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

Engineering Secure Software Systems
All Exercise Sheets Tasks Winter 2019/20

1 Exercise 1

exercise sheet issued October 22, 2019, due October 24, 2019 (in class: discussion and warm-up examples)

Exercise 1.1, git account (1 Points)

Register for an exercise group with a partner in the OLAT course for the lecture. Send the username of your *git*-account from the computer science department's git server `https://git.informatik.uni-kiel.de` to Henning Schnoor. In the email, also state

- your name and your git username,
- which lecture you want to register for (Engineering Secure Software Systems),
- and your group number from the OLAT course for the lecture.

You will need the git account during the lecture both for handing in your answers to exercise questions as well as for accessing material such as example source code etc. Using this account to hand in solutions to exercises is mandatory.

Usually, you should have such an account from your Bachelor's studies. If you did not obtain your Bachelor in Kiel or do not have such an account for some other reason, see `http://www.inf.uni-kiel.de/de/service/technik-service/accounts` for details on how to obtain such an account.

Crypto Protocols

2 Exercise 2

exercise sheet issued October 29, 2019, due November 7, 2019

Exercise 2.1, simple example protocol (10 Points)

We consider the following simple authentication protocol:

- Alice sends a message M to Bob, together with her name A,
- Bob answers with a Nonce N_b,
- Alice answers with the term $\mathrm{sig}_{k_A}([M, B, N_b])$.

Please answer the following questions:

1. What are the security properties guaranteed by the protocol?
2. What is the purpose of the nonce N_b? What happens if we omit the nonce?
3. What happens if the B is removed from Alice's last message?

Exercise 2.2, WhatsApp Authentication (10 Points)

The instant messenges service WhatsApp for mobile phones uses the following authorization schemes:

1. To activate an account, the user needs to register a phone number. The system then sends a text message (SMS) over the mobile phone network to the user. The message contains a random number, which the user enters into the app. This activates the account.
2. To mirror the mobile app in a web browser, the user visits a special web page, which displays a QR code. The user then scans this code using the app, and can then access her account from the web interface.

Use informal notation and arguments to specify and discuss the security of the protocols underlying these authentication mechanisms. In particular, consider whether encryption and/or signatures are used in the protocols, which (cryptographic) infrastructure is required to run the protocol, and which assumptions the protocol designers made.

Solution

1. In this protocol, there are two communication channels used:

   - The internet is used for communication between the WhatsApp mobile app and the WhatsApp server use to communicate. We assume that WhatsApp uses TLS to encrypt the messages between app and server. TLS also includes a key derivation protocol (some of you may remember this from the lecture *Betriebssysteme* / *Kommunikationssysteme*). We simplify this by letting the client (i.e., the modile app) to choose a key that is from then on used for communication. The server has a public key k_S.
   - The phone network is used for the short message. In reality, this is what is usually known as *out-of-band communication*, i.e., using a medium that the adversary cannot access. Since our model assumes that the adversary can intercept *all* network communication, we model this network as a symmetric-key encrypted network: We assume that messages on this network are protected with a symmetric key k_{SMS}, that is used for both encryption and authentication. We assume that only Alice has access to this key, in this way we model that a message can be sent on the phone network in such a way that only Alice can read it. For this, we introduce the following notation:

     $$\mathsf{SMS}(t) = \mathsf{MAC}_{k_{SMS}}\left(\mathsf{enc}^{\mathsf{s}}_{k_{SMS}}(t)\right).$$

   With these two channels, the communication steps can be written as follows (A is the app (run by Alice) S is the server, p_A is Alice's phone number, and N_A is Alice's nonce used as key):

   $$\begin{array}{lcll} A & \rightarrow & S & \mathsf{enc}^{\mathsf{a}}_{k_S}\left(p_A, A\right) \\ S & \rightarrow & A & \mathsf{SMS}(\mathsf{sig}_{k_S}(N_S)) \\ A & \rightarrow & S & \mathsf{enc}^{\mathsf{a}}_{k_S}(N_S, N_A, A) \end{array}$$

   The (in)security of this protocol of course depends on the assumptions that we make about k_{SMS}. If the adversary has access to this key (i.e., can interfere with SMS messaging), then the protocol is insecure. If the channel is secure, then so is the protocol.

2.

Exercise 2.3, Fixing Broken Authentication Protocols (10 Points)

Consider the two authentication protocols presented in the exercise class:

a)

1. $A \rightarrow B \quad (A, \mathsf{enc}^{\mathsf{a}}_{k_B}(N_A))$
2. $B \rightarrow A \quad (B, \mathsf{enc}^{\mathsf{a}}_{k_A}(N_A))$

b)

1. $A \to B \quad (\mathsf{enc}^{\mathsf{a}}_{k_B}(N_A), \mathsf{enc}^{\mathsf{a}}_{k_B}(A))$
2. $B \to A \quad (\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, N_B), \mathsf{enc}^{\mathsf{a}}_{k_A}(B))$

Both of these protocols can be attacked with a similar attack as the Needham-Schroeder protocol or the example protocol we covered in the first exercise class. Suggest changes to the protocols that address these problems, and argue why you think your revised versions of the protocols are secure. Be explicit in what "secure" means in this case.

Solution

a) The problem with the protocol is that the identity of the sender is not cryptographically secured. Informally, the protocol can be changed to

1. $A \to B \quad (\mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A))$
2. $B \to A \quad (\mathsf{enc}^{\mathsf{a}}_{k_A}(B, N_A))$

Written as formal instances, the protocol looks as follows:

Alice

1. $\epsilon \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$

Bob

1. $\mathsf{enc}^{\mathsf{a}}_{k_B}(A, x) \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_A}(B, x)$

To prove security of the protocol, we need to be precise about the intended security property. We also want to cover attacks that use different sessions between honest and dishonest participants. We assume the following setup:

- Clearly, the protocol only provides security for Alice (i.e., the initiator role), as she gets the guarantee that Bob was active during the protocol run. Bob does not obtain any guarantee. We formalize security as requiring that the adversary does not learn the value N_A.
- Alice and Bob are honest, Charlie is the adversary
- Alice starts protocol sessions A_B and A_C with both Bob and Charlie (as initiator).
- Bob starts protocol sessions B_A and B_C with both Alice and Charlie (as responder).
- (To be complete, we would also need to consider sessions where Alice is the responder and Bob is the initiator. We leave these to the reader.)
- The "secure" session is A_B, we therefore add a corresponding BREAK-rule to this instance.

We therefore get the following formalization:

A_B

1. $\epsilon \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$
2. $[\mathsf{BREAK}, N_A] \quad \to \quad \mathsf{FAIL}$

A_C

1. $\epsilon \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_C}(A, N'_A)$

B_A

1. $\mathsf{enc}^{\mathsf{a}}_{k_B}(A, x) \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_A}(B, x)$

B_C

1. $\mathsf{enc}^{\mathsf{a}}_{k_B}(C, x') \quad \to \quad \mathsf{enc}^{\mathsf{a}}_{k_C}(B, x')$

At attack now is an execution order and a substitution. First, note that the instance A_C is actually not required, since the nonce N'_A does not appear in any relevant instance, and therefore the adversary can simply generate the term sent by A_C on its own. Also, we can without loss of generality assume that the adversary triggers the FAIL-action of A_B as last action in the attack. Therefore, the execution order is defined by a permutation of the three first steps of the instances A_B, B_A, and B_C. Since the adversary's goal involves opening the decryption $\mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$, and only the instance B_A can do this, we know that A_B must be activated before B_A. It is also clear that the instance B_C does not help the adversary, since this instance expects a term of the form $\mathsf{enc}^{\mathsf{a}}_{k_B}\left(C, x'\right)$. Since we already eliminated the instance A_C above, we know that the term that B_C waits for is generated by the adversary herself. Therefore, she also knows the plaintext and therefore does not need the instance B_C. Hence, only the instances A_B and B_A are relevant, and their order is fixed. Therefore, the execution order is

a) A_B,

b) B_A.

This also fixes the first message sent by A_B, since the adversary must deliver ϵ. To "open" the encryption, the adversary has no other option that to deliver this message to Bob, and from the resulting term she cannot learn the value N_A. Therefore, the protocol indeed guarantees security in the above sense.

3 Exercise 3

exercise sheet issued November 5, 2019, due November 14, 2019

Exercise 3.1, Formal Representation of the Woo Lam Protocol (10 Points)

Study the authentication protocol by Woo and Lam (see slide 22 of the lecture from October 29).

1. Specify the protocol as sequence of receive/send actions, once in the intended execution between Alice and Bob, and once in a form that allows to model the attack introduced in the lecture.
2. Specify the attack on the protocol formally.
3. How can we modify the protocol in order to prevent this attack?

Solution

1. The following is the protocol as r/s actions in the intended setting. We therefore have three instances: Alice, Bob, and the server.

   Alice

   1. $\epsilon \quad \rightarrow \quad A$
   2. $x \quad \rightarrow \quad \mathsf{enc}^{\mathsf{s}}_{k_{AS}}(x)$

   Bob

   1. $A \quad \rightarrow \quad N_B$
   2. $y \quad \rightarrow \quad \mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(A, y\right)$

   Server

   1. $\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(A, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}(z)\right) \quad \rightarrow \quad \mathsf{enc}^{\mathsf{s}}_{k_{BS}}(z)$

2. The protocol setup containing all instances that are present in the attack presented in the lecture is as follows:

   - C is controlled by the adversary, hence the initial adversary knowledge is $I = \{k_{CS}, \mathsf{BREAK}\}$

- There are two protocol instances modeling sessions by honest Bob. Bob acts as responder in both of these instances (i.e., runs the "Bob" program from above). In one instance, Bob expects messages from Alice (we denote this instance with $\text{Bob}_{\text{Alice}}$), in the second instance (denoted with $\text{Bob}_{\text{Charlie}}$) he expects messages from Charlie.
- Similarly, there are two instances modeling the two sessions of the server. In both of these, the server expects Bob in the responder role of the protocol. In one instance (denoted $\text{Server}_{\text{Alice}}$), the server expects Alice as the initiator, in the second one (denoted $\text{Server}_{\text{Charlie}}$), he expects Charlie.
- We do not model a protocol instance containing Alice's rules, since in the attack scenario, no honest participant performs the initiator role.

The attack is against the instance $\text{Bob}_{\text{Alice}}$, since Alice is assumed to be honest (the adversary knowledge does not contain any of Alice's keys): If the protocol run completes, then Bob assumes that Alice in fact participated in the protocol run. Therefore, the FAIL-rule must also be part of the instance $\text{Bob}_{\text{Alice}}$: When the instance $\text{Bob}_{\text{Alice}}$ receives the final message, the protocol is broken.

$\text{Bob}_{\text{Alice}}$

1. $A \rightarrow N_B$
2. $y \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{BS}}(A, y)$
3. $\mathsf{enc}^{\mathsf{s}}_{k_{BS}}(N_B) \rightarrow \mathsf{FAIL}$

$\text{Bob}_{\text{Charlie}}$

1. $C \rightarrow N'_B$
2. $y' \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{BS}}(C, y')$

$\text{Server}_{\text{Alice}}$

1. $\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(A, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}(z)\right) \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{BS}}(z)$

$\text{Server}_{\text{Charlie}}$

1. $\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(C, \mathsf{enc}^{\mathsf{s}}_{k_{CS}}(z')\right) \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{BS}}(z')$

Note that the different instances of Bob use different nonces and variables. Also note that our modeling does not include the message containing the "trash" symbol from the lecture slides, since this clearly cannot be expressed in the formal model (adding a rule involving a trash symbol to the server's program would be very unnatural and would also require a case distinction in the server's modeling). However, modeling this message is not required, since the adversary can simply leave out the corresponding receive/send action from the server instance $\text{Server}_{\text{Alice}}$.

3. The attack on the protocol consists of the execution order and the substitution:
   - The execution order is as follows (the last step is for triggering the FAIL-rule):
     a) $\text{Bob}_{\text{Alice}}$
     b) $\text{Bob}_{\text{Charlie}}$
     c) $\text{Bob}_{\text{Alice}}$
     d) $\text{Bob}_{\text{Charlie}}$
     e) $\text{Server}_{\text{Charlie}}$
     f) $\text{Bob}_{\text{Alice}}$
4. The substitution is easily obtained from the above formalization of the instances and the attack as presented in the lecture:
   - $\sigma(y) = \sigma(y') = \mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)$,
   - $\sigma(z') = N_B$.

5. To demonstrate that Charlie can derive the messages he sends, we list the actually appearing terms in the receive/send actions that are performed by the attack—the terms are obtained by simply replacing the variables appearing in the instances above with their values assigned by σ

step / instance	receive		send
1. $\text{Bob}_{\text{Alice}}$	A	$\rightarrow$	N_B
1. $\text{Bob}_{\text{Charlie}}$	C	$\rightarrow$	N'_B
2. $\text{Bob}_{\text{Alice}}$	$\mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)$	$\rightarrow$	$\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(A, \mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)\right)$
2. $\text{Bob}_{\text{Charlie}}$	$\mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)$	$\rightarrow$	$\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(C, \mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)\right)$
1. $\text{Server}_{\text{Charlie}}$	$\mathsf{enc}^{\mathsf{s}}_{k_{BS}}\left(C, \mathsf{enc}^{\mathsf{s}}_{k_{CS}}(N_B)\right)$	$\rightarrow$	$\mathsf{enc}^{\mathsf{s}}_{k_{BS}}(N_B)$
3. $\text{Bob}_{\text{Alice}}$	$\mathsf{enc}^{\mathsf{s}}_{k_{BS}}(N_B)$	$\rightarrow$	FAIL

Exercise 3.2, Otway Rees Protocol (10 Points)

Consider the following protocol (Otway-Rees-Protocol):

1. $A \rightarrow B \quad [M, A, B, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, M, A, B])]$
2. $B \rightarrow S \quad [M, A, B, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, M, A, B]), \mathsf{enc}^{\mathsf{s}}_{k_{BS}}([N_b, M, A, B])]$
3. $S \rightarrow B \quad [M, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, k_{AB}]), \mathsf{enc}^{\mathsf{s}}_{k_{BS}}([N_b, k_{AB}])]$
4. $B \rightarrow A \quad [M, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, k_{AB}])]$
5. $A \rightarrow B \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\mathsf{FAIL})$

1. Why are the subterms M, A, and B in the second message sent both encrypted and as plaintext?
2. Why is the nonce N_b encrypted in message 2?
3. Is the protocol secure? (You do not need to give a formal proof of security or insecurity.)

Solution The protocol is insecure, since the adversary can fool A into accepting (M, A, B) as the key. This works as follows:

- In Alice's last step, she expects the message $[M, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, k_{AB}])]$.
- To perform the attack, the adversary simply needs to deliver the message $[M, \mathsf{enc}^{\mathsf{s}}_{k_{AS}}([N_a, (M, A, B)])]$ instead.
- The adversary learns the relevant (encrypted) part of this message in the second step of the protocol.

Note that this attack relies on the fact that the principals do not perform type-checking, i.e., Alice is willing to accept a complex term as a key where maybe a nonce would be more natural. Such an attack can therefore be circumvented in a real implementation, however this defense cannot be expressed in our model. (One can of course extend our model with type-checking, however this is not straight-forward when we e.g., discuss type-checking of ciphertexts.)

4 Exercise 4

exercise sheet issued November 12, 2019, due November 21, 2019

Exercise 4.1, Security Modeling Issues: Are we Missing Something? (10 Points)

In the lecture, we defined security of a protocol as, essentially, unreachability of a state in which the adversary learns the constant FAIL. However, this FAIL-constant obviously does not have a correspondance in a real implementation of a protocol. In particular, the rules releasing the FAIL-constant are removed from the protocol in a real implementation. As a consequence, a potential security proof of a protocol in our formal model treats a different protocol than the protocol running in a real implementation.

Are there cases where this difference results in an insecure protocol that can be proven secure in our formal model? If this is the case, how can we circumvent this issue?

Solution This can in fact happen, as an example consider the following protocol: The protocol goal is for Alice and Bob to exchange a secret nonce N_A, the security goal is thus for the adversary to not learn N_A. We assume that there is a PKI in place.

Alice's protocol instance is as follows:

1. $\epsilon \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_B}(N_A)$
2. $[\mathsf{BREAK}, N_A] \rightarrow \mathsf{FAIL}$
3. $\epsilon \rightarrow N_A$.

Bob does not perform any actions in this protocol run, and therefore does not need to be specified. Clearly, assuming that the adversary does not know $\hat{k}_B$, the original protocol is secure, since the adversary can never learn N_A. However, if we remove the second rule, the protocol is completely insecure with respect to the security goal of protecting N_A. A natural way to address this issue is to assume that the BREAK-rules only appear at the end of a protocol instance. In this case, as long as we restrict ourselves to reachability conditions, a secure protocol remains secure when removing the BREAK-rules.

Exercise 4.2, Formal Protocol Model: Features and Omissions (10 Points)

There are a couple of usually assumed properties of cryptographic systems that are not explicitly expressed in our protocol model. Which of the following properties are implicitly expressed in our model, and which are not? Are any of the "omissions" problematic?

- Nonces are indeed used only once, and are freshly generated for each session.
- Private keys are never sent over the network.
- There is a complete PKI available.
- Nonces are long enough so that the adversary cannot guess them correctly.
- The adversary knows the involved algorithms, including the protocol (Kerckhoffs's principle).
- In the absence of an adversary, the network simply forwards the protocol participant's messages as intended.

Exercise 4.3, 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 $t \in \mathsf{DY}(S)$, then there is a derivation of t from S: $S = S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} \ldots S_{n-1} \rightarrow_{L_{n-1}} S_n$ with $t \in S_n$.

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

Note: You can assume that both S and t 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 4.4, minimal derivation properties (10 Points)

In the lecture, we proved a lemma characterizing the shortest derivations (lecture from November 12, 2019). Can you generalize this lemma to also handle signatures, MACs, and hash functions? In general, which properties does the modeling of cryptographic primitives have to satisfy for an analog of this lemma to hold? 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 4.5, 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 $S = S_0 \to_{L_0} S_1 \to_{L_1} S_2 \to_{L_2} \ldots \to_{L_{n-1}} S_n$ 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.

Exercise 4.6, exponential attack size (10 Points)

For $i \in \mathbb{N}$, the protocol P_i is defined as follows:

- There are two instances:
  1. $\mathscr{I}_1$ has a single receive/send action $[x_1, \ldots, x_i] \to \mathsf{enc}^{\mathsf{s}}_k([t_1, t_2])$, with

     $$\begin{aligned} t_1 &= [x_1, [x_2, [x_3, [x_4, [\ldots, [x_{i-1}, [x_i, 0]]\ldots]]]]] \\ t_2 &= [[[[[\ldots[[0, x_i], x_{i-1}], \ldots], x_4], x_3], x_2], x_1]. \end{aligned}$$

  2. $\mathscr{I}_2$ has a single receive/send action $\mathsf{enc}^{\mathsf{s}}_k(y, y) \to \mathsf{FAIL}$.
- The initial adversary knowledge is the set $\{0, 1\}$.

  Show that each protocol P_i is insecure, but a successful attack requires terms of exponential length. How can you use DAGs to obtain a shorter representation of the involved terms?

Solution In order to derive FAIL, the adversary must be able to derive a term of the form $\mathsf{enc}^{\mathsf{s}}_k(t, t)$ for some term t. The only way to generate any term encrypted with the secret key k for the adversary is to supply, via a substitution σ, values for the variables $x_1, \ldots, x_i$ to $\mathscr{I}_1$. As a reply, the adversary receives the term $\mathsf{enc}^{\mathsf{s}}_k([t_1\sigma, t_2\sigma])$. Therefore, a successful attack consists of a substitution σ that satisfies $t_1\sigma = t_2\sigma$ (the execution order consists of the two obvious steps). We denote the subterms of t_1 and t_2 as follows:

- with $t_1^{j\to}$, denote the term $[x_j, [x_{j+1}, \ldots, [x_{i-1}, [x_i, 0]]\ldots]]$,
- with $t_2^{j\to}$, denote the term $[[\ldots[[0, x_i], x_{i-1}], \ldots, x_{j+1}], x_j]$.

By construction, $t_1 = t_1^{1\to}$ and $t_2 = t_2^{1\to}$, further:

- $t_1^{j\to} = [x_j, t_1^{j+1\to}]$ and
- $t_2^{j\to} = [t_2^{j+1\to}, x_j]$.

Assume that σ is a substitution with $t_1\sigma = t_2\sigma$. We show inductively that for all j, we have that $t_1^{j\rightarrow}\sigma = t_2^{j\rightarrow}\sigma$. For $j = 1$, the claim follows since $t_a = t_a^{1\rightarrow}$. Now assume the claim is true for j, i.e., $t_1^{j\rightarrow}\sigma = t_2^{j\rightarrow}\sigma$. By the above, we have

$$t_1^{j\rightarrow} = [x_j, t_1^{j+1\rightarrow}] \text{ and } t_2^{j\rightarrow} = [t_2^{j+1\rightarrow}, x_j].$$

By applying σ to both components of the terms, we obtain

- $\sigma(x_j) = \sigma(t_2^{j+1\rightarrow})$ (left-hand side of the pair), and
- $\sigma(x_j) = \sigma(t_1^{j+1\rightarrow})$ as claimed.

This additionally establishes that for all j, $\sigma(x_j) = \sigma(t_1^{j+1\rightarrow}) = \sigma(t_2^{j+1\rightarrow})$.

We now show inductively that $\left|\sigma(x_j)\right|$ is at least 2^{i-j}. For $j = 1$, this completes the proof. The induction start $j = i$ is clear, as $\sigma(x_i) = 0$. For the induction step, assume that $\left|\sigma(x_j)\right| \geq 2^{i-j}$, we show the claim for $j-1$. By the above, we have that

$$\sigma(x_{j-1}) = \sigma(t_1^{j\rightarrow}) = \sigma(t_2^{j\rightarrow}).$$

By the definition of $t_\cdot^{j\rightarrow}$, we get

$$\sigma(x_{j-1}) = \sigma([x_j, t_1^{j+1\rightarrow}]) = [t_2^{j+1\rightarrow}, x_j].$$

Therefore, $\sigma(x_{j-1}) = [\sigma(x_j), \sigma(x_j)]$, and in particular, $\left|\sigma(x_{j-1})\right| \geq 2\left|\sigma(x_j)\right| \geq 2 \cdot 2^{i-j} = 2^{i-(j-1)}$, as claimed.

5 Exercise 5

exercise sheet issued November 19, 2019, due November 28, 2019

Exercise 5.1, no unique minimal attack (10 Points)

Show that in general, there is no unique minimal attack on a protocol. That is, construct a protocol and two different attacks on it that both have minimal size.

Exercise 5.2, parsing lemma proof (10 Points)

In the proof of the Parsing Lemma, we showed that in that particular setting, the term $\sigma(x)$ is constructed by the adversary. Is this generally true? More formally: Is there a protocol P with initial knowledge I and a successful minimal attack (o, σ) such that there is a variable x with $\sigma(x) \neq x$ and $\sigma(x) \notin \mathsf{DY}(S)$, where S is the set of terms availabe to the adversary at the step where the first term containing $\sigma(x)$ is sent?

6 Exercise 6

exercise sheet issued November 26, 2019, due December 5, 2019

Exercise 6.1, applying the Rusinowitch Turuani Theorem (10 Points)

In the lecture, we discussed how to model the Needham-Schroeder protocol formally as an input to INSECURE such that the attack can be detected. Can you come up with a general mechanism translating a natural representation of a protocol (for example, as the list of "intended instances" for a single session) into an instance that can be used as input for INSECURE? If not, why not?

Exercise 6.2, The FFGG prototocol: too complicated? (10 Points)

Can you come up with a simpler protocol that is secure when only one session is running, but becomes insecure if the adversary can start as many instances as she wishes? Is there an "advantage" of the ffgg protocol (as an example illustrating the need for the analysis of parallel sessions) over your example?

Solution A very simple protocol that requires two sessions in order to be insecure is given by the following receive/send rules:

$$\begin{array}{lll} \text{Alice:} & \epsilon & \rightarrow \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\mathsf{FAIL})\right) \\ \text{Bob:} & \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(x) & \rightarrow \quad x \end{array}$$

Clearly, if the adversary has only one Bob-instance available, she cannot obtain the FAIL-constant, but with two instances, there is an obvious attack. The example can be generalized to requiring n instances of Bob by using encryption nested n times.

The disadvantage of this protocol is that here is is very obvious how to proceed, and simple preprocessing based on the depth of the involved terms could add the additionally required instances. Since in general, insecurity for an arbitrary number of sessions is undecidable (as proven in the lecture), such a simple "preprocessing" does not work in the general case. An example where this does not suffice is of course the protocol used in the lecture's proof of the undecidability result.

Exercise 6.3, unbounded instances formalization (10 Points)

Specify the Needham-Schroeder protocol as an instance of the decision problem UNBOUNDED-INSECURE, and show that it is insecure in this formalization. Discuss the differences between expressing the protocol using this formalism compared to the earlier formalization using the decision problem INSECURE.

Solution We repeat the specification of the protocol:

$$\begin{array}{ll} A \rightarrow B & \mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_a) \\ B \rightarrow A & \mathsf{enc}^{\mathsf{a}}_{k_A}(N_a, N_b) \\ A \rightarrow B & \mathsf{enc}^{\mathsf{a}}_{k_B}(N_b) \end{array}$$

An input to UNBOUNDED-INSECURE needs to contain the following:

- the specification of a "target session" that contains the FAIL-constant,
- the specification of (replicable) sessions for Alice and Bob.

We first specify Bob's session, the one under analysis. This is the same as Bob's usual specification, with the FAIL-rule added. Note that we can assume, without loss of generality, that Bob in this session is interacting with honest Alice.

$$\begin{array}{lll} \mathsf{enc}^{\mathsf{a}}_{k_B}(A, x) & \rightarrow & \mathsf{enc}^{\mathsf{a}}_{k_A}(x, N_B) \\ [\mathsf{BREAK}, x, N_B] & \rightarrow & \mathsf{FAIL} \end{array}$$

We now specify "replicable" sessions for Alice and Bob. Note that we can assume, without loss of generality, that Alice and Bob are the only honest principals, as this is a maximally pessimistic assumption. We define a "pattern" for an initializer session:

parameters public key k_{init} of the identity running this session, name id_{init} matching to the public key, public key k_{resp} of the instance that this initializer wants to connect to. Note that we can assume that k_{init} is Alice's or Bob's key, as the adversary can run Charlie-controlled sessions himself, running number i

session

$$\begin{array}{lcl} \epsilon & \rightarrow & \mathsf{enc}^{\mathsf{a}}_{k_{resp}}\left(id_{init}, N^i_{init}\right) \\ \mathsf{enc}^{\mathsf{a}}_{k_{init}}\left(N^i_{init}, y^i\right) & \rightarrow & \mathsf{enc}^{\mathsf{a}}_{k_{resp}}\left(y^i\right) \end{array}$$

Similarly, we define a "replicable" responder session:

parameters public key k_{resp} of the identity running this session, name id_{init} and key k_{init} of the corresponding communication partner, running number i

session $\mathsf{enc}^{\mathsf{a}}_{k_{resp}}\left(id_{init}, x^i\right) \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_{init}}\left(x^i, N^i_{resp}\right)$

The attack can now easily be represented in this formalization:

- The attacker starts one instance of the initiator session, using parameters $k_{init} = k_A$, $id_{init} = A$, $k_{resp} = k_C$ and $i = 1$.
- The attacker also interacts with Bob's (explicit) session.
- The attacker can now deliver the messages analogous to the standard representation of the protocol.

The main difference between this formalization and the one as input to INSECURE is that in this version, we do not build any information about the attack into the input, but let the adversary start arbitrarily many sessions.

7 Exercise 7

exercise sheet issued December 3, 2019, due December 12, 2019

Exercise 7.1, Missing Proof (10 Points)

Prove the following lemma that was stated in the lecture without proof:

If E is a convergent equational theory, then:

1. For every term t, there is a unique term $[t]$ with
   - $[t]$ is in E-normal-form,
   - $t \equiv_E [t]$.
2. For terms t and t', we have that $t \equiv_E t'$ if and only if $[t] = [t']$.

Solution Let E be a convergent equational theory, let t, and t' be terms (over the term signature corresponding to E).

1. We first prove that a normal form exists. For this, let $t_0 = t$, and inductively define t_{i+1} as an arbitrary term with $t_i \twoheadrightarrow_E t_{i+1}$, where we choose $t_{i+1} \neq t_i$ if possible. Since E is terminating, we know that there is some i such that $t_i = t_j$ for all $j \geq i$. By choice of t_{i+1}, we have that t_i is in E-normal form, and by construction, $t \equiv t_i$.

   Now assume that t_{n_1} and t_{n_2} are both normal forms of t. Then, in particular, $t \twoheadrightarrow^*_E t_{n_1}$ and $t \twoheadrightarrow^*_E t_{n_2}$. Therefore, there exists some t_{nf} with $t_{n_1} \twoheadrightarrow^*_E t_{nf}$ and $t_{n_2} \twoheadrightarrow^*_E t_{nf}$. Since t_{n_1} and t_{n_2} are in normal form, it follows that $t_{n_1} = t_{nf}$ and $t_{n_2} = t_{nf}$. Therefore, $[t] = [t_{nf}]$.
2. First assume that $t \equiv_E t'$. Therefore, there are terms $t = t_0, \ldots, t_n = t'$ such that for each i, we have that $t_i \twoheadrightarrow_E t_{i+1}$ or $t_{i+1} \twoheadrightarrow_E t_i$. We prove the claim by induction over n.

- *Induction Start.* If $n = 0$, then $t = t'$ and the claim holds trivially.
- *Induction Step.* By induction, it follows that $[t_1] = [t_n]$. Hence it suffices to show that $[t_0] = [t_1]$. Without loss of generality, assume that $t_0 \twoheadrightarrow_E t_1$. Then t_1 is a possible step in the construction of the (unique) normal form of t_0 from the proof of the first part above, hence the claim follows.

The other direction is trivial, since if $[t] = [t']$, we have that $t \equiv_E [t] = [t'] \equiv_E t'$.

Exercise 7.2, "Badly-Behaved" Equational Theories (10 Points)

Define equational theories for which the resulting rewrite relation $\twoheadrightarrow_E$ is not a convergent subterm theory, i.e., one that is not confluent, not terminating, or not a subterm theory.

8 Exercise 8

exercise sheet issued December 10, 2019, due December 19, 2019

Exercise 8.1, ProVerif example (10 Points)

Consider the following protocol:

1. $A \to B \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A)$
2. $B \to A \quad [\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_B), N_A]$
3. $A \to B \quad N_B$

Here, k_{AB} is a long-term symmetric key shared by Alice and Bob. Is the protocol secure in the sense, that it can only be completed correctly if both Alice and Bob participate in the protocol run? Analyse the protocol "by hand" and using ProVerif.

Note: If you use the standard ProVerif **query attacker**(FAIL) modeling, you need to express the "participation property" as secrecy property. We will study a different method using events later in the lecture.

Solution First we need to clarify what it means that "both Alice and Bob" participate in the protocol run. Given that the protocol only uses symmetric keys, and identities do not appear in the protocol, this question is not formally well-defined. In particular, if we model this protocol formally, there is not even a notion of an identity that runs the protocol. Therefore, the security property under discussion should be formalized as follows:

- an honest protocol session is one in which the value k_{AB} appears
- we say the protocol is secure if for every honest responder (initiator) session, there is also an honest initiator (responder) session
- more precisely: we define the following two instances, using receive/send rules:

initiator^i

$$\begin{array}{lcl} \epsilon & \to & \mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(N^i_A\right) \\ (\mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(y^i\right), N^i_A) & \to & y^i \end{array}$$

responder^i

$$\begin{array}{lcl} \mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(x^i\right) & \to & (\mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(N^i_B\right), x^i) \\ N^i_B & \to & \epsilon \end{array}$$

We say that an intiator or responder instance is *active* in a protocol run if it completes its second (first) step. The security property then can be formalized as follows: If the responder (initiator) session finishes, then the initiator (responder) session was active as well. This security property is obviously satisfied, since the adversary cannot generate terms of the form $\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(t)$ itself:

- If the initiator instance finishes, then some other instance must have decrypted N_A^i, as this value is received in the second step. In the case that only one instance is running, this "other instance" can only be a responder instance.
- If the responder instance finishes, then some instance must have generated the term received in the first step, which the adversary cannot do. In the case that only one responder instance is running, this must have been an initiator instance.

However, this analysis only considers the case where there is only one initiator and one responder session. A stronger security requirement would be the following: In a protocol with some finite number of initiator and responder sessions (parametrized with different values of i), for every responder (initiator) session that finishes, there is a **unique** initiator (responder) session that is active.

This security property does not hold, as the responder can essentially be used as a "decryption oracle." Therefore, an attack can be performed as follows, using a single intiator instance (A) and three responder instances (B_1, B_2, and B_3):

A: $\epsilon \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A)$

B_1: $\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_A) \rightarrow (\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_{B_1}), N_A)$

at this point, the attacker simply uses the instance B_2 to decrypt N_{B_1}:

B_2: $\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_{B_1}) \rightarrow (\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_{B_2}), N_{B_1})$

this allows the attacker to complete the session B_1:

B_1: $N_{B_1} \rightarrow \epsilon$.

To finish the session B_2, the adversary needs the nonce N_{B_2}. For this, he can simply use the session B_3 in a similar way.

Exercise 8.2, ProVerif examples (10 Points)

Choose a cryptographic protocol and use ProVerif to analyze its security properties, that is:

1. Specify the protocol in ProVerif (including the required cryptographic primitives),
2. specify the security property in ProVerif,
3. run ProVerif to search for attacks. Does the result match with your expectations?

The following is a list of suggestions for protocols to analyse:

- the (broken) authentication protocols presented in the first exercise class
- the repaired versions of the above protocols
- the repaired version of the handshake-protocol from the ProVerif tutorial (the broken version was discussed in the lecture)
- the Needham-Schroeder protocol
- the Needham-Schroeder-Lowe protocol

Exercise 8.3, indistinguishability (10 Points)

Prove that Boolean combinations of I-tests are not necessary. Formally: Show that if messages m and m' are I-distinguishable, then they are distinguishable via an atomic test.

Solution We prove that if m and m' are messages that can be distinguished by a test involving Boolean combinations, then they can also be distinguished using an atomic test. Hence, let φ be a formula that distinguishes m and m', where φ is a (nested) Boolean combinatino of atomic tests. We choose a minimal distinguishing φ with respect to number of occurring Boolean operators. Without loss of generality, assume that $\varphi(m)$ evaluated to true, and $\varphi(m')$ evaluates to false. If φ does not involve any Boolean operators, we are done. Therefore, consider the outmost Boolean operator appearing in φ. We make a case distinction.

- Assume that the operator is negation, i.e., $\varphi = \neg\psi$. Then $\psi(m) = \mathsf{false}$, and $\psi(m') = \mathsf{true}$. Hence ψ is a contradiction to the minimality of φ.
- Assume that the operator is conjunction, i.e., $\varphi = \psi \wedge \chi$. Due to minimality, it follows that $\psi(m) = \psi(m')$ and $\chi(m) = \chi(m')$. Therefore, $\varphi(m) = \varphi(m')$, a contradiction.
- The disjunction case follows with the same argument as the conjunction case.

9 Exercise 9

exercise sheet issued December 17, 2019, due January 9, 2020

Exercise 9.1, indistinguishability II (10 Points)

For the following pairs of terms, determine whether they are I-distinguishable, where $I = \{\hat{k}_C, \mathsf{yes}, \mathsf{no}\}$ 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 9.2, strong secrecy and derivation-based secrecy (10 Points)

Assume an equational theory E. For a set I of terms and a term t, we say t is E-derivable from I, if there is a term M built from E-constructors, E-deconstructors and elements from I with $M \equiv_E t$.

For example, let E model (deterministic) 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

$$M = \mathsf{proj}_2(\mathsf{dec}^{\mathsf{s}}_{k_{AC}}(u)).$$

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 algorithm for the nonce derivation problem.)

Exercise 9.3, secrecy properties and events (10 Points)

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

- secrecy properties, modeled in the theoretical model 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; **inj-event(termServer(x))** $\Rightarrow$ **inj-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 9.4, injective events (10 Points)

Use **inj-event** to analyze the security of a protocol for authenticated message exchange exchange. Proceed as follows:

1. Consider a naive protocol for message exchange that only verifies whether messages come from the intended source.
2. Use **event** to verify the authentication property.
3. Which attacks are possible on the protocol? Use **inj-event** to discover the attack in ProVerif.
4. Correct the protocol and verify its correctness in ProVerif.

Solution A protocol for authenticated message exchange ensures the initiator and the receiver to exchange messages such that the receiver is assured that, if the protocol run is finished correctly, that the message indeed comes from the initiator. In order to be able to obtain assurances about itentities, we assume, as usual, that a PKI is in place. As usual, we assume that Alice is the sender and Bob is the receiver, we denote the message that Alice wants to send to Bob with m.

A first approach for such a protocol could be the following:

$$A \quad \rightarrow \quad B \quad \mathsf{sig}_{k_A}(m)$$

The protocol is trivially secure in the sense that if Bob completes the protocol run successfully, then Alice did indeed send the message m. Using **event**, this is easily confirmed with ProVerif.

However, the protocol is clearly vulnerable against replay-attacks. Hence if we use **inj-event**, ProVerif should be able to determine the protocol's insecurity. As noted by ProVerif, we then also need to place the events under replication in order for the analysis to be non-trivial. After doing this, ProVerif, as expected, finds the error.

To avoid replay-attacks, as usual we let Bob send a challenge nonce to Alice which she then also signs. Hence the new protocol is as follows:

$$\begin{array}{lcll} A & \rightarrow & B & \mathsf{sig}_{k_B}(m) \\ B & \rightarrow & A & N_B \\ A & \rightarrow & B & \mathsf{sig}_{k_B}(N_B) \end{array}$$

Clearly, this protocol is also vulnerable to replay-attacks. It is also important to ensure that in ProVerif, a new nonce is in fact generated in every session, in particular, N_B should not be a global variable. However, this protocol makes another crucial mistake: It lets Alice act as a "signature oracle," which lets the protocol be insecure even with respect to the non-injective event property.

The issue in the protocol is that, in the last step, Alice simply signs anything that Bob sends to her. This obviously needs to be restricted, Alice's signature of N_B must be tied to the message m. A simple approach is to simply let

Alice mention the message m again in her last step (Bob then of course needs to verify that the message in the last step is indeed the same as the one from the first step):

$$\begin{array}{lcll} A & \rightarrow & B & \mathsf{sig}_{k_B}(m) \\ B & \rightarrow & A & N_B \\ A & \rightarrow & B & \mathsf{sig}_{k_B}(N_B, m) \end{array}$$

This protocol successfully verifies in ProVerif. Clearly, sending the message m in the first step is not required, the first step can be replaced by Alice simply opening a session with Bob:

$$\begin{array}{lcll} A & \rightarrow & B & k_A \\ B & \rightarrow & A & N_B \\ A & \rightarrow & B & \mathsf{sig}_{k_B}(N_B, m) \end{array}$$

Exercise 9.5, 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?

10 Exercise 10

exercise sheet issued January 7, 2020, due January 16, 2020

Exercise 10.1, voting protocols (10 Points)

In the lecture, several security properties for voting protocols were discussed. Design a simple voting protocol that satisfies at least one of these properties. Use ProVerif to show that the property is in fact satisfied.

Solution If privacy is not a concern at all, then a very simple property is **verifiability**: Alice wants to be sure that her vote was collected and counted correctly. We can achieve verifiability in a voting protocol as follows:

- We assume, as usual, a PKI in place, such that the election chair Charlie knows the public keys of all registered voters.
- We denote the candidates (i.e., items that the voters can vote for) with $C = \{c_1, \ldots, c_n\}$,.
- The protocol runs as follows:
  - Alice sends her vote c_i to Charlie, signed with her private key:

    $$A \quad \rightarrow \quad C \quad \mathsf{sig}_{k_A}(k_A, c_i)$$

    Charlie simply broadcasts every vote in the network. Hence Charlie consists of the following receive/-send rule (executed under replication):

    $$\mathsf{sig}_{k_A}(x) \quad \rightarrow \quad A \quad \mathsf{sig}_{k_C}(k_A, x)$$

    This modeling works in our single-session model, a ProVerif model of course extracts the public key from the signed message.

    A straight-forward modeling of the security property is then that if Alice receives the confirmation of her vote, then she accepts. However, this modeling would also deem a protocol as secure where Alice waits for a confirmation that can never arrive, we therefore also need to express that it is possible that Alice in fact receives the confirmation.

In the ProVerif modeling, we have the following steps:

version-01 Version that verifies as secure, but only since the verification event is never reached.

version-02 additional query testing whether the verification event is never reached. Since this query verifies true, the verification is in fact never reached, hence there is a bug in the protocol.

version-03 verification fixed, so that it is actually performed, the protocol is secure.

version-04 better modeling with more than one session and random strings as votes.

Exercise 10.2, insecure composition (10 Points)

Specify (either in our formal protocol model or in ProVerif) two cryptographic protocols such that

- each protocol on its own is secure,
- their composition (i.e., a protocol that contains both individual protocols) is insecure.

Can you suggest some design guidelines for protocols to reduce the risk of such "insecure composition?"

Solution A "pathological" example in the formal model is the protocol used in the lecture as an example for ProVerif's incompleteness: The protocol merely contains two instances, each with a single receive/send rule.

$\mathscr{I}_0 \ : \epsilon \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\mathsf{FAIL})\right)$

$\mathscr{I}_1 \ : \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(x) \rightarrow x$

As discussed in the lecture, a protocol containing each instance once (i.e., the protocol $P_1 = (\mathscr{I}_0, \mathscr{I}_1)$ with appropirate initial adversary knowledge) is secure, while the protocol composed with itself $P_2 = (\mathscr{I}_0, \mathscr{I}_1, \mathscr{I}_0, \mathscr{I}_1)$, is not. (The example is in fact a little less pathological than it possibly seems, since self-composition is in fact a key issue in compositional security.) Another example for self-composition leading to security issues is the original Needham-Schroeder protocol, which is secure when only a single session of the protocol is run, but, as discussed in detail in the lecture, cannot be composed safely with itself (depending on the involved identities).

A possibly more convincing, but still pathological example, is the following: Assume that we have two protocols, P_1 and P_2, which each perform some useful function (say, authentication, or electronic voting). Assume that the protocols share a symmetric key k_{AB} and are secure on their own, but each protocol also contains a bug. The bugs in the protocols are harmless on their own, but in combination, the protocols become insecure. In the following model, we omit the actual function of the protocols (imagine some additional receive/send rules in the protocols that do something useful) and only specify the rules that lead to the bug. In the following, let t be a term that does not appear in the protocol description otherwise, say t is a constant that is otherwise unused.

The rules are then as follows:

$r_0 \ : \epsilon \rightarrow \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(t)$

$r_1 \ : \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(t) \rightarrow \mathsf{FAIL}$

Adding these rules to any instances of a secure protocol will clearly lead to an insecure protocol (as long as the rules are added at some "reachable" position in the original protocol). This example also works when the original protocols are secure under self-composition (i.e., the replication operator in ProVerif).

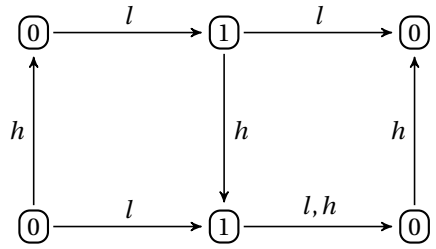
A design rule that reduces the risk of such attacks is to add, to each application of a cryptographic primitive, a tag that references the protocol name as well as the message in the protocol. Clearly, this is difficult to realize in practice due to the amount of coordination required between designers of different protocols.

11 Exercise 11

exercise sheet issued January 14, 2020, due January 23, 2020

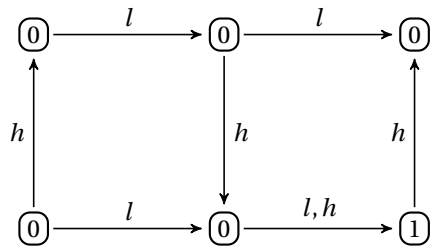
Exercise 11.1, P-Security Example I (10 Points)

Is the following system P-secure? Justify your answer.



Exercise 11.2, P-Security Example II (10 Points)

Is the following system P-secure? Justify your answer.



Exercise 11.3, alternative definition of P security I (10 Points)

Let $M = (S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ be a system and let $\rightarrowtail$ be a policy for M. Prove that the following are equivalent:

1. M is P-secure with respect to $\rightarrowtail$,
2. for all states $s \in S$, all $u \in D$, and all action sequences $\alpha \in A^*$, we have that

$$\texttt{obs}_u(s \cdot \alpha) = \texttt{obs}_u(s \cdot \texttt{purge}_u(\alpha)).$$

Solution The implication from 1 to 2 is clear, since for $\alpha_1 = \alpha$ and $\alpha_2 = \texttt{purge}_u(\alpha)$ we clearly have that $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$, hence the equality follows directly from the definition of P-Security.

Hence assume that the second condition is satisfied, and let $\alpha_1, \alpha_2 \in A^*$ with $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$. It then follows that

$$\begin{array}{lcll} \texttt{obs}_u(s \cdot \alpha_1) & = & \texttt{obs}_u(s \cdot \texttt{purge}_u(\alpha_1)) & \text{(since the second condition is satisfied)} \\ & = & \texttt{obs}_u(s \cdot \texttt{purge}_u(\alpha_2)) & \text{(since } \texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)\text{)} \\ & = & \texttt{obs}_u(s \cdot \alpha_2) & \text{(since the second condition is satisfied)} \end{array}$$

as required.

Exercise 11.4, alternative definition of P security II (10 Points)

An alternative definition of P-security is the following[1]:

- for an agent u, a state s, and an action sequence α, define $\texttt{obs}_u(s \to \alpha)$ as the sequence of observations that u makes when α is performed, starting in state s. Formally:
  - $\texttt{obs}_u(s \to \epsilon) = \texttt{obs}_u(s)$,
  - for a sequence α and an action a, $\texttt{obs}_u(s \to a\alpha) = \texttt{obs}_u(s) \rtimes \texttt{obs}_u(s \cdot a \to \alpha)$,

    (here, $\rtimes$ is string concatenation with elimination of repetitions).
- a system is *secure* if the following holds: For each state s, each agent u and each action sequences α_1, α_2 with $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$, we have $\texttt{obs}_u(s \to \alpha_1) = \texttt{obs}_u(s \to \alpha_2)$.

Show that this definition is equivalent to P-security, i.e., that any system is secure with respect to the above definition if and only if it is P-secure.

Solution First assume that a system is secure with respect to the above definition, let u be an agent, and let α_1, α_2 be traces with $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$. Since the system is secure with respect to the new definition, We know that $\texttt{obs}_u(s \to \alpha_1) = \texttt{obs}_u(s \to \alpha_2)$. Since this sequence contains $\texttt{obs}_u(s \cdot \alpha_1)$ resp. $\texttt{obs}_u(s \cdot \alpha_2)$ as the last entry, it follows that these observations are identical as well; hence the system is P-secure.

For the other direction, assume that a system is not secure with respect to the above definition. It easily follows that then there is a trace α such that α and $\texttt{purge}_u(\alpha)$ form a counter-example. Choose α to be a shortest sequence with this property. In particular, then the last elements of $\texttt{obs}_u(s \to \alpha)$ and $\texttt{obs}_u(s \to \texttt{purge}_u(\alpha))$ differ. These elements are exactly the values $\texttt{obs}_u(s \cdot \alpha)$ and $\texttt{obs}_u(s \cdot \texttt{purge}_u(\alpha))$. Therefore, α and $\texttt{purge}_u(\alpha)$ are a counter-example for P-security of the system.

Exercise 11.5, P-security reduction to two domains (10 Points)

For a system $M = (S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ and a subset of agents $C \subseteq D$, we define the *restriction of M to C* as follows: $M{\restriction}C = (S, s_0, A', \texttt{step}', C, O, \texttt{obs}', \texttt{dom})$, where

- $A' = \{a \in A \mid \texttt{dom}(a) \in C\}$,
- $\texttt{step}'$ is the restriction of $\texttt{step}$ to S and the actions in A',
- $\texttt{obs}'$ analogously is the restriction of $\texttt{obs}$ to S and the agents in C,

For a policy $\rightarrowtail$, the restriction to C is defined as $\rightarrowtail{\restriction}C = \rightarrowtail \cap (C \times C)$.

Prove or disprove the following statement: A system M is P-secure with respect to a policy $\rightarrowtail$ if and only if $M{\restriction}C$ is secure with respect to $M{\restriction}C$ for all $C \subseteq A$ with $|C| = 2$. (Later: Does the corresponding claim hold for IP-security?)

Solution In the following we assume that $|A| \geq 1$, since a system with only one agent is trivially secure.

The statement does not hold, not even for P-security.

- The first direction holds: First assume that M is P-secure with respect to $\rightarrowtail$, and let $C \subseteq A$ with $|C| = 2$, i.e., $C = \{u, v\}$ for two agents. We show that $M{\restriction}C$ is P-secure with respect to $\rightarrowtail{\restriction}C$. We show security from u's point of view. Hence, without loss of generality, assume that $v \not\rightarrowtail u$, since otherwise security is trivial. Now let s be a state, and let α_1 and α_2 be sequences of actions of $M{\restriction}C$, i.e., sequences of actions with domain u or v, and such that $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$. Since α_1 and α_2 are also action sequences for the original system M, and clearly the $\texttt{purge}$-function of both systems (policies) behaves identically for sequences containing only actions of u and v, P-security of M implies that $\texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot \alpha_2)$.

[1] as usual, we fix a noninterference policy $\rightarrowtail$

- For the second direction, a counter-example can be obtained easily by constructing a system that has an "insecure" part (subsystem) that only can be reached when three distinct agents perform an action. The same counter-example of course can be used for IP-security.

Exercise 11.6, P-security and non-transitive policies (10 Points)

Prove or disprove the following: If $M = (S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ is a system and $\rightarrowtail$ is a policy for M, then the following are equivalent:

- M is P-secure with respect to $\rightarrowtail$,
- M is P-secure with respect to the transitive closure of $\rightarrowtail$.

Solution The characterization is clearly not correct. Assume a policy $\rightarrowtail = A \rightarrow B \rightarrow C \rightarrow A$. Since the transitive closure of $\rightarrowtail$ is the complete relation on the agents $\{A, B, C\}$, any system is trivially P-Secure with respect to the transitive closure. We now show that there is a system which is not secure with respect to $\rightarrowtail$ itself. For this, let M be any system that is not secure with respect to the policy $H \not\rightarrowtail L$. We can easily translate this system by mapping H's actions and observations to A, and L's actions and observations to B, without adding any actions for B. Then trivially, the system is insecure with respect to $\rightarrowtail$.

12 Exercise 12

exercise sheet issued January 21, 2020, due January 30, 2020

Exercise 12.1, implications between security properties (10 Points)

In the lecture, some implications between security definitions were stated without proof. Choose and prove one of the following (in the following, M is a system and $\rightarrowtail$ a policy).

1. If M is TA-secure with respect to $\rightarrowtail$, then M is also IP-secure with respect to $\rightarrowtail$.
2. If M is P-secure with respect to $\rightarrowtail$, then M is also TA-secure with respect to $\rightarrowtail$.

Solution

1. The proof is from the full version of [Mey07]. Assume that M is TA-secure with respect to $\rightarrowtail$, we show that IP-security holds as well. Hence let s be a state, let u be an agent, and let α_1, α_2 be action sequences with $\texttt{ipurge}_u(\alpha_1) = \texttt{ipurge}_u(\alpha_2)$. To show that $\texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot \alpha_2)$, it suffices to prove that $\texttt{ta}_u(\alpha_1) = \texttt{ta}_u(\alpha_2)$, then TA-security of the system implies that both have the same observations. To show this, it suffices to prove that, for all $X \subseteq D$ with $u \in X$, we have that $\texttt{ta}_u(\alpha) = \texttt{ta}_u(\texttt{ipurge}_X(\alpha))$. We show the claim by induction on α. If $\alpha = \epsilon$, the claim is trivial. Hence assume the claim is true for α, we consider αa for some action $a \in A$. As usual, we consider two cases:
   - If $\texttt{dom}(a) \not\rightarrowtail u$, then, by definition, $\texttt{ta}_u(\alpha a) = \texttt{ta}_u(\alpha)$. We consider two subcases, writing $v \rightarrowtail X$ is $v \rightarrowtail w$ for some $w \in X$.
     - case 1: $\texttt{dom}(a) \not\rightarrowtail X$. Then $\texttt{ipurge}_X(\alpha a) = \texttt{ipurge}_X(\alpha)$. Hence

$$\begin{array}{lcll} \texttt{ta}_u(\texttt{ipurge}_X(\alpha a)) & = & \texttt{ta}_u(\texttt{ipurge}_X(\alpha)) & \\ & = & \texttt{ta}_u(\alpha) \text{ (by induction)} & \text{as required.} \\ & = & \texttt{ta}_u(\alpha a), & \end{array}$$

– case 2: $\texttt{dom}(a) \rightarrowtail X$. Then $\texttt{ipurge}_X(\alpha a) = \texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)\cdot a$. Hence, it follows that

$$\begin{array}{rcl}\texttt{ta}_u(\texttt{ipurge}_X(\alpha a)) & = & \texttt{ta}_u(\texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)\cdot a)\\ & = & \texttt{ta}_u(\texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)) \text{ (recall that } \texttt{dom}(a) \not\rightarrowtail u)\\ & = & \texttt{ta}_u(\alpha) \text{ (by induction)}\\ & = & \texttt{ta}_u(\alpha a).\end{array}$$

- if $\texttt{dom}(a) \rightarrowtail u$, then $\texttt{ipurge}_X(\alpha a) = \texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)\cdot a$. Thus

$$\begin{array}{rcll}\texttt{ta}_u(\texttt{ipurge}(\alpha a)) & = & \texttt{ta}_u(\texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)\cdot a) & \\ & = & (\texttt{ta}_u(\texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)), \texttt{ta}_{\texttt{dom}(a)}(\texttt{ipurge}_{X\cup\{\texttt{dom}(a)\}}(\alpha)), a) & \\ & = & (\texttt{ta}_u(\alpha), \texttt{ta}_{\texttt{dom}(a)}(\alpha), a) \text{ (by induction)} & \text{as required.}\\ & = & \texttt{ta}_u(\alpha a), & \end{array}$$

A more intuitive, but less formal, argument is as follows: As above, it suffices to show that if $\texttt{ipurge}_u(\alpha_1) = \texttt{ipurge}_u(\alpha_2)$, then $\texttt{ta}_u(\alpha_1) = \texttt{ta}_u(\alpha_2)$. To see this, consider each action a appearing in the sequence α, i.e., let $\alpha = \beta a \gamma$ that is removed by $\texttt{ipurge}_u$. This happens when there is no "downgrading chain" from $\texttt{dom}(a)$ to u in the sequence $a\gamma$. In this case, the event a is also not "forwarded" to u in the application of the ta-function. Therefore, the event a is not relevant to the computation of $\texttt{ta}_u(\alpha)$, and the value $\texttt{ta}_u(\alpha)$ does not change when we remove a from the sequence, i.e., we have that $\texttt{ta}_u(\alpha) = \texttt{ta}_u(\beta\gamma)$. Inductively, this shows that $\texttt{ta}_u(\alpha_1) = \texttt{ta}_u(\alpha_2)$.

2. Assume that M is P-secure, and let α_1, α_2 be sequences with $\texttt{ta}_u(\alpha_1) = \texttt{ta}_u(\alpha_2)$. Clearly, $\texttt{ta}_u(\alpha)$ contains at least as much information about α as $\texttt{purge}_u(\alpha)$ does. In particular, $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$, and from P security it follows that $\texttt{obs}_u(s\cdot\alpha_1) = \texttt{obs}_u(s\cdot\alpha_2)$.
3. This proof was done in the lecture.

Exercise 12.2, equivalence for transitive policies (10 Points)

Show that for transitive policies, P-security, IP-security, and TA-security are equivalent. More formally: Let M be a system, and let $\rightarrowtail$ be a transitive policy. Show that the following are equivalent:

1. M is P-secure with respect to $\rightarrowtail$,
2. M is TA-secure with respect to $\rightarrowtail$,
3. M is IP-secure with respect to $\rightarrowtail$,

Solution We know that the implications P-security to TA-security, and TA-security to IP-security, always hold. It therefore remains to show that for a transitive policy, IP-security implies P-security. Hence assume that $\rightarrowtail$ is a transitive policy, and that M is IP-secure. To show that M is also P-secure, let s be a state, let u be an agent, and let α_1, α_2 be traces with $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$. Since M is IP-secure, it suffices to show that $\texttt{ipurge}_u(\alpha_1) = \texttt{ipurge}_u(\alpha_2)$. To show this, we first prove the following claim: If v is an agent such that an action a with $\texttt{dom}(a) = v$ appears in α, then $v \in \texttt{sources}(\alpha, u)$ if and only if $v \rightarrowtail u$. We show the left-to-right implication inductively over α:

- In the base case $\alpha = \epsilon$, there is no such v, and the claim holds.
- Assume the claim holds for α, and let a be an action. We consider two cases:
  1. Assume there is some $v \in \texttt{sources}(\alpha, u)$ with $\texttt{dom}(a) \rightarrowtail v$. In this case:
     – $\texttt{sources}(a\alpha) = \texttt{sources}(\alpha, u) \cup \{\texttt{dom}(a)\}$,
     – by induction, we know that $v \rightarrowtail u$. Since $\rightarrowtail$ is transitive, it also follows that $\texttt{dom}(a) \rightarrowtail u$.

     In both cases, the claim follows.
  2. If such a v does not exist, we have that $\texttt{sources}(a\alpha, u) = \texttt{sources}(\alpha, u)$, and the claim follows by induction.

The right-to-left implication is trivial, and holds for intransitive policies as well.

We now use this result to show that $\texttt{ipurge}_u(\alpha_1) = \texttt{ipurge}_u(\alpha_2)$. In fact we show that $\texttt{ipurge}_u(\alpha) = \texttt{purge}_u(\alpha)$ for all sequences α. This again easily follows by induction:

- If $\alpha = \epsilon$, the claim is trivial.
- Assume the claim holds for α, we consider the trace $a\alpha$. Again, there are two cases to consider:
  - If $\texttt{dom}(a) \in \texttt{sources}(a\alpha, u)$, then by the above we have that $\texttt{dom}(a) \rightarrowtail u$. Therefore, $\texttt{ipurge}_u(a\alpha) = a\texttt{ipurge}_u(\alpha) = a\texttt{purge}_u(\alpha) = \texttt{purge}(a\alpha)$. (The equalities follow from the definition of ipurge, induction, and the definition of purge.)
  - If $\texttt{dom}(a) \notin \texttt{sources}(a\alpha, u)$, then by the above $\texttt{dom}(a) \rightarrowtail u$. Therefore, analogously to the above, we have that $\texttt{ipurge}_u(a\alpha) = \texttt{ipurge}_u(\alpha) = \texttt{purge}_u(\alpha) = \texttt{purge}(a\alpha)$.

13 Exercise 13

exercise sheet issued January 28, 2020, due never (you can use these for exam preparation)

Exercise 13.1, uniqueness of unwindings (10 Points)

Show that P-unwindings are not unique, but that mininal P-unwindings are, that is:

1. give an example for a system M and a policy $\rightarrowtail$ such that there are (at least) two different P-unwindings for M and $\rightarrowtail$,
2. show that if M is P-secure with respect to a policy $\rightarrowtail$, then there is a P-unwinding for M and $\rightarrowtail$ that is contained (via set inclusion) in all P-unwindings for M and $\rightarrowtail$.

Solution

1. Simply choose any non-trivial P-secure system M with a correspondings policy $\rightarrowtail$, and a corresponding unwinding $(\sim)_{u\in D}$. Then modify the system M such that all observations are identical in all states. We immediately obtain two unwindings for the new system M' and $\rightarrowtail$:
   - the unwinding for M clearly still is an unwinding for M',
   - since in M', all agents have the same observations in all states, the universal relation $S \times S$ is trivially an unwinding.
2. We prove the following fact:

   > Let I be a non-empty set, and for each $i \in I$, let $\sim_i = (\sim_u^i)_{u\in D}$ be a P-unwinding for M and $\rightarrowtail$.. Then the family $(\sim_u)_{u\in D}$ defined as $\sim_u = \cap_{i\in I} \sim_u^i$ also is a P-unwinding for M and $\rightarrowtail$.

   This obviously completes the proof, since the minimal unwinding is then simply the intersection of all unwindings. Except for output consistence, the proof is an easy consequence of the fact that closure properties are invariant under intersection. Let $u \in D$, and let $\sim_u$ be defined as in the lemma. We prove that $\sim_u$ in fact satisfies all required properties: That $\sim_u$ is an equivalence relation (i.e., reflexive, symmetric, and transitive), and that $\sim_u$ satisfies output consistency, step consistency, and left respect (in their respective P-security versions).

   - *Reflexivity.* Since each $\sim_u^i$ is reflexive, we have that, for each $s \in S$, $(s, s) \in \sim_u^i$, hence $(s, s) \in \cap_{i\in I} \sim_u^i = \sim_u$.
   - *Symmetry.* Let $s \sim_u t$. By definition, then $(s, t) \in \cap_{i\in I} \sim_u^i$, i.e., $s \sim_u^i t$ for each $i \in I$. Since each $\sim_u^i$ is symmetric, it follows that $t \sim_u^i s$ for each $i \in I$, therefore, $(t, s) \in \cap_{i\in I} \sim_u^i = \sim_u$.

- *Transitivity.* Let $s \sim_u t$ and $t \sim_u r$. As above, we then have $s \sim_u^i t$ and $t \sim_u^i r$ for each $i \in I$, the transitivity of each $\sim_u^i$ then implies that $s \sim_u^i r$ for each i, and thus $s \sim_u r$.
- *Output Consistency.* Let $s \sim_u t$. Since $i \neq \emptyset$, there is some $i \in I$. By definition of $\sim_u$, it follows that $s \sim_u^i t$. Since $\sim_u^i$ satisfies output consistency, it follows that $\mathtt{obs}_u(s) = \mathtt{obs}_u(t)$ as required.
- *Step Consistency.* Let $s \sim_u t$, and let $a \in A$. Since each $\sim_u^i$ satisfies step consistency, we have that $s \cdot a \sim_u^i t \cdot a$ for each $i \in I$, i.e., $(s \cdot a, t \cdot a) \in \cap_{i \in I} \sim_u^i$, therefore, $s \sim_u t$.
- Left Respect. Let $\mathtt{dom}(a) \not\mapsto u$, and let $s \in S$. Since each $\sim_u^i$ satisfies left respect, it follows that $s \sim_u^i s \cdot a$ for all $i \in I$, and hence again, $s \sim_u s \cdot a$.

Exercise 13.2, strong secrecy and non-interference (10 Points)

As suggested by the ProVerif keyword **noninterf**, strong secrecy of cryptographic protocols and non-interference (in the information-flow sense) are related. In this exercise, we will make this relationship more precise. For this, use a simple cryptographic protocol and construct a system M such that strong secrecy of the protocol directly corresponds to P-security of the system M.

Note: Depending on the protocol you choose to model, a finite state system might require a mechanism to limit the possible number of terms and thus the state space. To avoid this, you may use systems with an infinite state space to solve this task.

Solution The idea is to let H send a value (say, 0 or 1) using strong, randomized encryption. Since in the P-security model, L must be unable to detect whether H performed any action at all, we cannot simply let H perform a "send action" as in a natural modeling of the protocol. Instead, some value must be sent regardless of what H does (if anything), but H can influence that value by sending some special message to the participant sending the ciphertext.

Then the system can be modeled as L's observations being modulo the equational theory (i.e., indistinguishability).