time. For those with some background in complexity theory: The key point here is that the derivability algorithm works in **deterministic** polynomial time, so the usual NP/coNP duality issues do not apply. (For example, a sub-algorithm checking whether some protocol **cannot** be attacked can not be performed in NP, unless of course NP and coNP coincide.) Therefore, event implications can be verified using the Rusinowitch-Turuani algorithm with slight modifications.

For the converse, we can reduce to secrecy to an event-based modeling as follows: Simply add a new participant who, on receiving the FAIL-value, triggers an event **FAIL**. Then there is a run in which **FAIL** occurs if and only if there is a run in which the adversary can derive FAIL.

Exercise 10.4, ProVerif modeling of Needham Schroeder (10 Points)

Study the modeling of the Needham Schroeder protocol given in the ProVerif distribution (various models of the protocol can be found in `examples/pitype/secr-auth/`). Which additional properties were modeled compared to our models from the lecture and exercise class?