Lecture Notes

Engineering Secure Software Systems

Winter 2020/21

Priv.-Doz. Dr. Henning Schnoor

Christian-Albrechts-Universität zu Kiel

Working Group Software Engineering, Institut für Informatik

Version from 15. Februar 2021

About this Document

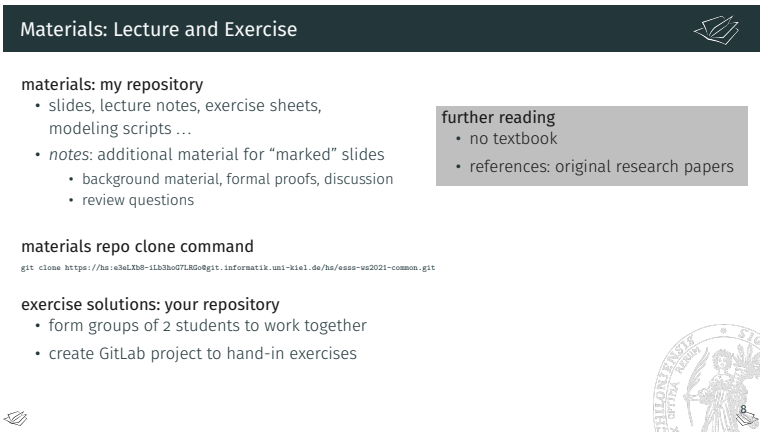
These are in fact merely lecture *notes*, and not a complete script. These notes contain technical content that does not fit on the slides, and which I present on the whiteboard during class, such as longer formal examples and proofs. There is also some background information and additional discussion that I do not have enough time for in the lecture. I also try to add answers to questions asked during the lecture, please let me know if there are any omissions. At the end of the document, you will find the solutions to (most of) the exercises we discussed this semester.

The structure of the notes follows the lecture slides, but only contains those slides from the lecture for which additional material is presented here.

Contents

1 Admin

1.1 Accounts, Materials



Some slides (like this one, as a demo) have a little book symbol in three of the corners (some of these might be covered by text). These slides are accompanied by additional material here in the lecture notes. This material can be clarifications, background information, formal content that is better presented in script form than on slides, or review questions. Most review questions are "immediate" questions which you can try to answer directly after the lecture, other questions try to help you connecting issues mentioned early in the lecture with the solutions presented later and therefore can only be answered once you know the later material. Both types of questions are indicated as such. Some of the "immediate" questions will be discussed in the lecture or the exercise class.

Exercise Task: *exercise git project* Create a git project together with your exercise partner at `https://git.informatik.uni-kiel.de` using the naming scheme `LL-SEM-Lastname1-Lastname2` and add Henning Schnoor (username hs) as a **Maintainer** to your project. Usually, you should have 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. In the project name, `LL` is an abbrevation for the lecture, (e.g., **SEPVS** for Software Engineering für Parallele und Verteilte Systeme or **ESSS** for Engineering Secure Software Systems), **SEM** is an abbrevation for the semester, like **WS20** for Winter 2020/2021. Lastname1 and Lastname2 are the last names of the two students in the working group. Write an email to Henning Schnoor with the URL of the repository (it suffices for one student in each group to write this mail).

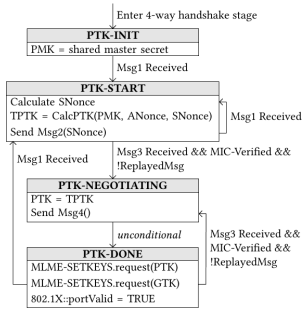
Handing in of exercises and feedback to your tasks will use this git account. For non-programming exercises, answers must be submitted in one of the formats pdf, markdown, or plain text.

1.2 Distance Learning in ESSS 20/21

2 Overview and Motivation

2.1 Introduction

Handshake Specification II



25

Here, the protocol is formalized as a sequence of messages and state diagram (i.e., a finite automaton or a "UML-Zustandsautomat" we discussed in Softwaretechnik).

Review questions during semester:

The notation used here also has similarities with finite automata as familiar from Theoretical Computer Science. What is the key difference between the automata used here and the finite automata used in theory? Why might this difference be relevant to the goals of this lecture, i.e., for automatic security analysis of protocols?

Review questions after semester:

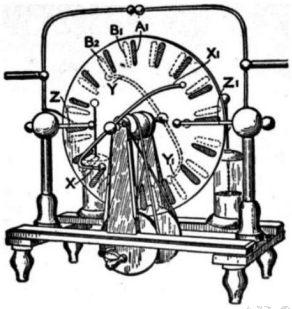
It is helpful to compare this formalization to the formal model we introduce later in the lecture. Which model do you think is more expressive? What is the motivation for our choise of formal model, compared to e.g., "Zustandsautomaten"?

Ideal Situation

push-button tool
- input system, security criterion
- output SECURE or INSECURE

this lecture
how close can we come (for two areas)?

first question
what does "secure" mean? need formal model!



32

Review questions during semester:

What do you think are the main obstacles—both conceptual and practical—for a "push-button-tool" to analyse security? What kind of **formal** analysis methods can you imagine to

be helpful? What do you think is the main difference between formal and informal tools in this context?

2.2 Topic Overview: Three Areas in a Nutshell

2.2.1 Crypto Protocols

Cryptographic Protocols

what are crypto protocols?

- application of crypto primitives to provide "secure" services
- examples: Diffie-Hellman, Internet Key Exchange, TLS, …

Needham-Schroeder protocol

$A \to B \quad \mathrm{enc}^{\mathrm{a}}_{k_B}(A, N_A)$
$B \to A \quad \mathrm{enc}^{\mathrm{a}}_{k_A}(N_A, N_B)$
$A \to B \quad \mathrm{enc}^{\mathrm{a}}_{k_B}(N_B)$

The Needham-Schroeder protocol shown on this slide is a classic "running example" for formal protocol security literature. As such, we will discuss it in depth in this lecture.

Review questions during semester:

- Can you annotate the individual messages with comments as for the protocol on the previous slide ("prove this," etc.)?
- On this slide, the Needham-Schroeder protocol is presented in a classic "Alice-and-Bob-notation." While easily readable, this presentation is not suited for formal analysis. Can you see drawbacks of this notation (even if we accept that we will always use an abstract view of cryptography, i.e., will always just reference "asymmetric encryption" instead of a specific algorithm like 4096-bit-RSA or 2048-bit-ElGamal)?

2.2.2 Information Flow

Declassification

requirement

`top secret` data may not influence `public` data

practice

need exceptions

- information flow needed in some cases
- which ones?
- how do we make sure there are no other exceptions?

Review questions during semester:

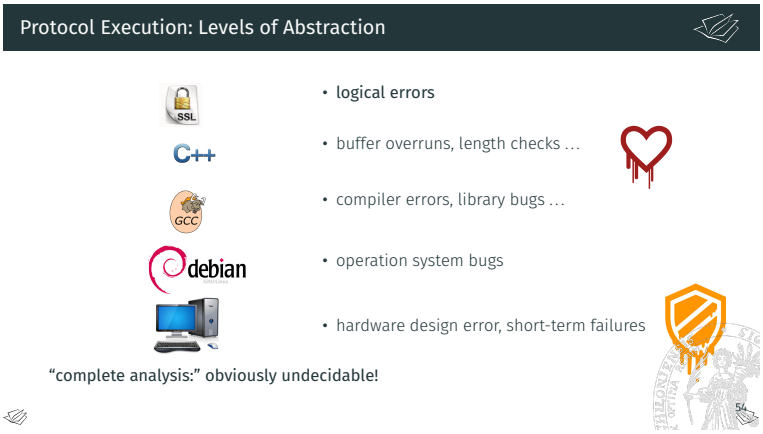
The motivation for the strict "no information flow" condition is straight-forward: If "secret" information has any influence on "publicably observable" information, then in some cases it might be possible to derive some knowledge about the "secret information" from facts that are available to the public, which means that the "secret information" does not remain entirely secret. However, in many cases, requiring that "secret" information may not influence "public" information at all is much too strict. Can you think of real-world cases (e.g., from politics, science, or industry) where this strict idea of information-flow security needs to be relaxed?

2.3 Learning Goals

2.4 Lecture Overview

3 Part I: Crypto Protocols

3.1 Foundations



Security is obviously relevant on many levels, from hardware to implementation details to conceptual errors. In the majority of this lecture, we focus on a very high level of abstraction. There are two main reasons for this: First, on this level we are most likely to encounter the issues that are actually specific to security. The second reason is a more practical one: Most issues on the "lower" levels will already be undecidable except for very specialized cases, so we cannot hope to perform a complete automatic analysis.

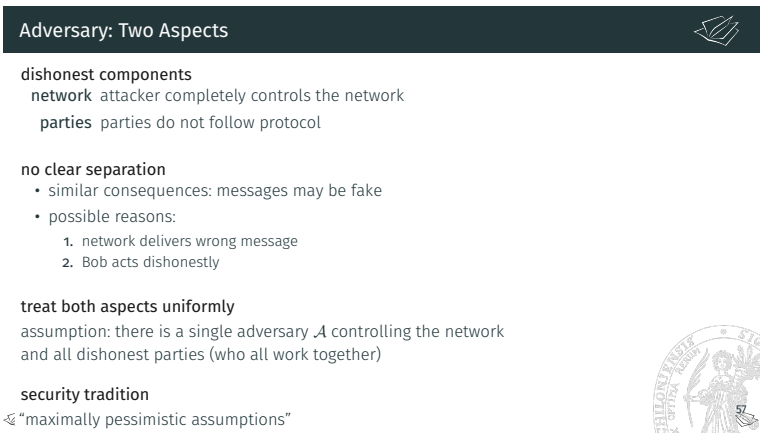
The decision to focus on this level of abstraction has several consequences. The most problematic one is that, using this approach, we make a key assumption: In this lecture, we always assume that the protocols we consider are performed by the machine *exactly* as specified, unless the machine is completely controlled by the adversary. That is, we adopt a pretty simplistic black-and-white world-view of machine- and user corruption: As soon as some participant does not completely control her machine, or does not completely follow the intended protocols honestly, then we assume that the user (and her machine) is completely under control of the adversary. All aspects of protocol execution that happen outside of the

machines controlled by honest participants are treated as completely unreliable (i.e., under control of the adversary).

The results presented in the information-flow part of the lecture are not tied to a specific level of abstraction and can be used to e.g., model issues like Meltdown.

Review questions after semester:

Why is there a difference between the protocol results and the information-flow results with regard to the level of abstraction?



In particular, since our adversary controls the entire network, she can perform the usually studied network attacks:

kill prevent message delivery

sniff obtain copy of message

intercept sniff & kill

re-route deliver message to wrong recepient

delay deliver message too late

reorder deliver messages out of order

replay deliver message multiple times

fake deliver fake message

3.2 Cryptography

Cryptographic Primitives: Encryption

two cases

symmetric (AES, DES, …) single key k for encryption and decryption

- encrypt: $X \times K \to Y$
- decrypt: $Y \times K \to X$

$$\mathsf{dec}^{\mathsf{s}}_k\left(\mathsf{enc}^{\mathsf{s}}_k(x)\right) = x$$

asymmetric (RSA, ElGamal, …) key k_A for encryption, $\hat{k}_A$ for decryption

- encrypt: $X \times K \to Y$
- decrypt: $Y \times \hat{K} \to X$
- mapping: $\hat{}$: $K \to \hat{K}$

$$\mathsf{dec}^{\mathsf{a}}_{\hat{k}_A}\left(\mathsf{enc}^{\mathsf{a}}_{k_A}(x)\right) = x$$

security property in both cases

without decryption key: **no** information about plaintext x obtainable from ciphertext y.

In this slide, the sets X, Y, K and $\hat{K}$ denote the sets of plaintexts, ciphertexts, public/shared keys and secret keys, respectively. The assumption that given the ciphertext without the key it is impossible to obtain "any information" about the plaintext is of course very vague. We will formalize this later in the lecture, using two notions. The first one (Dolev-Yao derivability) is conceptually simple and suffices as long as the "secrets" we consider are cryptographic keys, random numbers and similar technical terms. To also model cases where the adversary has some prior knowledge about the possible communication (e.g., she knows that the plaintext is one word from a small set, known to the attacker), we later introduce equational theories and the concept of indistinguishability.

Cryptographic Primitives: Signatures

two cases

symmetric (MAC) (CMAC, …) single key k_{AB} for "signing" and verification

- sign: $mac: X \times K \to T$
- verify: $test: X \times K \times T \to \{\mathtt{ok}, \mathtt{error}\}$

$$test(x, k, mac_k(x)) = \mathtt{ok}$$

asymmetric key (RSA, ElGamal, …) $\hat{k}_a$ to sign, k_a to verify

- sign: $sig: X \times \hat{K} \to S$
- verify: $test: X \times K \times S \to \{\mathtt{ok}, \mathtt{error}\}$
- mapping: $\hat{}$: $K \to \hat{K}$

$$test(x, k, \mathsf{sig}_{\hat{k}}(x)) = \mathtt{ok}$$

security property in both cases

without signing key: **impossible** to generate accepted signature t / s

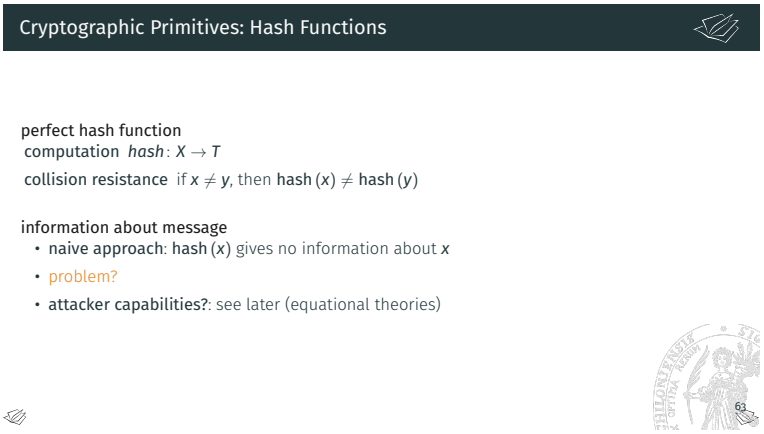
Similarly to encryption, X denotes the message (plaintext) set, K and $\hat{K}$ are the sets of symmetric/public keys, and $\hat{K}$ is the set of private keys. The sets T and S are the sets of message tags and signatures, respectively.

Note that in our later formal term notation, we will write $\mathsf{sig}_{k_A}(m)$ for a message m which is signed by Alice. This might seem unnatural, since this representation references the *public* key of Alice, even though the private key is needed and used to actually compute the signature in a real algorithm (and obviously in our formal model knowledge of the private key will be required for constructing the signature). The reason for this is that from a realistic signature,

the public key can often be extracted, and therefore it is natural to model signatures such that the public key appears as a subterm of the signature. In particular, a principal should be able to distinguish the messages $\mathsf{sig}_{k_A}(N_A)$ and $\mathsf{sig}_{k_B}(N_A)$, they do not both look like "random noise."

Review questions after semester:

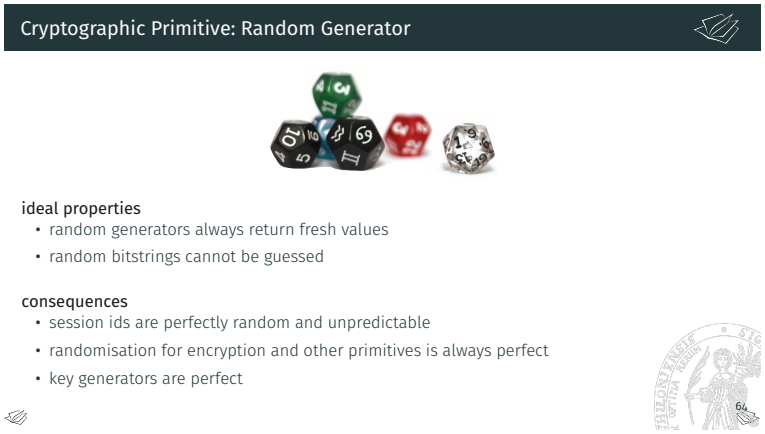
It also might seem surprising that we only have an equation for the "positive" case of signature verification, i.e., only the case where the tests return `ok` (the unused `error` result is only included to make the modeling more natural). Why do we not have en equation for the "negative" case of failed signature verification?



The issue is that requiring $\mathsf{hash}(x)$ to contain no information at all about the value x is not only unrealistic (as in the case of encryption, where at least some information about the length will leak), but actually incompatible with what we expect of a hash function. For example, given the value $\mathsf{hash}(x)$, it is easy to determine whether the message x is a fixed, known message m (e.g., the message $m =$"yes"), by computing $\mathsf{hash}(m)$ and then comparing $\mathsf{hash}(x)$ with $\mathsf{hash}(m)$. Assuming a collision-free hash function, this tells us whether $x = m$. Even if the hash function is not collision-free, we know that if $\mathsf{hash}(x) \neq \mathsf{hash}(m)$, then x does not equal m. Hence $\mathsf{hash}(x)$ clearly gives us information about x. However, as long as we only apply hash functions to random numbers (such as cryptographic keys), these issues do not arise. These questions will formally be considered when we study indistinguishability later in the lecture.

Review questions during semester:

Note that, in contrast to encryption and signatures, we do not state any equation here that a hash function is supposed to satisfy. Why is this not needed?



Encryption and signatures, among other primitives, require randomization on their own (beyond the requirement that, obviously, keys need to be randomly generated). The same is true for protocols (we will see a lot of protocols that use so-called nonces in the lecture). Since randomization is an essential to obtain security of a cryptographic primitive or protocol, a "weak" random generator (e.g., one whose output is predictable) can render an otherwise secure construction useless, in the same way as using a weak encryption function can.

Exercise Task: *WhatsApp Authentication* 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. Think about 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.

Exercise Task: *simple example protocol* 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 $\mathsf{sig}_{k_A}([M, B, N_B])$.

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 it?
3. What happens if the B is removed from Alice's last message?

3.3 An Example and an Attack

An Example: Authentication

goal
authentication: Bob wants to be sure that he is talking to Alice

infrastructure
PKI: Alice and Bob have public keys k_A and k_B

authentication protocol

$A \to B$	A	Hi, I'm Alice!
$B \to A$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_B)$	Prove this!
$A \to B$	$\mathsf{enc}^{\mathsf{a}}_{k_B}(N_B)$	I know Alice's secret key $\hat{k}_A$!

secure protocol?
- can Bob be sure he is talking to Alice when he receives $\mathsf{enc}^{\mathsf{a}}_{k_B}(N_B)$?
- obvious "bug" in protocol?

Review questions during semester: Do you think the protocol is secure? If not, can you come up with a concrete attack on the protocol? And even if there are security issues in the protocol, is there any guarantee that Alice or Bob have after the protocol has successfully completed?

The Needham-Schroeder Protocol

goal
authentication and key exchange

protocol

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

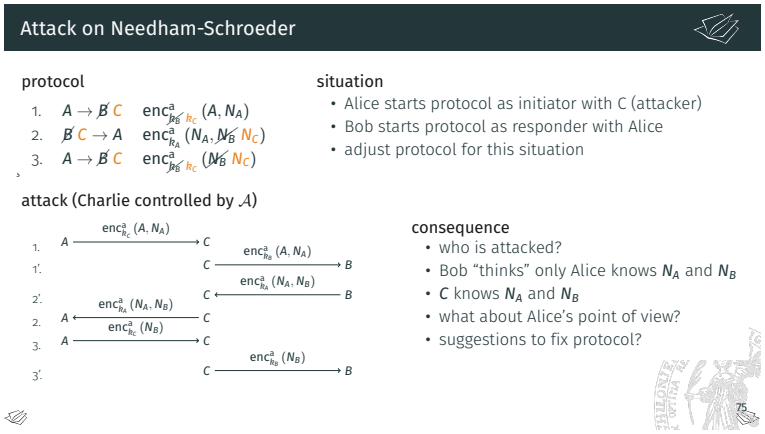
then $N_A \oplus N_B$ secure session key for A and B

recall
Needham: three-line programs!

really?
- protocol: 1978 [NS78]
- attack found: 1995 [Low96]

The nonces N_A and N_B are supposed to guarantee freshness: Both Alice and Bob receive their newly-created nonce back from the other, under cryptographic protection. This is supposed to imply that each of them can be assured that the other one was active during the protocol run and decrypted the nonce.

As we will see, the Needham-Schroeder protocol is insecure, a fixed version has been proposed by Lowe, and is known as the Needham-Schroeder-Lowe protocol. Since then, a "benchmark" of every formalism capable of reasoning about the security of authentication protocols is whether it can show the original Needham-Schroeder protocol as insecure, and the corrected Needham-Schroeder-Lowe protocol as secure [War05].



Review questions during semester: In the modeling here, we had to "rewrite" our protocol first in order to be applicable in the situation (Alice wants to establish a session with Charlie, not with Bob as in the protocol specification). This seems needlessly complicated. Can you think of a way to avoid this issue?

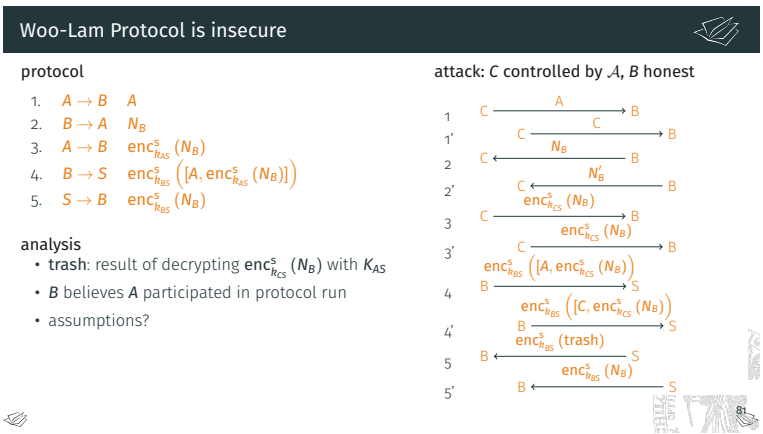
Exercise Task: *Fixing Broken Authentication Protocols* Consider the two authentication protocols presented in the exercise class:

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

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 as specific as possible in what "secure" means in this case.

3.4 More Examples



For the attack to work, we need to assume that Bob does not perform certain checks. For example, in steps 3 and 3′ of the attack, Bob gets the exact same message, even though (from Bob's point of view) one of the messages comes from a session with Alice, and the other one comes from a session with Charlie. However, it is realistic that Bob does not note this, since a normal protocol implementation will not try to match messages from different sessions with each other if these sessions are supposed to be unrelated. If such a check is required, a protocol must specify this explicitly.

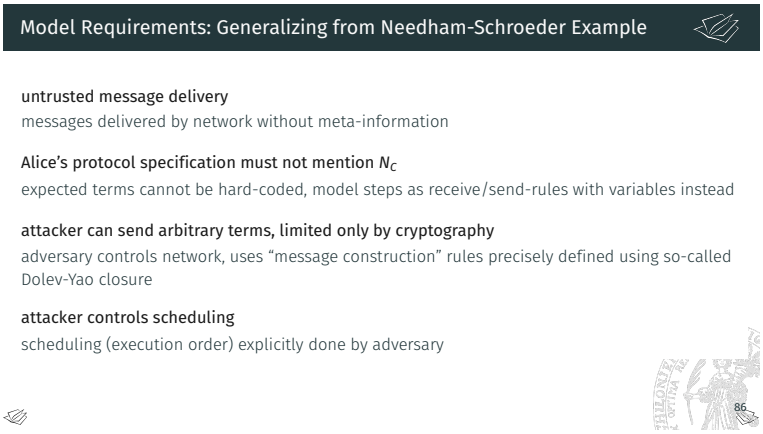
Additionally, the attack only can be successful if the step involving "trash" can be completed. A more suspicious Bob might abort the session with "Alice" when receiving message 5. However, the way the protocol is specified here does not even allow Bob to determine that the problematic message comes from the session with Alice, hence Bob instead treats message 5′ as coming from Alice's session, which allows the attack to complete successfully.

Review questions after semester:

Given the issue with the "trash"-message, can you still formally model the attack on the protocol using either our theoretical model used in the proof of the Rusinowitch-Turuani Theorem or in ProVerif?

3.5 Formal Protocol Model

3.5.1 Motivation and Requirements



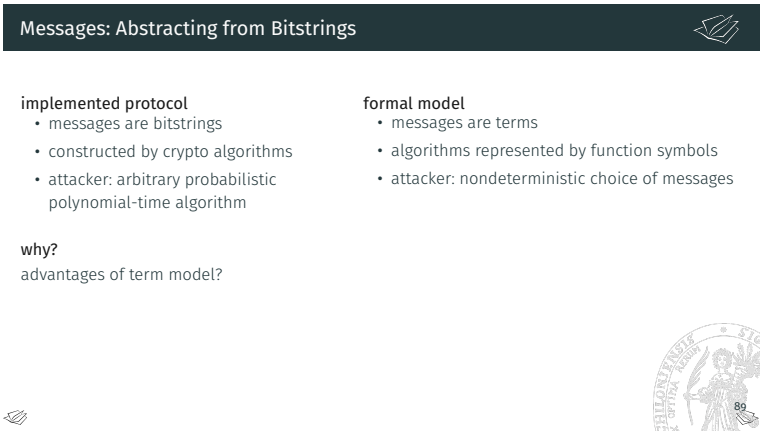
In any formal analysis, and maybe even more so in security, it is crucial to ensure that our formal models indeed capture everything that is relevant for the analyses we want to perform. For security, this includes correct modeling of the attacker capabilities.

Review questions during semester:

- What happens if our formal model gives too much or too little power to the attacker? As we will see later in the lecture, precisely modeling the attacker's capabilities leads to fundamental problems. Hence in some situations, we are forced to under- or over-approximate the attacker's capabilities. What are the consequences of each approach? With the goal of automatic security analysis in mind, which option is preferrable?

– The features mentioned for our model in this slide essentially correspond to attacker capabilities. Are these capabilities realistic?
– The list of model features (i.e., attacker capabilities) listed here were motivated by the fact that we need all of these capabilities to capture the attack on the Needham-Schroeder protocol. Clearly this approach risks missing some attacker capabilities that might not be needed here, but are relevant for other protocols. Did we miss anything, or are you reasonably convinced that we captured all realistic adversary capabilities?

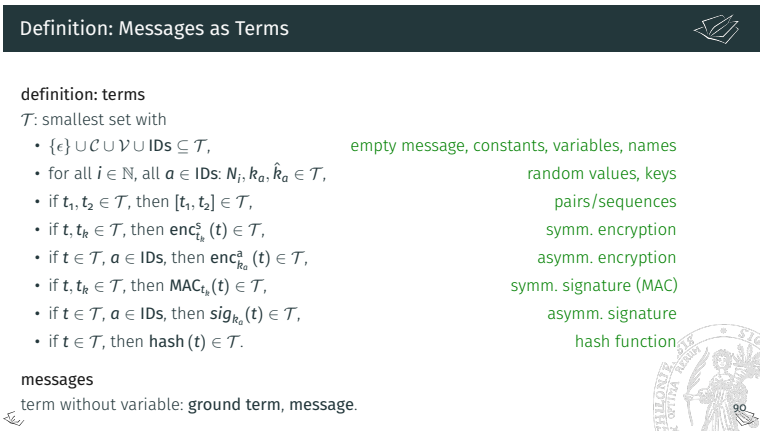
3.5.2 Messages: Formal Terms



Using a term model instead of concrete bitstrings is essential for performing a formal analysis. As a start, it is not clear how a "bitstring representation" of the messages involved in our protocols could even work without fixing the used algorithms for cryptographic primitives as well as the keys used by the participants beforehand (clearly, such a representation is also possible, and is used in "cryptographic models" as well as in real implementations). Moreover, using an abstract symbolic model such as terms gives our protocol messages a clear structure: We can read off, from a term, the sequence of cryptographic operation with which it is created. More importantly, using terms allows us to use formalisms that are more lightweight than algorithms processing bitstrings. We will work with term replacement rules and equational theories in the remainder of the lecture.

There are different ways to model atomic shared keys for symmetric cryptosystems. (Note that our model will also allow *composed* keys, which clearly need to contain some atomic value that satisfies some suitable secrecy condition.) In this lecture, we model these atomic keys (when they are pre-shared) as constants, which is also the reason why (different to some of the literature), we do not assume that the attacker knows all the constants appearing in the protocol. The difference between a constant and a nonce is that a constant may appear in more than one instances rules, while a nonce is generated by a single instance and is therefore local to that instance. Another reason to not simply assume that the adversary knows all constants is that for our attack definition (see later), we also need the value **FAIL** to be unknown to the attacker. However, note that our approach "conveniently" hides the fact that symmetric keys that appear in the protocol specifications of more than one participant are

required to be agreed upon or shared beforehand. This is an important aspect of the infrastructure required for a protocol run, and can (naturally) not be established by the protocol itself.[1]



In the above definition, the symbols $\mathcal{C}$, $\mathcal{V}$, and IDs denote finite sets of

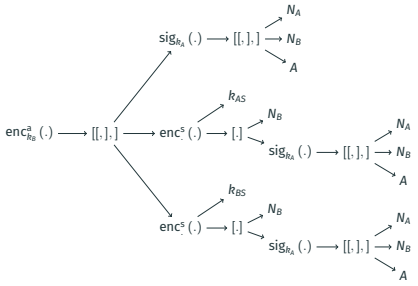
- *constants* e.g., fixed strings appearing in the protocol that are usually public, like `yes`, `no`, `ok`, `request`, `reply`, and similar values,
- *variables* are used to store parts of terms that participants receive during the protocol run,
- *identities* are strings that identify the participants in a protocol (e.g., email adresses or public keys).

Throughout the lecture, we will slightly relax the presentation of terms. For example, even though formally, our nonces are of the form N_i for some natural number i, we will also use notation as N_A or N_A^1, N_A^2 for nonces generated by Alice. Also, we will often leave out brackets used to indicate pairs and sequences, when the scope of the involved cryptographic operators is clear. For example, we write $\mathsf{enc}^{\mathsf{a}}_{k_A}\left(N_A, N_B, \mathtt{key}, k_{AB}\right)$ instead of $\mathsf{enc}^{\mathsf{a}}_{k_A}\left([N_A, [N_B, [\mathtt{key}, k_{AB}]]]\right)$.

Also note that our definition of terms contains the application of "cryptographic primitives" like encryption or signatues. An important detail missing from their representation here is what is often called the *security parameter*. This is a parameter of involved cryptographic primitives that e.g., determines the length of the keys and random strings used. It is obvious that this value has a high impact on the security of the resulting system, as clearly, using RSA with 64 bit keys offers no security at all, whereas we believe RSA with 4096 bit keys to be secure against realistic adversaries. Similarly, a session id (modeled as a nonce) consisting of 4 bits is easily guessed, but using long random numbers (generated by a cryptographically secure generator) as sessions ids offers an arbitrarily high level of security against guessing attacks.

[1] Of course there are key-exchange protocols that *do* establish shared keys. But these protocols require some other infrastrusture in place, as e.g., a public key infrastructure.

Terms: Tree Representation



remarks
- converting to term representation is straight-forward
- optimization possibilities?

91

Review questions during semester:

- Strictly speaking, this tree representation does not match our definition of terms exactly. Can you see a mismatch?
- The translation to terms is also not absolutely as straight-forward as claimed in the lecture. Do you see the problem?
- Which fixes to the presentation would solve these issues?
- One of these issues needs to be (and will, later in the lecture) addressed in our formal model working with terms as trees, we can pretty much ignore the other one. Which is which?

Definition: Subterms

definition: subterms
term $t \in \mathcal{T}$, then $\mathsf{Sub}(t)$ defined inductively:

- $\mathsf{Sub}(t) = \{t\}$ if t atomic, i.e., $t \in \{\epsilon\} \cup \mathcal{C} \cup \mathsf{IDs} \cup \left\{N_i, k_a, \hat{k}_a \mid i \in \mathbb{N}, a \in \mathsf{IDs}\right\}$
- $\mathsf{Sub}([t_1, t_2]) = \{[t_1, t_2], t_1, t_2\} \cup \mathsf{Sub}(t_1) \cup \mathsf{Sub}(t_2)$,
- $\mathsf{Sub}\left(\mathrm{enc}^s_{t_k}(t)\right) = \left\{\mathrm{enc}^s_{t_k}(t), t_k, t\right\} \cup \mathsf{Sub}(t) \cup \mathsf{Sub}(t_k)$,
- $\mathsf{Sub}\left(\mathrm{enc}^a_{k_a}(t)\right) = \left\{\mathrm{enc}^s_{k_a}(t), k_a, t\right\} \cup \mathsf{Sub}(t)$,
- $\mathsf{Sub}\left(\mathrm{sig}_{k_a}(t)\right) = \left\{\mathrm{sig}_{k_a}(t), k_a, t\right\} \cup \mathsf{Sub}(t)$,
- $\mathsf{Sub}\left(\mathrm{MAC}_{t_k}(t)\right) = \left\{\mathrm{MAC}_{t_k}(t), t_k, t\right\} \cup \mathsf{Sub}(t)$,
- $\mathsf{Sub}\left(\mathrm{hash}(t)\right) = \left\{\mathrm{hash}(t)\right\} \cup \mathsf{Sub}(t)$.

for $S \subseteq \mathcal{T}$: $\mathsf{Sub}(S) = \cup_{t \in S} \mathsf{Sub}(t)$.

92

Review questions during semester:

The definition of subterms shows how to split up a term into its components, which is of course relevant for e.g., parsing a message. The definition probably seems familiar from other areas of computer science (possibly the lecture “Fortgeschrittene Programmierung”).

- Is there anything surprising in this definition from a security point of view?
- How are subterms represented in the tree presentation of a term?

3.5.3 Message Construction: Dolev-Yao Closure

Definition: Dolev-Yao Closure

intuition
- DY closure contains everything we cannot stop the adversary from knowing
- and nothing else!
- represents *optimistic* view of cryptography

$S \subseteq \mathcal{T}$, then $\mathsf{DY}(S)$ is the smallest set $D \subseteq \mathcal{T}$ with

- $S \cup \{\epsilon\} \cup \mathsf{IDs} \subseteq D$,
- $t_1, t_2 \in D$ iff $[t_1, t_2] \in D$,
- if $t \in D$ and $a \in \mathsf{IDs}$, then $\mathsf{enc}^{\mathsf{a}}_{k_a}(t) \in D$,
- if $t, t_k \in D$, then $\mathsf{enc}^{\mathsf{s}}_{t_k}(t), \mathsf{MAC}_{t_k}(t) \in D$,
- if $t \in D$ and $\hat{k}_a \in D$, then $sig_{k_a}(t) \in D$,
- if $\mathsf{enc}^{\mathsf{s}}_{t_k}(t) \in D$ and $t_k \in D$, then $t \in D$,
- if $\mathsf{enc}^{\mathsf{a}}_{k_a}(t) \in D$ and $\hat{k}_a \in D$, then $t \in D$,
- if $\mathsf{sig}_{k_a}(t) \in D$, then $t \in D$,
- if $\mathsf{MAC}_{k_t}(t) \in D$, then $t \in D$,
- if $t \in D$, then $\mathsf{hash}(t) \in D$.

note

model allows composed keys for symmetric cryptosystems

96

Note that this model allows composed keys for symmetric encryption, since in $\mathsf{enc}^{\mathsf{s}}_{t_k}(t)$, the key t_k can be an arbitrary term.

Review questions during semester:

- Which purpose do composed symmetric keys serve? Can you give an example of a protocol where using composed keys makes the analysis more natural?
- The equation describing verifying signatures is not represented in the definition of the Dolev-Yao closure (decryption is represented by allowing the plaintext to be derived when both ciphertext and key are available). Why is this not required?

3.5.4 Algorithm: Computing the Dolev-Yao Closure

Video Lecture The following slides are presented in the video "Computing the Dolev-Yao Closure" (`https://cloud.rz.uni-kiel.de/index.php/s/LXBHnCergZ35sfF`).

Result: Decision Procedure

decision problem

Problem:	DERIVE
Input:	set of terms S, term m
Question:	is $m \in \mathsf{DY}(S)$?

theorem

DERIVE can be decided in polynomial time.

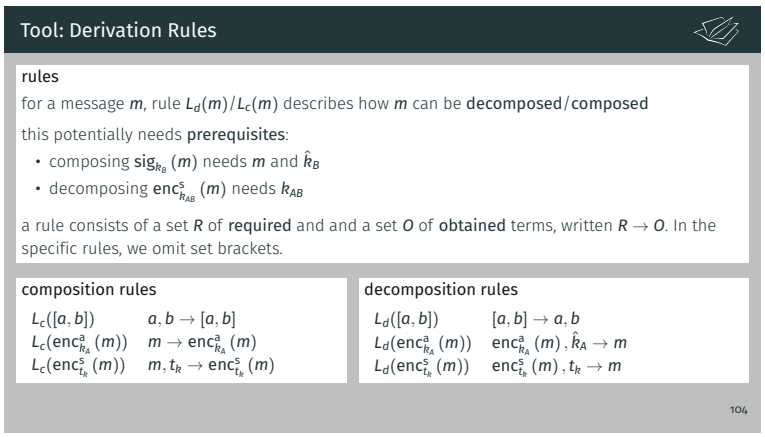
reference

Michaël Rusinowitch and Mathieu Turuani. "Protocol insecurity with a finite number of sessions, composed keys is NP-complete". In: Theoretical Computer Science 1-3.299 (2003), pp. 451–475

102

Our main interest in this algorithm is its role in the proof of the Rusinowitch-Turuani Theorem (see later in the lecture), our main result on automatic protocol analysis.

Review questions after semester:
Does the proof of the Rusinowitch-Turuani Theorem actually require the fact that the derivation problem can be solved in polynomial time, or would it suffice to show that it can be solved in *nondeterministic* polynomial time, i.e., in NP? And is it significantly easier to prove the latter fact?



The semantics of the rules are as follows:

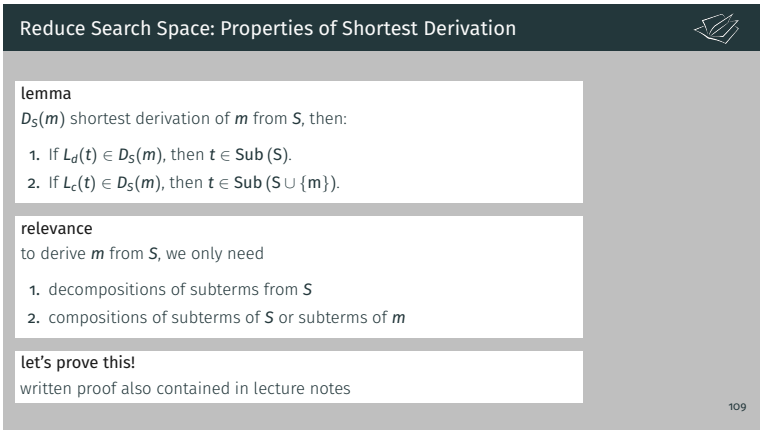
- A rule $L_d(m)$ written as $R \to O$, where R and O are sets of messages with $m \in R$ (the set brackets are omitted in the slide presentation) describes that the rule is a *d*econstruction rule for the message m, which allows any participant (and in particular the adversary) with access to m and the other terms in R to decompose the message m, and obtain the terms in O as a result.
- A rule $L_c(m)$ written as $R \to O$, where R and O are sets of messages with $m \in O$ (set brackets again omitted on the slide) describes that the rule is a *c*econstruction rule for the message m, which allows any participant (and in particular the adversary) with access to all terms in R to compose the term m.

Exercise Task: *DY closure and derivations* 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: $S = S_0 \to_{L_0} S_1 \to_{L_1} \ldots S_{n-1} \to_{L_{n-1}} S_n$ 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.



Lemma 1 *If $D_S(m)$ is a shortest derivation of m from S, then:*

1. *if $L_d(t) \in D_S(m)$, then $t \in \mathsf{Sub}(\mathsf{S})$.*
2. *if $L_c(t) \in D_S(m)$, then $t \in \mathsf{Sub}(\mathsf{S} \cup \{\mathsf{m}\})$.*

The lemma reduces the search space when determining whether a message m can be derived from a set S by showing that only derivation steps of two kinds are necessary: In a shortest derivation, a deconstruction step is only needed to access messages that are subterms of the set S which is the source of the derivation. Similarly, construction steps are only required for messages that appear as subterms of terms in the set S or of the target message m.

The idea behind the proof is simple: Deconstructing a term that does not appear (as a subterm) in S can never be required, because with our derivation rules, this only gives us terms that we had access to already previous to this hypothetical derivation step. Hence such a step does not occur in a *minimal* derivation. For the second part, constructing a term that neither appears in S nor in m can also not be required, since (due to the first part) we will never deconstruct such a term (since it is not in S), and we also do not need it for our target message m (since it is not a subterm of m). We now formally prove the lemma.

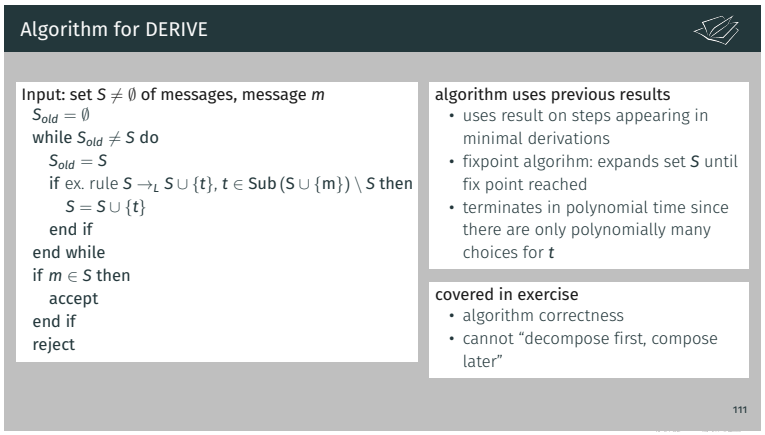
Proof Let $D_S(m)$ be the derivation $S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} \cdots \rightarrow_{L_{n-1}} S_n$, with $S_0 = S$ and $m \in S_n \setminus S_{n-1}$. Consider step i in the derivation.

1. Let $L_i = L_d(t)$ for some $i \in \{0,\ldots,n-1\}$. We indirectly assume that $t \notin \mathsf{Sub}(\mathsf{S})$. Then there is some rule $L_j = L_c(t)$ with $j \in \{0,\ldots,i-1\}$, in which t is constructed. We therefore have $\mathsf{Sub}^1(\mathsf{t}) \subseteq S_j$. Hence, the application of the rule L_i is superflous, and we have a contradiction to the minimality of $D_S(m)$.
2. Let $L_i = L_c(t)$ for some $i \in \{0,\ldots,n-1\}$. We assume that $t \notin \mathsf{Sub}(\mathsf{S} \cup \mathsf{m})$, and choose i maximal with this property. Since $t \neq m$ and $D_S(m)$ is minimal, the term t is used in some later application of a rule. Following part 1, there is no rule of the form $L_d(t)$ in $D_S(m)$. Therefore, there must be a later rule L_ℓ, $\ell > i$ that constructs a term t' with $t \in \mathsf{Sub}(\mathsf{t}')$. It follows that $t' \notin \mathsf{Sub}(\mathsf{S} \cup \mathsf{m})$, which is a contradiction to the maximality of i.

□

Exercise Task: *minimal derivation properties* 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?



The condition "if there exists a rule ..." of course is supposed to range over the set of derivation rules we defined for our cryptographic primitives. It is easy to see that different sets of primitives can be covered as well, if their Dolev-Yao closure is captured by derivation rules that fulfull the condition discussed in the exercise task. Also, the notation "rule $S \rightarrow_L S \cup \{t\}$ (as in our definition of a derivation as a sequence of derivation rule applications) in this algorithm is a shorthand for "a rule $R \rightarrow O$ with $R \subseteq S$ and $t \in O$.

Review questions during semester: The algorithm heavily relies on our characterization of shortest derivations. Does the algorithm always find a shortest derivation if it finds a derivation at all?

Exercise Task: *DY algorithm correctness* 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).

3.5.5 Message Parsing and Delivery: Receive/Send Actions, Substitutions, Matching

Definition: Receive/Send Actions

formalize protocol instruction
parse incoming message, send reply

receive/send actions
receive/send action: pair $(r, s) \in \mathcal{T} \times \mathcal{T}$, write $r \to s$.

example from Needham-Schroeder
Bob's rule: $\mathsf{enc}^{\mathsf{a}}_{k_B}(A, x) \to \mathsf{enc}^{\mathsf{a}}_{k_A}(x, N_B)$
- k_A, k_B, A: known, assume knowledge of $\hat{k}_B$
- N_B: new nonce (generated by Bob)
- x: references Alice's nonce, repeated in Bob's response

variable handling
- x: stores (supposedly) nonce from Alice
- nonce (value of x) potentially used again later in protocol, must be stored

Needham Schroeder (informal)

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

recall
Bob's (Alice's) protocol description must not contain N_A (N_B, N_C, ...).

116

Review questions during semester: The definition of receive/send actions is very general and does not contain any specific conditions related to security. Therefore, it is not surprising that, using this definition, we can define some rules that are problematic from a security point of view.

Do you see any issues with the following rules? Which of these issues would be found by a security analysis, and which ones are more fundamental?

- $\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(y) \to [k_{AB}, y]$
- $\mathsf{enc}^{\mathsf{s}}_{x}(y) \to [x, y]$
- $\mathsf{enc}^{\mathsf{a}}_{k_a}(x) \to \mathsf{sig}_{k_a}(x)$
- $\mathsf{hash}(x) \to x$
- $\mathsf{enc}^{\mathsf{a}}_{k_a}(N_A) \to \hat{k}_a$

Definition: Substitutions

definition: substitutions
- substitution: function $\sigma : \mathcal{V} \to \mathcal{T}$ with $\sigma(x) \neq x$ for a finite number of x
- σ ground substitution, if $\sigma(x)$ message for all x with $\sigma(x) \neq x$.

intuition
- finite local memory of participants
- $\sigma(x) = x$: "uninitialized" variable

extension to terms
for $t \in \mathcal{T}$, σ substitution, $\sigma(t)$ defined inductively:
- $\sigma(x)$ defined for $x \in \mathcal{V}$
- $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$

examples
⇝ lecture notes

117

As an example, consider the following:

- $\sigma(x) = \mathsf{enc}^{\mathsf{a}}_{k_a}(N_A, \mathsf{sig}_{k_B}(N_B, y))$
- $\sigma(y) = \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_C)$

– $t = \mathsf{MAC}_{\mathsf{k_{AB}}}(x)$

then: $\sigma(t) = \mathsf{MAC}_{\mathsf{k_{AB}}}\left(\mathsf{enc}^{\mathsf{a}}_{k_a}\left(N_A, \mathsf{sig}_{k_B}\left(N_B, \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(N_C)\right)\right)\right)$

Review questions during semester:

Clearly, every protocol participant (and, going further, every parallel session of each participant) will have its own "local memory," which is modeled as a substitution. However, we only use a single global substitution σ that models the "local memory" of all participants simultaneously. Why does this not lead to inconsistencies, i.e., multiple possible values for $\sigma(x)$ for some variable x?

Matching: Applying Receive/Send Actions

situation in protocol run
- memory: substitution σ
- next action: $r \to s$
- incoming message: m

parsing m with $r \to s$
- update substitution to σ'
- outgoing term: $\sigma'(s)$

definition: matching

a term r matches with message m and substitution σ via substitution σ', if
- $\sigma'(r) = m$, and — σ' consistent with incoming message
- $\sigma'(x) = \sigma(x)$ for all x with $\sigma(x) \neq x$. — σ' consistent with state

119

Obviously, there can be more than one substitution σ' that satisfies the above criteria. We are therefore usually interested in a *minimal* σ', i.e., one that assigns values only to the variables that appear in the already-processed receive/send rules. However, it is not necessary to formally restrict our substitutions in such a way, since it is the adversary's job to come up with a substitution that is consistent among the entire run of the protocol (see our later definition of when a protocol is insecure). Since the substitution is largely under the control of the adversary, we can assume the substitution to avoid inconsistencies arising from, e.g., assigning a value $\sigma'(x)$ early on for a variable that does not occur in the protocol steps so far.

Review questions during semester: Who performs the computation of the matching σ' (or the "final matching" after the last protocol step) in our protocol model? Which keys, nonces, and other values are required to compute this matching, and which participant (honest or adversary) has access to all these values?

Review questions after semester:

– Can you see whether the protocol model ensures that all terms sent over the network (i.e., the term $\sigma'(x)$ above) are in fact messages, i.e., do not contain variables?
– In our later definition of a (successful) attack, the requirement that all sent messages match with the corresponding receive/send rule is not explicitly stated. Why is this not necessary? Why do we discuss the definition of matching terms and substitutions at all?

Matching: Example

motivation
- matching: checks whether incoming term fits expectations
- expectations depend on
  - next rule in the protocol: receive/send rule from protocol
  - terms seen previously in protocol run: current substitution σ

example situation
- next receive/send rule:
  $(\mathrm{enc}^{\mathrm{a}}_{k_A}(x_A^1, N_A^1), \mathrm{sig}_{k_B}(x_A^2, y)) \to \mathrm{sig}_{k_A}(y, x_A^1, x_A^2, N_A^1, N_A^2)$
- substitution:
  - $\sigma(x_A^1) = N_B^1$
  - $\sigma(x_A^2) = N_B^2$
  - $\sigma(y) = y$

reactions to incoming terms

matches? resulting substitution/reply?
- $(\mathrm{enc}^{\mathrm{a}}_{k_A}(N_B^1), \mathrm{sig}_{k_B}(N_B^2, N_C))$
- $(\mathrm{enc}^{\mathrm{a}}_{k_A}(N_B^2, N_A^1), \mathrm{sig}_{k_B}(N_B^1, N_C))$
- $(\mathrm{enc}^{\mathrm{a}}_{k_A}(N_B^1, N_A^1), \mathrm{sig}_{k_B}(N_B^2, N_C))$
- $(\mathrm{enc}^{\mathrm{a}}_{k_A}(N_B^1, N_A^1), \mathrm{sig}_{k_B}(N_B^2, N_B^2))$

120

Review questions during semester: What are the reactions to the four examples for possible incoming terms?

3.5.6 Protocol Specifications: Instances and Protocols

Formal Protocol Definition

definition: protocol instance $\mathcal{I}$
sequence of actions
- $r_0 \to s_0$,
- $r_1 \to s_1$,
- …,
- $r_{n-1} \to s_{n-1}$

with $\mathcal{V}(s_i) \subseteq \cup_{j \le i} \mathcal{V}(r_i)$ for all i.
($\mathcal{V}(t)$: variables in term t)

example role
1. $\epsilon \to \mathrm{enc}^{\mathrm{a}}_{k_B}(A, N_A)$
2. $\mathrm{enc}^{\mathrm{a}}_{k_A}(N_A, x) \to \mathrm{enc}^{\mathrm{a}}_{k_B}(x)$

consequences for modeling

what kind of protocols can (can't) we express?

definition: protocol

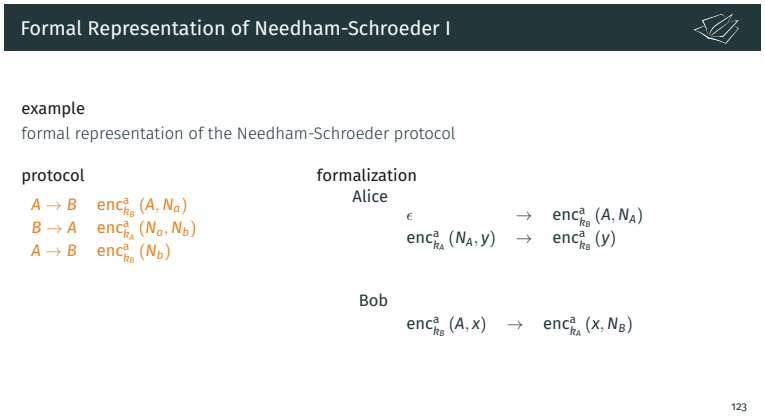
protocol consists of
- instances $\mathcal{I}_0, \ldots, \mathcal{I}_{n-1}$, and
- a finite set I of messages (the initial adversary knowledge).

122

In the above definition, the notation $\mathcal{V}(t)$ for a term t denotes the set of variables appearing in the term t. The condition ensures that variables appearing in the right-hand side of a receive/send action also appear in the left-hand side of the current or a previous receive/send action. Therefore, the variables appearing in the reply sent to the current incoming terms all have been assigned values before and the reply is in fact a message (i.e., a ground term).

Review questions during semester:

– This formalization of protocols omits some information that is present in the informal Alice-and-Bob notation we used previously to express protocols. What exactly is "missing" from this more formal representation, and why do we omit these details here?
– It is sometimes simpler to disregard the initial knowledge in a protocol. Can you see a way to rewrite a protocol such that its relevant properties will not change, but we can set the initial knowledge to the empty set?



Example 1 This is the formalization of the Needham-Schroeder protocol as receive/send actions:

Alice

$$\epsilon \to \text{enc}^{\text{a}}_{k_B}(A, N_A)$$
$$\text{enc}^{\text{a}}_{k_A}(N_A, y) \to \text{enc}^{\text{a}}_{k_B}(y)$$

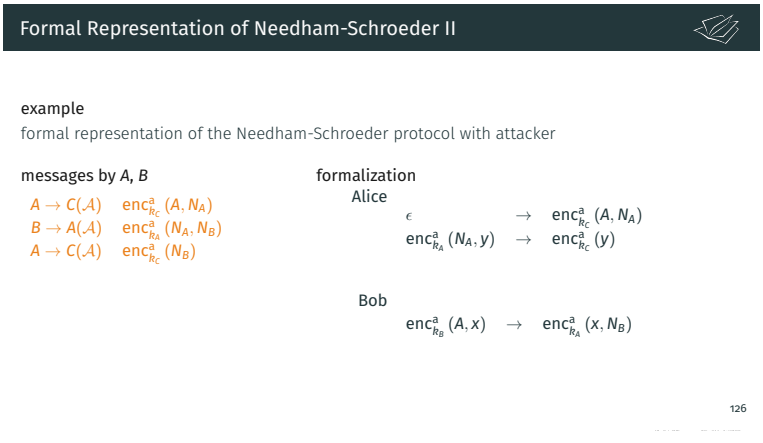
Bob

$$\text{enc}^{\text{a}}_{k_B}(A, x) \to \text{enc}^{\text{a}}_{k_A}(x, N_B)$$

Review questions during semester:

Compared to our informal notation on the left side of the slide (also called "Alice-and-Bob-notation"), the formal representation is longer. The main reason is that each message is represented here twice, once for the sending participant and once for the receiving one. Is this necessary, or redundandy that we can avoid?

3.5.7 Sessions and Scheduling: Execution Orders



Example 2 This is very similar to Example 1 above: The only difference is that now, Alice expects to be talking to Charlie instead of Bob, since this is the scenario in which the attack succeeds.

Alice	ϵ	$\rightarrow$	$\mathsf{enc}^{\mathsf{a}}_{k_C}(A, N_A)$
	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, y)$	$\rightarrow$	$\mathsf{enc}^{\mathsf{a}}_{k_C}(y)$
Bob	$\mathsf{enc}^{\mathsf{a}}_{k_B}(A, x)$	$\rightarrow$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(x, N_B)$

Protocol Model

saw
formal protocol model, example for protocol run

crucial question
when is a protocol secure?

need: attacker model
- completely controls network
- can control participants (obtain their private keys)
- also: can start protocol sessions!

intuition
- Needham-Schroeder: attack when Alice "starts protocol with $\mathcal{A}$ (C)"
- to find attack: adversary must be able to "start protocol"
- also: attacker controls interleaving

our model: sessions (for now) part of the protocol
consequence?

127

Review questions during semester: As indicated, there is an obvious contradiction on the slide. What is the problem here, and why do you think we define our model in this "problematic" way? What would a "fix" for the model look like, and what kind of problems result from such a fix?

3.5.8 Protocol Security: (Successful) Attacks

Executing Instances II

situation
- attacker controls scheduling, network

assumption: $\mathcal{A}$ controls everyhing we don't.

next step: message delivery
step $A \rightarrow C$ *(Charlie controlled by $\mathcal{A}$)*
- belongs to instance "A with C ($\mathcal{A}$)"
- message created by protocol instance

step $A \rightarrow B$ *(Alice impersonated by $\mathcal{A}$)*
- belongs to instance "A ($\mathcal{A}$) with B"
- messages created by adversary

questions
- which messages can $\mathcal{A}$ send?
- restricted only by cryptography!
- what does this mean concretely?
- $\rightsquigarrow$ recall Dolev-Yao closure!

131

Review questions during semester:

– In both steps, the adversary in some sense "plays" one of the parties (Charlie in the step $A \rightarrow C$, and Alice in the step $A \rightarrow B$). What is the difference between a party being

controlled by the adversary (as with Charlie in the first step) and being impersonated by the adversary (as Alice in the second step)?
– The two missing cases from this slide are "honest-to-honest" and "adversary-controlled-to-adversary-controlled" communication. How are these cases treated?

Protocol Runs: Formal Definition

definition: attack

$P = \{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$ protocol, initial adversary knowledge I, i-th rule of $\mathcal{I}_j$ named $r_i^j \to s_i^j$. An attack on P consists of

- an execution order o for P,
- a substitution σ on the variables in P such that

$$\sigma(r^{o(k)}_{\#o(k)}) \in \mathsf{DY}\left(I \cup \left\{\sigma(s^{o(\ell)}_{\#o(\ell)}) \mid 0 \le \ell < k\right\}\right).$$

terms

- $\sigma(r^{o(k)}_{\#o(k)})$: received in step k
- $\sigma(s^{o(k)}_{\#o(k)})$: sent in step k

simplification

- identify "protocol run" and "attack"
- protocol cannot be executed without $\mathcal{A}$ (network)

132

The final requirement simply means that every message that a participant receives in some step can actually be constructed by the adversary from the messages that she "knows," i.e., her initial intruder knowledge plus all messages the honest participants sent previously—since we assume that the adversary controls the network, this implies that she learns all messages sent by the other principals. In particular, this implies that it is always possible for the adversary to act as an "honest" network and simply deliver the messages sent by the honest principals to their intended receipients.

Successful Attack: Definition

variable "attack definition"

- success criterion for attack depends on protocol (goal)
- automatic analysis: security definition should be part of algorithm input
- integrate security definition into protocol: let designer specify security
- add "challenge interface" for adversary: define instances so that $\mathcal{A}$ can get constant **FAIL** if successful (**BREAK**-rules)

definition: successful attack

$P = \{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$ with initial adversary knowledge I. Attack (o, σ) is successful, if:

$$\mathsf{FAIL} \in \mathsf{DY}\left(I \cup \left\{\sigma(s^{o(\ell)}_{\#o(\ell)}) \mid 0 \le \ell < |o|\right\}\right).$$

literature: **secret** instead of **FAIL**

134

Review questions during semester: In this lecture, we always assume an active attacker who controls the network completely. As a consequence, even a successful run of a protocol is not possible without any attacker action, since without an attacker, network messages are not delivered at all. Is it still possible to model "passive attackers" in our setting? A common example is the so-called "honest-but-curious" attacker, who follows the protocol specification but tries to obtain more information than she should have. Can you think of a way to express such an attack in our model?

Application

example
application of definition to Needham-Schroeder protocol

steps
1. formalise protocol as r/s actions
2. formalise security specification: add BREAK-rules
3. find execution order for attack
4. find substitution for attack
5. check that attack is successful

135

Note that the BREAK-rules are not "natural steps of the protocol:" They provide the adversary with an interface to prove that she successfully broke the protocol. The usage of the constant BREAK is not technically needed, it only serves to ensure that these additional rules do not, by accident, look like other rules of the protocol. In an implementation of the protocol, these rules of course need to be removed: First, the rules serve no real-world purpose, and second, it is unclear which message in a real implementation would correspont to the constant FAIL.

Example 3 We continue Example 2.

1. The above formalization as receive/send actions already contains the required sessions (Alice with Charlie and Alice with Bob).
2. We add BREAK-rules to specify what it means for the protocol to be (in)secure. In this example, we choose secrecy of Bob's nonce N_B (see below for discussion of alternatives). Note that the attack is against Bob, not Alice: In the situation we model, Alice intends to communicate with Charlie, and hence it is no security issue that the adversary (with Charlie's key $\hat{k}_C$) can access the nonces. However, from Bob's point of view, the nonces N_A and N_B should only be known by Alice. Therefore, we add the rule to Bob's protocol instance.
   Alice's sequence of rules is therefore unchanged:
   $$\begin{array}{lll} r_0^A \to s_0^A & \epsilon & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_C}(A, N_A) \\ r_1^A \to s_1^A & \mathsf{enc}^{\mathsf{a}}_{k_a}(N_A, y) & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_C}(y) \end{array}$$
   Bob's set of rules contains the additional BREAK-rule:
   $$\begin{array}{lll} r_0^B \to s_0^B & \mathsf{enc}^{\mathsf{a}}_{k_A}(A, x) & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_A}(x, N_B) \\ r_1^B \to s_1^B & [\mathsf{BREAK}, N_B] & \to \quad \mathsf{FAIL} \end{array}$$
   Note we again relax our notation a bit, and use A and B to indicate the instances of Alice and Bob, instead of 0 and 1 as strictly required by our definition.
3. The execution order simply fixes the order in which the honest participants are active in the protocol run representing the successful attack. Since we also model the final step in which the adversary learns the FAIL-constant, the execution order is $ABAB$.
4. The substitution simply assigns values to the variables so that the messages are exactly the ones from the attack on the protocol:

- $\sigma(x) = N_A$
- $\sigma(y) = N_B$

5. To verify that the attack is successful, we need to verify that the adversary can construct each message she sends from her initial knowledge and the messages received in earlier protocol steps. The initial knowledge for the adversary can in this case be assumed to only contain the two values BREAK and $\hat{k}_C$.

step	message sent by $\mathscr{A}$	recipient of $\mathscr{A}$ message	message received as reply
0	ϵ	A	$\mathsf{enc}^{\mathsf{a}}_{k_C}(A, N_A)$
1	$\mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$	B	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, N_B)$
2	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, N_B)$	A	$\mathsf{enc}^{\mathsf{a}}_{k_C}(N_B)$
3	$[\mathsf{BREAK}, N_B]$	B	FAIL

It is easy to see that each of the messages sent by the adversary is in the Dolev Yao closure of the then-available terms, and that $\mathscr{A}$ can derive FAIL after the protocol run. Therefore, the attack is successful.

This choice of modeling security of the protocol may seem arbitrary here, and there are in fact several ways to formalize the security criterion. As discussed in the introduction, finding the "correct" security criterion is highly nontrivial, and wrong specifications can have serious consequences. For the Needham-Schroeder protocol, we will revisit the issue when we model the protocol and its security in ProVerif later in the lecture. In our concrere modeling, the security property we chose is "the value N_B stays secret from the adversary." Another option is to only require that the combination of N_A and B_B remains secret—this is a weaker security guarantee that can easily be modeled similarly to the above.

The strictest (and possibly most natural) property is to require that *both* N_A and N_B remain secret. Can you model this using our framework? Carefully think about what the attack scenario is in this case, and who is attacked.

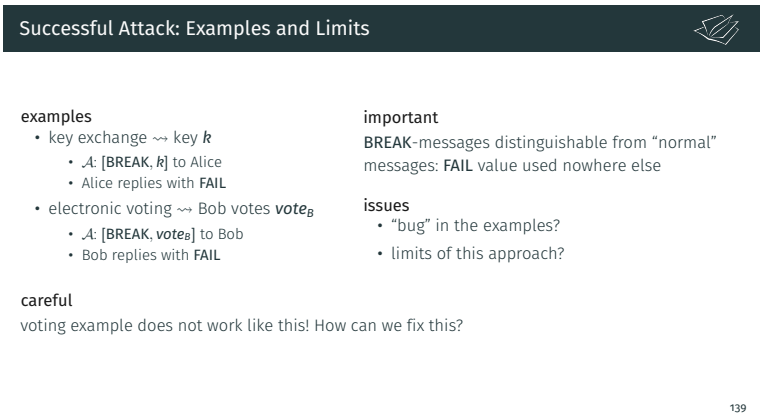
Exercise Task: *Formal Representation of the Woo Lam Protocol* Study the authentication protocol by Woo and Lam (see slide 17 of the lecture from November 10).

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?

Exercise Task: *Otway Rees Protocol* Consider the following protocol (Otway-Rees-Protocol):

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

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.)



The issue with the above is in the example for electronic voting. If we model the term vote_B as simply the identifier of the party (or candidate) that Bob intends to vote for, and assume that the list of parties or candidates is public knowledge (i.e., contained in the adversary's initial knowledge) then clearly, the message $[\mathsf{BREAK}, \mathsf{vote}_B]$ can always be derived by the adversary. Therefore, the adversary can always send the corresponding BREAK-message to Bob, hence an attack is always possible and any protocol modeled like this is insecure. This problem comes from the earlier-discussed issue that the Dolev Yao closure only models which terms the adversary can derive, and does not allow to study the question whether the adversary *knows* the content of a particular message. More generally, the approach to define security at this point in the lecture only allows to model security properties that can be defined as reachability questions in an appropriately defined state graph. More general properties (like trace properties that go beyond reachability, or episetemic properties of which the above is an example) cannot be modeled with this definition. However, the definition is attractive since it is conceptually simple, and it suffices to model most secrecy and authentication properties, among others. We will revisit election protocols and how to handle their security properties later in the lecture.

Exercise Task: *Security Modeling Issues: Are we Missing Something?* 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?

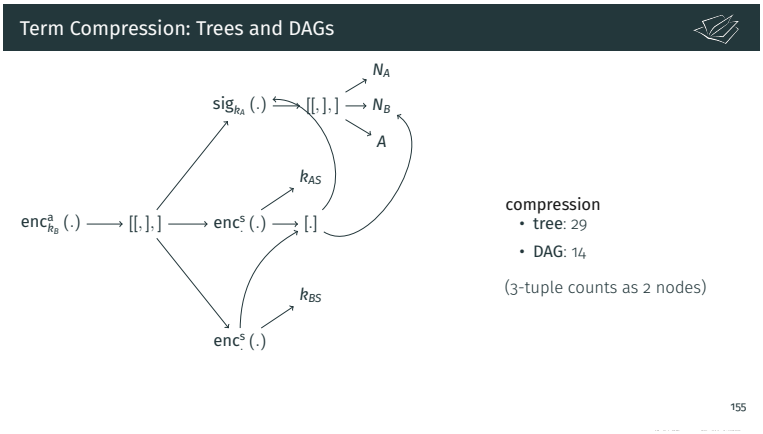
Exercise Task: *Formal Protocol Model: Features and Omissions* There are a couple of usually assumed properties of cryptographic systems that are not explicitly expressed in our protocol model. Which of the following properties are implicitly expressed in our model, and which are not? Are any of the "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 public-key infrastructure 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.

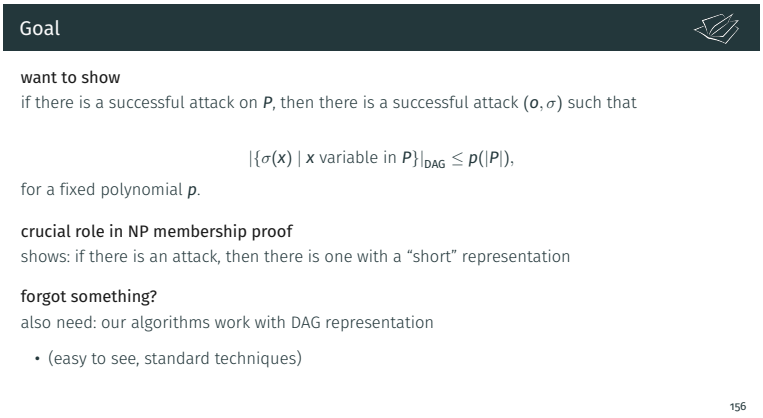
3.6 Automatic Analysis: Theoretical Foundations

3.6.1 Decidability: The Rusinowitch-Turuani Theorem

3.6.2 DAGs



Note that, as alluded to in the review questions for the initial introduction of the tree representations of a term (see page 18), we again omit the order of edges in this slide: In the formal definition of the graph representation, the edges are effectively labelled as "left" or "right" edges. In the example of the slide here, we would also need to define an order on the outgoing edges of each node in the graph.

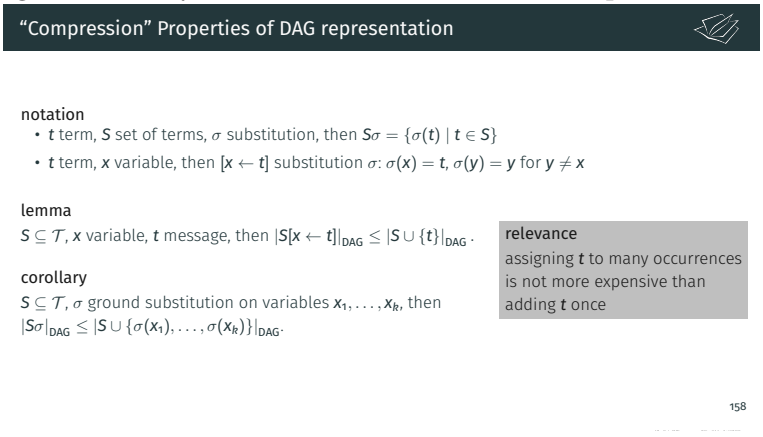


Review questions during semester: The proof of the Rusinowitch-Turuani theorem focuses on the construction of a short representation of the substitution contained in a successful attack. However, an attack also contains an execution order, which clearly also needs to be represented in a "short" (i.e., polynomial-length in the protocol description) encoding. Why do we not discuss this issue in the lecture?

Exercise Task: *exponential attack size* 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
     $t_1 = [x_1, [x_2, [x_3, [x_4, [\ldots, [x_{i-1}, [x_i, 0]] \ldots]]]]]$
     $t_2 = [[[[[\ldots[[0, x_i], x_{i-1}], \ldots], x_4], x_3], x_2], x_1]$.
  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?



We prove the lemma from the slide (the corollary follows trivially via induction on k). Note

that the statement of the lemma should seem very natural: The purpose of the DAG representation is to avoid having to add many copies of the same term t, and instead only add references to the term. Hence replacing many occurrences of a variable x with the term t is not more expensive than adding the term t once.

Proof We define an injective function $f\colon \mathsf{Sub}(\mathsf{S}[\mathsf{x} \leftarrow \mathsf{t}]) \to \mathsf{Sub}(\mathsf{S} \cup \{\mathsf{t}\})$ as follows: For a subterm α of $S[x \leftarrow t]$, we define

- $f(\alpha) = \alpha$, if $\alpha \in \mathsf{Sub}(\mathsf{t})$,
- $f(\alpha) = \alpha'$, if $\alpha = \alpha'[x \leftarrow t]$ for some subterm $\alpha' \in \mathsf{Sub}(\mathsf{S}) \setminus \mathsf{Sub}(\mathsf{t})$.

We show that f is injective: Let $\alpha, \beta \in \mathsf{Sub}(\mathsf{S}[\mathsf{x} \leftarrow \mathsf{t}])$ with $\alpha \neq \beta$.

- If $\alpha, \beta \in \mathsf{Sub}(\mathsf{t})$, then $f(\alpha) = \alpha \neq \beta = f(\beta)$.
- If $\alpha \in \mathsf{Sub}(\mathsf{t}), \beta \in \mathsf{Sub}(\mathsf{S}) \setminus \mathsf{Sub}(\mathsf{t})$, then $f(\alpha)$ is a subterm of t, and $f(\beta)$ is a subterm of a term in $S \setminus t$. Hence $f(\alpha) \neq f(\beta)$.
- If $\alpha, \beta \in \mathsf{Sub}(\mathsf{S}) \setminus \mathsf{Sub}(\mathsf{t})$, then $\alpha \neq \beta$ implies that $\alpha' \neq \beta'$, and hence $f(\alpha) \neq f(\beta)$.

□

3.6.3 Short Attacks

Size of an Attack

want to show

if P insecure, then there is successful attack with "short" representation (choose the "shortest")

definition

(σ, o) attack on protocol P. Then the size of σ (denoted with $|\sigma|$) is

$$\sum_{x \text{ variable in } P} |\sigma(x)|_{\mathsf{DAG}}.$$

definition

a successful attack (σ, o) on P is minimal, if $|\sigma| \leq |\sigma'|$ for every successful attack (σ', o') on P.

remark

- every insecure protocol has a minimal successful attack
- protocols can have several minimal successful attacks (see exercise)

161

Review questions during semester: In the above definition, we look at the size of each DAG representing the value of a variable individually. This only takes advantage of repetitions of subterms inside $\sigma(x)$ for each x separately. Is it also possible to use repetitions of, say, subterms of $\sigma(x)$ inside $\sigma(y)$ with this formalism? Why is this not considered in our proof?

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

Notation for Proof

simplification

attack consists of

- substitution σ
- execution order o

rule application

- $r^{o(0)}_{\#o(0)} \to s^{o(0)}_{\#o(0)}$
- $r^{o(1)}_{\#o(1)} \to s^{o(1)}_{\#o(1)}$

  $\vdots$
- $r^{o(n)}_{\#o(n)} \to s^{o(n)}_{\#o(n)}$

simplification

- write $r_i \to s_i$ instead of $r^{o(i)}_{\#o(i)} \to s^{o(i)}_{\#o(i)}$.
- note: this fixes execution order from attack
- used in proof of parsing Lemma, Rusinowitch-Turuani Theorem

163

Note that this notation can even be made formally correct, by rewriting the protocol such that there is only one protocol instance that performs all receive/send actions (in this case, the execution order is hard-coded into that "rewritten" protocol instance). However, this rewritten protocol would possibly no longer satisfy the requirement we discussed earlier for well-formed protocols, since to execute this protocol, knowledge of more than one private key may be required.

Example

protocol fragment (cp. Woo Lam Protocol)

$$\begin{array}{lcll} A & \to & B & \mathsf{enc}^{\mathsf{a}}_{k_B}\left(A, \mathsf{enc}^{\mathsf{a}}_{k_{AS}}\left(A, \mathsf{MAC}_{k_{AS}}\left(N_A, A, S\right)\right)\right) \\ B & \to & S & \mathsf{enc}^{\mathsf{a}}_{k_S}\left(B, \mathsf{verify}, \mathsf{enc}^{\mathsf{a}}_{k_{AS}}\left(A, \mathsf{MAC}_{k_{AS}}\left(N_A, A, S\right)\right)\right) \\ S & \to & A & \mathsf{enc}^{\mathsf{a}}_{k_B}\left(\mathsf{MAC}_{k_{BS}}\left(N_A, A\right)\right) \end{array}$$

attack

$\sigma(x) = \mathsf{enc}^{\mathsf{a}}_{k_{AS}}\left(A, \mathsf{MAC}_{k_{AS}}\left(N_A, A, S\right)\right)$

$\dots$

represent as receive/send actions

$$\begin{array}{lll} A & \epsilon & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_B}\left(A, \mathsf{enc}^{\mathsf{a}}_{k_{AS}}\left(A, \mathsf{MAC}_{k_{AS}}\left(N_A, A, S\right)\right)\right) \\ B & \mathsf{enc}^{\mathsf{a}}_{k_B}\left(A, x\right) & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_S}\left(B, \mathsf{verify}, x\right) \\ S & \mathsf{enc}^{\mathsf{a}}_{k_S}\left(B, \mathsf{verify}, \mathsf{enc}^{\mathsf{a}}_{k_{AS}}\left(A, \mathsf{MAC}_{k_{AS}}\left(y, A, S\right)\right)\right) & \to \quad \mathsf{enc}^{\mathsf{a}}_{k_B}\left(\mathsf{MAC}_{k_{BS}}\left(y, a\right)\right) \end{array}$$

term complexity

- Bob cannot build $\mathsf{enc}^{\mathsf{s}}_{k_{AS}}(\dots)$, needs variable
- $\sigma(x)$ is a "complex" term
- can be made arbitrarily complex → cannot prove "$|\sigma(x)| < c$" for any constant c"
- **reason** why term must be "complex?" structure of $\sigma(x)$ appears in r/s rules
- $\sigma(x)$ will be **parsed** by matching with a protocol rule

question

other reasons why $\sigma(y)$ must be "complex?"

165

The example is admittedly a bit contrived, but the motivation is as follows: We assume that B and S have a known public key (e.g., established with a certificate chain), but A does not (this is realistic, as most end-users do not have a verifiable pubblic key). The protocol structure then borrows from the Woo-Lam protocol, but knowledge of that protocol is not required to follow the example.

Parsing Lemma: Terms Always Match with Protocol

parsing lemma

P protocol, (o, σ) minimal successful attack on P, x variable in P with $|\sigma(x)| > 1$. Then there is a term t such that

- $t \in \mathsf{Sub}(r_0, \ldots, r_n, s_0, \ldots, s_n)$,
- t is not a variable,
- $\sigma(t) = \sigma(x)$.

statement

terms in variables represent steps in the protocol

informal justification

assume x with $|\sigma(x))| > 1$ counter-example

- for all $t \in \mathsf{Sub}(P)$ with $\sigma(t) = \sigma(x)$: t is variable.
- outmost operator in $\sigma(x)$ does not match with protocol.
  - then $\sigma(x)$ computed by $\mathcal{A}$
  - then $\sigma(x)$ does not get "parsed"
  - $\sigma(x)$ more complex than needed
  - contradiction, since (σ, o) minimal attack

166

The "informal justification" on this slide seems deceptively simple, we will see the entire proof in the sequel. Later in the lecture, we prove that automatic analysis for the "unbounded sessions case" is undecidable. Undecidability can also be shown for bounded sessions protocols with more complex processing instead of our receive/send rules. Therefore, a comparatively simple argument as the parsing lemma cannot be used to prove decidability of an appropriately generalized version of INSECURE in more general settings.

Review questions after semester: Which aspects of the parsing lemma (and also the following proof of the Rusinowitch-Turuani theorem, of which the parsing lemma is a major part) rely on certain ways in which our protocol model is "simple?"

Video Lecture The following slides are presented in the video "Parsing Lemma Proof" (`https://cloud.rz.uni-kiel.de/index.php/s/WoWy5TScbjFai6b`).

Parsing Lemma Proof Overview

parsing lemma statement

P protocol, (o, σ) minimal successful attack on P, x variable in P with $|\sigma(x)| > 1$. Then there is a term t such that

- $t \in \mathsf{Sub}(r_0, \ldots, r_n, s_0, \ldots, s_n)$,
- t is not a variable,
- $\sigma(t) = \sigma(x)$.

proof structure

1. assume (o, σ) counter-example: minimal attack, $\sigma(x)$ matches no term in protocol, $|\sigma(x)| > 1$
2. collect facts about appearence of $\sigma(x)$ in protocol run
3. show that $\sigma(x)$ can be derived by adversary
4. replace $\sigma(x)$ with ϵ in attack, this is a smaller successful attack on P

168

Lemma 2 *P protocol, (o, σ) minimal successful attack on P, x variable in P with $|\sigma(x)| > 1$. Then there is a term t such that*

- *$t \in \mathsf{Sub}(r_0, \ldots, r_n, s_0, \ldots, s_n)$,*
- *t is not a variable,*
- *$\sigma(t) = \sigma(x)$.*

Proof We can assume that the initial intruder knowledge is the empty set by rewriting the protocol such that there is an instance sending all terms from the initial knowledge as reply to ϵ as the first step. We also assume the adversary to perform this step first in any attack, hence $r_0 = \epsilon$. For the proof, assume indirectly that x is a counter-example.

Since (σ, o) is a minimal attack and $|\sigma(x)| > 1$, x appears in some receive-step. Let N_x be the first step for which $\sigma(x)$ appears as a subterm of a received term (this can be before the variable x appears for the first time):

$$N_x = min\left\{j \mid \sigma(x) \in \mathsf{Sub}\left(\sigma(\mathsf{r}_0), \ldots, \sigma(\mathsf{r}_j)\right)\right\}.$$

We list a few facts that follow immediately from the choice of N_x and our assumptions that we will use in the remainder of the proof.

Fact 1 $\sigma(x) \notin \mathsf{Sub}\left(\left\{\sigma(\mathsf{s}_0), \ldots, \sigma(\mathsf{s}_{\mathsf{N}_x-1}), \sigma(\mathsf{r}_0), \ldots, \sigma(\mathsf{r}_{\mathsf{N}_x-1})\right\}\right)$ The first appearence of $\sigma(x)$ in the protocol run cannot be in a send-term $\sigma(s_i)$: Otherwise $\sigma(x)$ would match with a subterm of s_i, a contradiction to the choice of x. Therefore, the first appearance of $\sigma(x)$ is as a subterm of some $\sigma(r_j)$. By choice of N_x, this happens in the step $r_{N_x} \to s_{N_x}$.

Fact 2 $\sigma(x) \in \mathsf{Sub}\left(\sigma(\mathsf{r}_{\mathsf{N}_x})\right)$. More: there is some $y \in \mathsf{Sub}\left(\mathsf{r}_{\mathsf{N}_x}\right)$ with $\sigma(x) \in \mathsf{Sub}\left(\sigma(\mathsf{y})\right)$ The first claim follows by choice of N_x. Existence of the variable y follows since by assumption, there is no non-variable subterm of the protocol (and hence in particular of r_{N_x}) that matches with $\sigma(x)$.

Fact 3 $\sigma(r_{N_x}) \in \mathsf{DY}\left(\underbrace{\left\{\sigma(s_0), \ldots, \sigma(s_{N_x-1})\right\}}_{=:S}\right)$. Since (o, σ) is an attack, we know that $\sigma(r_{N_x}) \in \mathsf{DY}\left(I \cup \left\{\sigma(s_0), \ldots, \sigma(s_{N_x-1})\right\}\right)$. Since we assumed $I = \emptyset$, the claim follows.

We now show that $\sigma(x)$ is a term that the adversary can constructed at time of the step N_x. This is important because this implies that any "complexity" in the term was introduced by the adversary.

Claim: $\sigma(x) \in \mathsf{DY}(S)$.

Proof By Fact 3, $\sigma(r_{N_x}) \in \mathsf{DY}(S)$. So, there is a minimal derivation

$$S = S_0 \to_{L_0} S_1 \to \cdots \to_{L_{n-1}} S_n$$

with $\sigma(r_{N_x}) \in S_n$. We show that there is some i with $\sigma(x) \in S_i$, the claim then follows since $S_i \subseteq \mathsf{DY}(S)$. Hence assume $\sigma(x) \notin S_i$ for all i. We show that $\sigma(x) \in \mathsf{Sub}(\mathsf{S}_\mathsf{i})$ for all i. For this we use induction, starting with $i = n$ and progressing to $i = 0$.

The start case $i = n$ follows since $\sigma(x) \in \mathsf{Sub}\left(\sigma(\mathsf{r}_{\mathsf{N}_x})\right)$ (by Fact 2) and $\sigma(r_{N_x}) \in S_n$ (by choice of the derivation).

For the induction step, assume that $\sigma(x) \in \mathsf{Sub}(\mathsf{S}_\mathsf{i})$, we need to show $\sigma(x) \in \mathsf{Sub}(\mathsf{S}_{\mathsf{i}-1})$. We distinguish two cases, depending on the type of derivation rule L_{i-1} that leads from S_{i-1} to S_i.

- If L_{i-1} is a decomposition rule, then the newly derived terms, i.e., the set $S_i \setminus S_{i-1}$, only consists of subterms of S_{i-1}. In particular, $\sigma(x) \in \mathsf{Sub}(\mathsf{S_{i-1}}) = \mathsf{Sub}(\mathsf{S_{i-1}})$ as required.
- If L_{i-1} is a composition rule, then $L_{i-1} = L_c(t)$ for some term t with $\mathsf{Sub}(\mathsf{t}) \setminus \{t\} \subseteq S_{i-1}$, and we have $S_i = S_{i-1} \cup \{t\}$. In particular, $\mathsf{Sub}(\mathsf{S_{i-1}}) = \mathsf{Sub}(\mathsf{S_i}) \setminus \{t\}$. Since we assumed $\sigma(x) \notin S_i$, we know that $\sigma(x) \neq t$. Hence it follows that

$$\sigma(x) \in \mathsf{Sub}(\mathsf{S_i}) \setminus \{t\} = \mathsf{Sub}(\mathsf{S_{i-1}}),$$

  as required.

Therefore, $\sigma(x) \in \mathsf{Sub}(\mathsf{S})$. This is a a contradiction to Fact 1 and therefore proves our claim. □

Due to Fact 1, the final step of the derivation of $\sigma(x)$ from S must be a composition rule[2].

Definition: We choose the substitution σ' defined as

$$\sigma'(y) = \sigma(y)[\sigma(x) \leftarrow \epsilon],$$

i.e., σ' is obtained from σ by replacing every occurrence of $\sigma(x)$ with ϵ. We now prove that (σ', o) is a successful attack on P. To prove this, it suffices to show that[2] $\sigma'(r_j) \in \mathsf{DY}\left(\{\sigma'(s_0), \ldots, \sigma'(s_{j-1})\}\right)$ for all j.

First note that in the case $j < N_x$, there is nothing to prove since the terms are unchanged and we can use the same derivation. Hence assume $j \geq N_x$. Since (σ, o) is an attack, we know that $\sigma(r_j) \in \mathsf{DY}\left(\underbrace{\sigma(s_0), \ldots, \sigma(s_{j-1})}_{=:T_0}\right)$. Hence there is a minimal derivation $D_{T_0}(\sigma(r_j))$ as follows:

$$T_0 \rightarrow_{L_0} T_1 \rightarrow_{L_1} \cdots \rightarrow_{L_n-1} T_n$$

We can choose the derivation above so that no step $L_d(\sigma(x))$ occurs[2] For a set of terms T (or a term t), T' (or t'), is the set T (the term t) where every occurrence of $\sigma(x)$ has been replaced with ϵ. We prove that there is a derivation

$$T'_0 \rightarrow_{L'_0} T'_1 \rightarrow_{L'_1} T'_2 \rightarrow_{L'_2} \cdots \rightarrow_{L'_{n-1}} T'_n,$$

by translating each rule L_i into a rule L'_i. (In order to avoid index translation, we allow some of the L'_i may be $L_\emptyset$, the artificial empty rule which does not derive anything) We make a case distinction:

- If $L_i = L_c([\alpha, \beta])$, then either
  - $\sigma(x) = [\alpha, \beta]$, then choose $L'_i = \emptyset$, or
  - $\sigma(x) \neq [\alpha, \beta]$, then $T'_{i+1} = T'_i \cup \{[\alpha', \beta']\}$ and this is a valid derivation.

  The case for symmetric or asymmetric encryption is analogous.

[2] See review questions for this proof.

– If $L_i = L_d([\alpha, \beta])$, then since there is no decomposition step $L_d(\sigma(x))$ in the original derivation, we know that $\sigma(x) \neq [\alpha, \beta]$, and hence $T'_{i+1} = T'_i \cup \{\alpha', \beta'\}$ for $[\alpha', \beta'] \in T'_i$, again a valid derivation.

This completes the proof of the lemma, as this is a contradiction to the minimality of (σ, o), since we know that $|\sigma(x)| > 1$. □

Review questions during semester: The above proof has some missing pieces, as indicated in the footnotes above:

– Why does it suffice to only prove derivability of the terms the adversary sends?
– Why do we know that the last rule in the derivation of $\sigma(x)$ can be assumed to be a compostion rule?
– Why can we assume that the derivation of $\sigma(r_j)$ contains no step $L_d(\sigma(x))$?

***Exercise Task:** parsing lemma proof* 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 precisely: 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 available to the adversary at the step where the first term containing $\sigma(x)$ is sent?

Size of a Minimal Attack

theorem
(σ, o) minimal successful attack on P, then $|\sigma(x)|_{\mathsf{DAG}} \leq |\{r_0, \ldots, r_n, s_0, \ldots, s_n\}|_{\mathsf{DAG}}$.

completes NP membership proof
- each x: representation of $\sigma(x)$ bound by protocol length
- number of variables is polynomial

therefore: polynomial representation of attack

proof (sketch)
- for every x with $\sigma(x) > 1$ ex. subterm t of P with $\sigma(t) = \sigma(x)$, t no variable
- every "long" $\sigma(x)$ is obtained by applying terms from the protocol
- replace long terms with references to protocol
  ⇝ small DAG size

173

In the following, we use the notation $\mathsf{Sub}(\mathsf{P})$ for the set of all subterms of all receive/send actions of a protocol P.

Theorem
Let (σ, o) be a minimal attack on a protocol P, then for all variables x occurring in the protocol, we have that $|x|_{\mathsf{DAG}} \leq |\{r_0, \ldots, r_n, s_0, \ldots, s_n\}|_{\mathsf{DAG}}$.

Proof For a set $U \subseteq \mathcal{V}$, we write $\overline{U}$ for the set $\{\sigma(y) \mid y \in U\}$. With V, we denote the set of variables appearing in P. We inductively construct a sequence of sets $S_i \subseteq \mathsf{Sub}(\mathsf{P})$ and sets V_i of variables such that $|\sigma(x)|_{\mathsf{DAG}} \leq \left|S_i \cup \overline{V_i}\right|_{\mathsf{DAG}}$.

The construction is as follows:

$i = 0$ We choose $S_0 = \emptyset$ and $V_0 = \{x\}$. The conditions are satisfied trivially.

$i \to i+1$ If $V_i = \emptyset$, we are done, since we then have $|\sigma(x)|_{\mathsf{DAG}} \leq \left|S_i \cup \overline{V_i}\right|_{\mathsf{DAG}} = |S_i|_{\mathsf{DAG}}$. Hence assume that there is some $y \in V_p$. Due to the Parsing Lemma, there exist a term $t \in \mathsf{Sub}(\mathsf{P})$, t no variable such that $\sigma(t) = \sigma(y)$. We define

- $S_{i+1} = S_i \cup \{t\}$
- $V_{i+1} = V_i \cup \mathcal{V}(t) \setminus \{y\}$. (Recall that $\mathcal{V}(t)$ is the set of variables appearing in t).

It remains to show that

$$|\sigma(x)|_{\mathsf{DAG}} \leq \left|S_{i+1} \cup \overline{V_{i+1}}\right|_{\mathsf{DAG}}.$$

By induction, we know that $|\sigma(x)|_{\mathsf{DAG}} \leq \left|S_i \cup \overline{V_i}\right|_{\mathsf{DAG}}$. It is therefore enough to show that $\left|S_i \cup \overline{V_i}\right|_{\mathsf{DAG}} \leq \left|S_{i+1} \cup \overline{V_{i+1}}\right|_{\mathsf{DAG}}$.
Recall that for a set S of terms, $|S|_{\mathsf{DAG}}$ is the number of subterms of S, It therefore suffices to construct an injective function

$$f\colon \mathsf{Sub}\left(\mathsf{S_i} \cup \overline{\mathsf{V_i}}\right) \longrightarrow \mathsf{Sub}\left(\mathsf{S_{i+1}} \cup \overline{\mathsf{V_{i+1}}}\right).$$

We define this function f for a subterm α as follows:

- if $\alpha \in \mathsf{Sub}(\mathsf{S_i})$, then we define $f(\alpha) = \alpha$. Since $S_i \subseteq S_{i+1}$, we have that $\alpha \in S_{i+1}$.
- if α is a subterm of some term $\overline{z}$ for a variable $z \in V_{i+1}$, then we also define $f(\alpha) = \alpha$.
- since $V_i \setminus \{y\} \subseteq V_{i+1}$, the remaining case is that α is a subterm of $\overline{y} = \sigma(t)$ and not of any other $\sigma(z)$ for any variable in V_{i+1}. Hence α is a subterm of t that is not a variable. Since $t \in S_{i+1}$, we can therefore define $f(\alpha) = \alpha$ again.

Obviously the construction terminates, since the size of $\sum_{y \in V_p} |\sigma y|$ decreases in each step. Finally, we obtain $V_p = \emptyset$ and therefore $|\sigma(x)|_{\mathsf{DAG}} \leq \left|S_p\right|_{\mathsf{DAG}}$ for a set $S_p \subseteq \{r_0, \ldots, r_n, s_0, \ldots, s_n\}$, as claimed. □

Review questions during semester:

- The argument showing termination of the construction only works if we can be sure that we do not have "cycles" in our definitions of variable values (e.g., a substitution with $\sigma(x) = \mathsf{hash}(x)$). Clearly such a substitution does not model any practically relevant attack, but does our formal model rule this out?
- The proof uses the parsing lemma to obtain the term t with $\sigma(t) = \sigma(y)$. The parsing lemma guarantees this term to be not a variable (otherwise, the parsing lemma would be trivial). Which role does the fact that t is not a variable play in the proof?

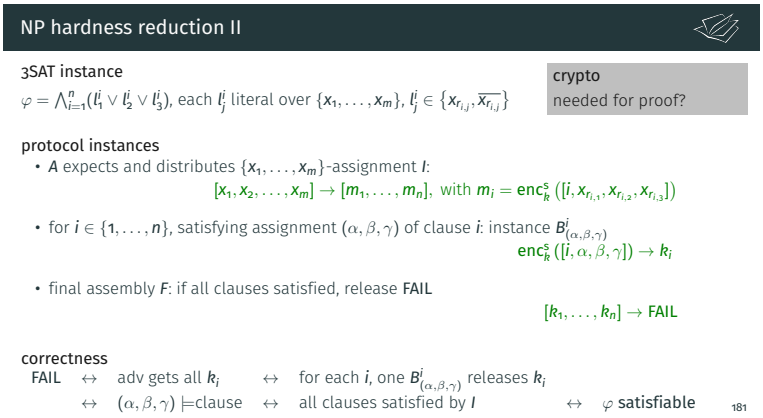
Exercise Task: *applying the Rusinowitch Turuani Theorem* In the lecture, modelled the Needham-Schroeder protocol as an input to INSECURE such that the attack is detected. However, this required us to already specify the "correct" sessions ("Alice with Charlie, Charlie with Bob") manually. For automatic analysis, such a manual step should not be required. Can you come up with a pre-processing step that makes this manual step unnecessary?

More precisely: Can you come up with a mechanism translating a natural representation of a protocol (e.g., as the list of "intended instances" for a single session) into a protocol P such that

- P can be used as input for the Rusinowitch-Turuani algorithm for INSECURE,
- P contains all relevant protocol instances (i.e., an initiator with Alice's identity expecting to communicate with a responder with Charlie's identity, and a responder with Bob's identity expecting to communicate with an initiator with Alice's identity),
- P is formally insecure if and only if there is a successful attack on any number of sessions with any set of identities in which the original protocol is run?

Note: You do not need to make your constructions formal.

3.6.4 NP hardness



Proof Let $\varphi = \bigwedge_{i=1}^{n}(l_1^i \vee l_2^i \vee l_3^i)$ be a 3SAT instance, where each l_j^i is a literal over $\{x_1,\dots,x_m\}$. Without loss of generality, we assume that each clause contains three distinct variables.[3] Let $r_{i,j}$ be the index of the variable that the literal l_j^i is based on, i.e., $l_j^i \in \{x_{r_{i,j}}, \overline{x_{r_{i,j}}}\}$.

We introduce the following instances:

- Instance A expects the adversary to supply an assignment to the variables $\{x_1,\dots,x_m\}$ and "splits up" the assignment to the participants, who check the individual clauses:

[3] A standard result shows that 3SAT remains NP-complete for this restriction. Note also that the restriction is only used in the proof to simplify presentation: The assumption implies that there are exactly 7 satisfying assignments for each clause, so we have 7 B_j^i-participants for each clause.

For each clause, A sends the values relevant for that clause. For this, A has a single r/s rule

$$[x_1, x_2, \ldots, x_m] \rightarrow [m_1, \ldots, m_n],$$

where m_i contains the values for the variables occurring in clause i, together with a label identifying the clause:

$$m_i = \mathsf{enc}^{\mathsf{s}}_k\left([i, x_{r_{i,1}}, x_{r_{i,2}}, x_{r_{i,3}}]\right).$$

The number i is a constant occurring in the protocol. Note that all the messages m_i are sent by one single instance of A, in a single step. This ensures that all of the m_i are fragments of the same propositional assignment.

– For each clause i, we introduce 7 instances[4] $B^i_1, \ldots, B^i_7$ that each "accept" one of the 7 possible satisfying assignments for this clause, and if the clause is satisfied, release a part of the "secret" that the adversary has to obtain. Since each clause has three distinct variables, there are exactly 7 possible satisfying assignments for each clause. For every triple (α, β, γ), such that the assignment I defined as $I(x_{r_{i,1}}) = \alpha$, $I(x_{r_{i,2}}) = \beta$, and $I(x_{r_{i,3}}) = \gamma$ it holds that $I \models (l^i_1 \vee l^i_2 \vee l^i_3)$, we introduce a participant with a single receive/send action

$$\mathsf{enc}^{\mathsf{s}}_k\left([i, \alpha, \beta, \gamma]\right) \rightarrow k_i.$$

– Finally, we introduce a participant F, who emits the constant FAIL if the adversary was able to collect all required "partial secrets" $k_1, \ldots, k_n$:

$$[k_1, \ldots, k_n] \rightarrow \mathsf{FAIL}.$$

As initial adversary knowledge, we define $I = \{0, 1\}$. Clearly, the reduction can be performed in polynomial time, since the number of protocol instances created is polynomial in the formula size, and each protocol instance itself is defined by a simple pattern.

The reduction is obviously correct: Since the adversary does not know (and cannot learn) the key k, the communication between the introduced honest participants is secure. Therefore, in a complete protocol run, the adversary obtains the value k_i if and only if the assignment sent to A satisfies the clause $(l^i_1 \vee l^i_2 \vee l^i_3)$. Since the adversary obtains the value FAIL if and only if he obtains all values $k_1, \ldots, k_n$, an attack is successful if and only if the message sent to A encodes a satisfying assignment for φ. □

Note that a possibly more natural proof would use signatures instead of encryptions (the construction is the same otherwise). We use encryption instead of signatures here to be consistent with the restriction of cryptographic operators in the section proving the Rusinowitch-Turuani Theorem.

[4] Note that the notation is slightly different here than in the slide, the presentation here makes the construction more explicit.

Review questions during semester: The above construction relies on the fact that in our modeling, the adversary can "activate" the distributing instance A only once, and so no parts of different, incompatible propositional assignments are distributed. This approach does not work anymore if we allow the adversary to start more than one copy of A. However, the decision problem clearly does not get any easier with this modification, so NP-hardness should still hold. Can you (informally) suggest a modification for the above proof that also works in this case?

3.7 Automatic Analysis: Undecidability

3.7.1 Arbitrarily Many Sessions

The "ffgg" Protocol

protocol

1	$A \to B$	A
2	$B \to A$	$[N_1, N_2]$
3	$A \to B$	$\mathsf{enc}^{\mathsf{a}}_{k_B}([N_1, \underbrace{N_2}_{=:x}, \underbrace{FAIL}_{=:y}])$
4	$B \to A$	$[N_1, x, \mathsf{enc}^{\mathsf{a}}_{k_B}([x, y, N_1])]$

more precisely

step 3:

- B verifies N_1
- B does not verify correctness of N_2
- matches N_2 with variable x, FAIL with variable y

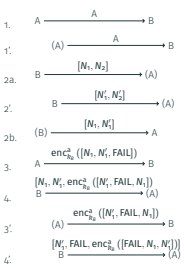
187

– Note that Bob encrypts a message to Alice with his own public key (so Alice cannot decrypt this message). This is intentional.

– The "flaw" in the protocol is that, by design, there are two (parts of) messages that can be exchanged with each other: The encryption generated in the third step has the same pattern as the encryption performed in the last step. However, the order of the elements in these two encryptions are different, in particular, the position of the FAIL constant. Therefore, the encryption generated in the fourth step can be used as input to the fourth step of a different protocol session.

Security of ffgg

attack



security
- there is an attack
- attack requires 2 responder instances
- fact: protocol is secure if there is only one instance

consequence
- analysing a single instance is not enough
- generalization: arbitrarily many instances
- analysis of unbounded number of instances required
- not covered by Rusinowitch Turuani

188

The attack is based on using the message obtained in the final step of the "left-hand" protocol session as input for the fourth step of the "right-hand" protocol session. It is important that Bob in fact checks that the nonce N_1 is correct. Clearly the attack would be successful also if Bob would not perform this check, however in that case there would be an attack involving only one responder instance, whereas the purpose of the protocol is to ensure that two protocol sessions are required.

Exercise Task: *the FFGG prototocol: too complicated?* 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?

Unbounded Version of INSECURE

required

analysis of extension of **INSECURE** to (arbitrarily many) parallel sessions

formalization
- input to algorithm may not contain explicit sessions anymore
- alternative: "template" for instances
  - instance $\mathcal{I}_{A\to B}$ may be started arbitrarily often
    - "between *A* and *B*"
    - "between *A* and *C*"
    - …

issue

FAIL-rule may only be contained in "relevant" instance

190

In general, as discussed with the Needham-Schroeder example before (see page 30), placement of the BREAK/FAIL rules is a nontrivial task, even more so with the unbounded sessions case. We will see a concrete example for this issue later in the lecture with ProVerif examples.

Exercise Task: *unbounded instances formalization* 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.

Note: You do not need to make your constructions formal. The goal of this exercise is to get a good understanding on how a formal definition of INSECURE (which we did not fully state in the lecture) would look like.

Undecidability Proof III

input
$(x_1, y_1), \ldots, (x_n, y_n)$ PCP instance,
$x_i = x_1^i \ldots x_{|x_i|}^i$, $y_i = y_1^i \ldots y_{|y_i|}^i$

idea
adversary can use protocol instances to initialize domino sequence or add new tile

instances
- for each $i \in \{1, \ldots, n\}$: A^i_{init} $\quad \epsilon \to \text{enc}^s_k\left([x^i_{|x_i|}, [x^i_{|x_i|-1}, [\ldots, x^i_1]\ldots]], [y^i_{|y_i|}, [y^i_{|y_i|-1}, [\ldots, y^i_1]\ldots]]\right)$
- f.e. i: A^i_{step} $\quad \text{enc}^s_k([x, y]) \to \text{enc}^s_k\left([x^i_{|x_i|}, [x^i_{|x_i-1|}, [\ldots, [x^i_1, x]]]], [y^i_{|y_i|}, [y^i_{|y_i-1|}, [\ldots, [y^i_1, y]]]]\right)$
- verification B_{check}: $\quad \text{enc}^s_k([x, x]) \to \text{FAIL}$

correctness
- A^i_{init}, A^i_{step}: Adversary gets exactly $\text{enc}^s_k([t_1, t_2])$, where t_1, t_2 constructed by "domino rules"
- B_{check}: if adversary solves domino puzzle, release FAIL constant
- so: domino solvable iff protocol insecure in unbounded setting

199

Proof Let $(x_1, y_1), \ldots, (y_1, y_n)$ be an instance of the PCP. For $i \in \{1, \ldots, n\}$, we denote x_i with $x_1^i \ldots x_{|x_i|}^i$, and analogously we denote y_i with $y_1^i \ldots y_{|y_i|}^i$. From this, we construct the following set of protocol instances:

– for every $i \in \{1, \ldots, n\}$, we construct an instance A^i_{init}, defined with a single receive/send action
$$\epsilon \to \text{enc}^s_K\left([x^i_{|x_i|}, [x^i_{|x_i|-1}, [\ldots, x^i_1]\ldots]], [y^i_{|y_i|}, [y^i_{|y_i|-1}, [\ldots, y^i_1]\ldots]]\right).$$
– for every $i \in \{1, \ldots, n\}$, we construct an instance A^i_{step}, defined with the single action
$$\text{enc}^s_K\left([x, y]\right) \to \text{enc}^s_K\left([x^i_{|x_i|}, [x^i_{|x_i-1|}, [\ldots, [x^i_1, x]]]], [y^i_{|y_i|}, [y^i_{|y_i-1|}, [\ldots, [y^i_1, y]]]]\right).$$

These instances allow the attacker to construct every pair of sequences that can be generated from the PCP instance, i.e., all sequences of the form $[x_{i_1} \ldots x_{i_\ell}, y_{i_1} \ldots y_{i_\ell}]$. We construct a final instance that (analogously to the instance F from our NP hardness proof) verifies whether the resulting sequence is in fact a solution to the PCP.

– B_{check}:
$$\text{enc}^s_K([x, x]) \to \text{FAIL}.$$

The instance B_{check} is marked as the analysis goal.

We define the initial attacker knowledge as the set $\{0, 1\}$. The encryption with the key K (unknown to the attacker) ensures that the terms processed by the A and B instances are in fact produced by instances of the form A^i_{init} or A^i_{step}.

Clearly, the reduction function is computable. We now show that it is correct. Hence first assume that the PCP instance is solvable, i.e., there exists a sequence $i_1, \ldots, i_\ell$ such that $x_{i_1} x_{i_2} \ldots x_{i_\ell} =$

$y_{i_1} y_{i_2} \dots y_{i_\ell}$. Then there is a protocol P_{unb} based on the protocol P that contains the following instances:

- One instance of $A^{i_1}_{\text{init}}$,
- for each $i \in \{1, \dots, n\}$, as many instances of A^{i}_{step} as the value i appears in $i_1, \dots, i_\ell$, where each protocol instance uses a fresh set of variables (For i_1, one instance fewer would suffice),
- a single instance of B_{check}.

This protocol is insecure, as the following attack shows:

- The execution order is $A^{i_\ell}_{\text{init}}$, $A^{i_{\ell-1}}_{\text{step}}$, …, $A^{i_1}_{\text{step}}$, B_{check}.
- The substitution is obtained by simply using the output received in one step as input for the next-activated instance.
- Clearly, the final term sent by $A^{i_1}_{\text{step}}$ is the reverse of $(x_{i_1} x_{i_2} \dots x_{i_\ell}, y_{i_1} y_{i_2} \dots y_{i_\ell})$ (with nested sequences expressing string concatanation), and hence the instance B_{check} releases the value FAIL.

For the converse, assume that there is an attack on some protocol P_{unb} based on P. Without loss of generality, we can assume that the instance activated in the last step is B_{check}. Since each instance A_{step} expects an input encrypted with the key K, the adversary can only deliver messages generated by honest principals to any of these instances. Therefore, the terms delivered to B_{check} are obtained using the honest instances in the above way, and thus the sequence of activated A_{check} instances yields a solution to the PCP problem. □

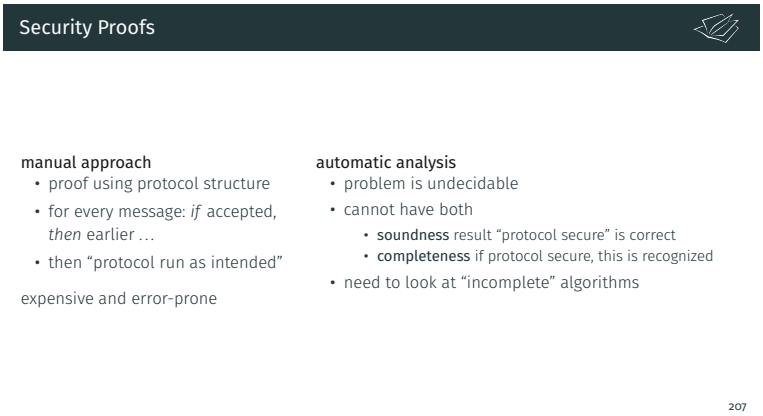
Review questions during semester:

- The reduction here (translating a PCP instance into a protocol instance) is very similar to the reduction establishing NP-hardness for INSECURE, and in fact the reduction in this undecidability proof can be performed in polynomial time. Therefore, we use the same tool (polynomial-time many-one reduction, expressed with $\leq^p_m$) to show NP-hardness in one case and undecidability in the other case. Can we therefore also show e.g., undecidability of INSECURE and/or NP-hardness of UNBOUNDED-INSECURE?
- (This question is out of scope for the current lecture, and only of interest to readers with background in cryptography.) The proof of the undecidability result (and simimarly the proof of the NP-hardness result earlier) relies on a property of encryption schemes that is ensured by our Dolev-Yao modeling, but not part of all security requirements for a (computational) encryption scheme. Which property do we require here?

Exercise Task: *Rusinowitch-Turuani with specified maximal number of sessions* We saw in the lecture that the "unbounded session" version of INSECURE is undecidable. A weaker version of that problem can be obtained by allowing instances to INSECURE to be accompanied by a maximal number of copies in which the adversary may start the corresponding protocol

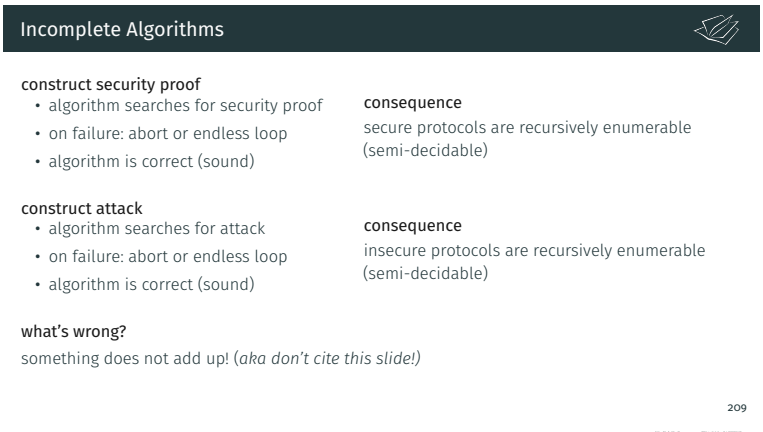
instance (we assume a mechanism that automatically renames variables to ensure that they are "local" to the copy in which they are used). Does the "positive" part of the Rusinowitch-Turuani theorem still hold for this generalization?

Hint: You are not expected to give a formal proof of your conjectures, an informal justification suffices. Also, be explicit about how the "maximal number of copies" is specified in the input to your generalized problem.



The discussion of soundness on this slide is a bit imprecise, since we do not properly distinguish here between algorithms that have a computable bound on the running time and algorithms in the "recursive enumeration" sense. See next slide for a more precise discussion of these issues.

3.7.2 Incomplete Algorithms



The issue with the argument is as follows: If both security and insecurity for protocols are semi-decidable, then both problems are decidable. This is implied by a standard result from foundations of theory, but also very natural: Simply let one algorithm search for the security proof, let another one search for an attack, and wait until one of them is successful. This "parallel" algorithm then allows to decide security. However, we just proved that (in-)security

is undecidable, so obviously one of the semi-decidability results must be wrong. Can you determine which one is incorrect?

Exercise Task: *Needham-Schroeder as Horn clauses* Model the Needham-Schroeder protocol as Horn clauses and use this formalism to show that the protocol is insecure. To do this, first list the facts, Dolev-Yao deductions, protocol deductions and the target clause. Then, use logical inference to show that the protocol is in fact insecure. Do you see any limits or imprecisions in this approach?

3.8 Automatic Analysis in Practice: ProVerif

3.8.1 Hello World and simple Examples

The Handshake Protocol

example: handshake (from ProVerif tutorial)

$A \to B \quad k_A$
$B \to A \quad \mathrm{enc}^{\mathrm{a}}_{k_A}(\mathrm{sig}_{k_B}([k_B, k]))$
$A \to B \quad \mathrm{enc}^{\mathrm{s}}_{k}(\mathrm{FAIL})$

properties?
- intended security properties?
- protocol secure?

attack

$\mathcal{A} \to B \quad k_{\mathcal{A}}$
$B \to \mathcal{A} \quad \mathrm{enc}^{\mathrm{a}}_{k_{\mathcal{A}}}(\mathrm{sig}_{k_B}([k_B, k]))$
$A \to B \quad k_A$
$\mathcal{A} \to A \quad \mathrm{enc}^{\mathrm{a}}_{k_A}(\mathrm{sig}_{k_B}([k_B, k]))$
$A \to B \quad \mathrm{enc}^{\mathrm{s}}_{k}(\mathrm{FAIL})$

fix
add receiver to message

225

Clearly, the protocol does not give any security guarantees to Bob. The only reasonable security property we can analyze here is that it is Alice's goal to learn a key that is generated by Bob, and no participant except Alice and Bob can learn the key during the protocol run. (The protocol is not intended to be secure, it serves as an example in the ProVerif documentation.)

ProVerif: Modeling Protocols With Crypto

lecture so far
behavior of crypto primitives
- asymmetric encryption
- symmetric encryption
- signatures
- hash functions

ProVerif: flexible modeling of primitives
technique: equational theories, more general than DY-like specification ısee example scripts

227

Review questions after semester: Equational theories provide a much more general way of

specifying cryptographic primitives than the Dolev-Yao style modeling we used in our theoretical model. Therefore, our results (in particular, the Rusinowitch-Turuani Theorem) do not cover this more general setting. For which equational theories will a generalization of the Rusinowitch-Turuani theorem be easy (e.g., follow with essentially the same proof)?

3.8.2 Equational Theories

Crypto Primitives: Generalizing Dolev-Yao

definition
- equation: pair (l, r) of terms, also written $l = r$
  - left: "complex" term
  - right: "simple" term
- equational theory: set E of equations

caveat: choose term signature matching to E (implicit)

rewrite relation
- $t_1 \twoheadrightarrow_E t_2$: t_2 obtained from t_1 by applying rule from E
- $\twoheadrightarrow_E^*$: closure of $\twoheadrightarrow_E$ under transitivity, reflexivity, application of function symbols
- $\equiv_E$: closure of $\twoheadrightarrow_E^*$ under symmetry and transitivity

229

As an example, the following equational theory models symmetric and asymmetric encryption:

- $\mathsf{dec}^{\mathsf{s}}_k\left(\mathsf{enc}^{\mathsf{s}}_k(x)\right) = x$
- $\mathsf{dec}^{\mathsf{a}}_{\hat{k}_A}\left(\mathsf{enc}^{\mathsf{a}}_{k_a}(x)\right) = x$

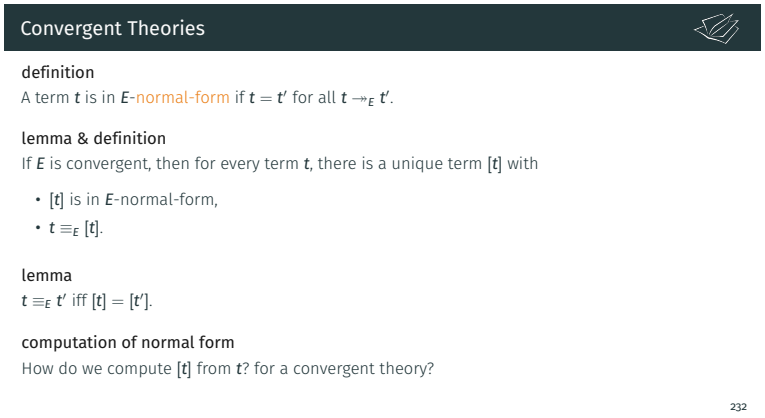
In this example, $\hat{\cdot}$ is an operator on the set of keys. We can also model signatures as an equational theory:

- $\mathsf{check}(k_A, \mathsf{sig}_{k_A}(x)) = \mathsf{ok}$
- $\mathsf{extr-key}(\mathsf{sig}_{k_A}(x)) = k_A$
- $\mathsf{extr-msg}(\mathsf{sig}_{k_A}(x)) = x$

Review questions during semester: We have now seen three ways to specify cryptographic primitives:

1. The Dolev-Yao style definitions in our theoretical model,
2. the stepwise definitions used in the proof of polynomial-time computability of the Dolev-Yao closure,
3. equational theories.

All these definitions work on a similar level of abstraction. What are the advantages and disadvantages of each of the definitions?



We prove the two lemmas:

Proof 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 $t \equiv_E [t] = [t'] \equiv_E t'$.

□

Equational Theories: Examples Revisited

primitives
- asym. encryption $\mathsf{dec}^{\mathsf{a}}_{k_A}\left(\mathsf{enc}^{\mathsf{a}}_{k_a}(x)\right) = x$
- sym. encryption $\mathsf{dec}^{\mathsf{s}}_{k}\left(\mathsf{enc}^{\mathsf{s}}_{k}(x)\right) = x$
- signature
  - $\mathsf{check}(k_A, \mathsf{sig}_{k_A}(x)) = \mathsf{ok}$
  - $\mathsf{extr-key}(\mathsf{sig}_{k_A}(x)) = k_A$
  - $\mathsf{extr-msg}(\mathsf{sig}_{k_A}(x)) = x$
- hash function
- bit commitment $\mathsf{open}(\mathsf{bc}(k, b), k) = b$
- trapdoor commitments
  - $\mathsf{open}(\mathsf{tdc}(m, r, t_d), r) = m$
  - $\mathsf{tdc}(m_2, f(m_1, r, t_d, m_2), t_d) = \mathsf{tdc}(m_1, r, t_d)$

discussion
- complexity?
- observations?

remember
modeling primitives at this level of abstraction loses details

algorithms
algorithms discussed so far do not cover all of these

233

The equations for bit commitment and for symmetric encryption are identical, which demonstrates that not all properties of real systems are expressed in our abstraction. The trapdoor commitment might seem counter-intuitive, the key idea is that given the "trapdoor" t_d, one can choose the r-argument in the computation of the commitment to m_2 to lead to the exact same (on a term level) result as the commitment to m_1.

Exercise Task: *Missing Proof* 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 $[t] = [t']$.

Exercise Task: *"Badly-Behaved" Equational Theories* 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.

Protocol Notation with Equational Theories

Needham Schroeder

Alice
$\epsilon \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$
$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, y) \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_B}(y)$

Bob
$\mathsf{enc}^{\mathsf{a}}_{k_B}(A, x) \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_A}(x, N_B)$

equational theory
$\mathsf{dec}^{\mathsf{a}}_{k_A}\left(\mathsf{enc}^{\mathsf{a}}_{k_a}(x)\right) = x$

"simplification"
- new notation for protocols?
- advantage?

additional equations for pairing
- $\mathsf{split1}(\mathsf{pair}(x, y)) = x$
- $\mathsf{split2}(\mathsf{pair}(x, y)) = y$

Alice
$\epsilon \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_B}(A, N_A)$
$x \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_B}\left(\mathsf{split2}\left(\mathsf{dec}^{\mathsf{a}}_{k_A}(x)\right)\right)$

Bob
$y \rightarrow \mathsf{enc}^{\mathsf{a}}_{k_A}\left(\mathsf{pair}\left(\mathsf{split2}\left(\mathsf{dec}^{\mathsf{a}}_{k_B}(y)\right), N_B\right)\right)$

237

An advantage is a more operational representation, which means that from the receive/send

actions, their implementation (using library access to the implementation of the cryptographic primitives) is given by the rules themselves. In particular, a discussion about "well-formed" protocols (see discussion of slide on page **??**) is no longer necessary.

Handshake-Protocol in ProVerif: Alice Process

protocol

$A \to B \quad k_A$
$B \to A \quad \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{sig}_{k_B}([k_B, k]))$
$A \to B \quad \mathsf{enc}^{\mathsf{s}}_{k}(\mathsf{FAIL})$

client (Alice)

```
let clientA(pkA:pkey,skA:skey,pkB:spkey)=
    out (c,pkA)
    in (c,x:bitstring)
    let y=adec(x,skA) in
        let (=pkB,k:key)=checksign(y,pkB) in
            out (c,senc(FAIL,k))
```

features
- keys as arguments
- FAIL global value
- decryptions explicit (pattern matching in formal model)

240

Review questions during semester: The notation for protocols used in ProVerif is, as discussed above, more operational and hence—at least from a programmer's perspective—more natural than the one used in our formal model. Why did we not use a similar operational notation in the definition of our formal model?

Handshake-Protocol in ProVerif: Bob Process

protocol

$A \to B \quad k_A$
$B \to A \quad \mathsf{enc}^{\mathsf{a}}_{k_A}(\mathsf{sig}_{k_B}([k_B, k]))$
$A \to B \quad \mathsf{enc}^{\mathsf{s}}_{k}(\mathsf{FAIL})$

server (Bob)

```
let serverB(pkB:spkey,skB:sskey)=
    in (c,pkX:pkey)
    new k:key
    out (c,aenc(sign((pkB,k),skB),pkX))
    in c,x:bitstring
    let z=sdec(x,k) in o
```

features
- type checking on message receive
- last line: "no match" if decryption fails

241

Review questions during semester: What is the point of the last line of Bob's specification? The instruction "0" simply does nothing, so whether the pattern-matching is successful does not have any influence on the execution of the protocol.

Exercise Task: *ProVerif example I* 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.

Exercise Task: *ProVerif example II* 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?

You can use any protocol you find interesting—all the protocols mentioned in the course so far are good candidates. The following is an incomplete list:

- your modeling of the WhatsApp authentication protocol in the first exercise,
- the (broken) authentication protocols presented in the first exercise class and their fixes,
- the Needham-Schroeder(-Lowe) protocol,
- the Woo-Lam protocol,
- the ffgg protocol,
- the repaired version of the handshake-protocol from the ProVerif tutorial (the broken version was discussed in the lecture).

Remember that most protocols are only insecure if the "correct" sessions are initiated. You can use a generic mechanism to start sessions, or use hard-coded sessions in your evaluation.

3.8.3 Randomized Encryption

3.8.4 Typing

3.8.5 Syntax: Pi-Calculus

3.8.6 Forward Secrecy

Diffie Hellman Key Exchange [DH76]

task
- Alice and Bob agree on a secret key
- use public (authenticated) channel

protocol

A, B			choose public values g, p
A			chooses secret value a
B			chooses secret value b
A	$\rightarrow$	B	g^a
B	$\rightarrow$	A	g^b
A			compute $(g^b)^a = g^{ab}$
B			compute $(g^a)^b = g^{ab}$
A, B			use g^{ab} as key

security
- can compute a from g, g^a: discrete logarithm
- choose structure where logarithm is hard

discrete logarithm
- structure: $\mathbb{Z}_p$ for prime p
- g generator of $\mathbb{Z}_p^*$
- DH assumption implies: logarithm computationally infeasable in $\mathbb{Z}_p^*$

what have we gained?

adversary can still store g, g^a, g^b and solve logarithm 20 years later

268

The Diffie-Hellman protocol of course cannot protect against future attacks on the discrete logarithm itself. For example, an attacker can record the network traffic of the protocol and solve the logarithm, allowing to recover the agreed session key and decrypt subsequent communication using this session key. The crucial point here is that solving the discrete logarithm allows only to decrypt the communication for one single session, whereas computation of a private key allows decrypting *all* communication encrypted with this key. In practice, session keys are therefore refreshed periodically such that solving the discrete logarithm only allows the attacker to decrypt messages exchanged in a short interval.

3.8.7 Strong Secrecy

Indistinguishability: Tests

definition; I-tests

for $I \subseteq \mathcal{T}$, an (atomic) I-test is a pair (M, M') of terms such that

- M and M' derivable from I (may contain E-destructors)
- in M and M' exactly one variable x occurs

definition: test semantics

message m satisfies test (M, M'), if $M[m/x] \equiv_E M'[m/x]$.

definition: indistinguishability

messages m and m' I-indistinguishable if there is no I-test satisfied by exactly one of m and m'.

275

Consider the following example:

- Let $t_1 = \mathsf{hash}(\mathrm{yes})$, $t_2 = \mathsf{hash}(\mathrm{no})$
- define two tests:
  - $(M_1, M_1') = (\mathsf{hash}(\mathrm{yes}), x)$
  - $(M_2, M_2') = (\mathsf{hash}(\mathrm{no}), x)$

  applying these tests to the terms t_1 and t_2, we get
  - $(M_1[t_1/x], M_1'[t_1/x]) = (\mathsf{hash}(\mathrm{yes}), \mathsf{hash}(\mathrm{yes}))$
  - $(M_2[t_1/x], M_2'[t_1/x]) = (\mathsf{hash}(\mathrm{no}), \mathsf{hash}(\mathrm{yes}))$

  Therefore, the first test is satistied by t_1, but the second one is not. analogously, the second test is satisfied by t_2, but the first is not. In particular, t_1 and t_2 are distinguishable via either of these tests.

Consider another example using encryption:

- With the same argument as above, $\mathsf{enc}^{\mathsf{a}}_{k_A}(\mathrm{yes})$ and $\mathsf{enc}^{\mathsf{a}}_{k_A}(\mathrm{no})$ are distinguishable (assuming, as always, that public keys are known).

We can model "secure" application of a hash function and of (asymmetric) encryption, even to known constants, using randomization:

- The terms $\mathsf{hash}([N_A, \mathrm{yes}])$ and $\mathsf{hash}([N_A, \mathrm{no}])$ are I-indistinguishable (if $N_A \notin I$).

– Similarly, $\mathsf{enc}^{\mathsf{a}}_{k_A}\left([N_A, \text{yes}]\right)$ and $\mathsf{enc}^{\mathsf{a}}_{k_A}([N_A, \text{no}])$ are I-indistinguishable (if $N_A, \hat{k}_A \notin I$).

(In)distinguishability Examples

distinguishable?

$I = \left\{k_A, k_B, k_C, \hat{k}_C, \text{yes}, \text{no}\right\}$

m	m'
$\mathsf{enc}^{\mathsf{a}}_{k_A}(\text{yes})$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(\text{no})$
N_A	N_B
$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \text{yes})$	$\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \text{no})$
$[\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A), \mathsf{enc}^{\mathsf{a}}_{k_A}(N_A)]$	$[\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A), \mathsf{enc}^{\mathsf{a}}_{k_A}(N_B)]$
$\mathsf{enc}^{\mathsf{a}}_{k_A}([N_A, N_A])$	$\mathsf{enc}^{\mathsf{a}}_{k_A}([N_A, N_B])$
$\mathsf{hash}(\text{yes})$	$\mathsf{hash}(\text{no})$
$\mathsf{hash}(N_A, \text{yes})$	$\mathsf{hash}(N_A, \text{no})$
$\mathsf{enc}^{\mathsf{a}}_{k_C}([N_A, \text{yes}])$	$\mathsf{enc}^{\mathsf{a}}_{k_C}([N_A, \text{no}])$
$[N_A, \mathsf{enc}^{\mathsf{a}}_{k_B}(N_A)]$	$[N_A, \mathsf{enc}^{\mathsf{a}}_{k_B}(N_B)]$

277

1. The first two are distinguishable by comparing them to $\mathsf{enc}^{\mathsf{a}}_{k_A}\left(yes\right)$, i.e., the test (M, M') is as follows:
   – $M = x$,
   – $M' = \mathsf{enc}^{\mathsf{a}}_{k_A}\left(yes\right)$.
   Clearly, the equality $M \equiv_E M'$ is true if x is substituted with $\mathsf{enc}^{\mathsf{a}}_{k_A}\left(yes\right)$, and false if x is substituted with any other term, in particular $\mathsf{enc}^{\mathsf{a}}_{k_A}(no)$.
2. The second two are indistinguishable (assuming that the adversary does not have access to the nonces).
3. The third pair is indistinguishable as well (the test from the first pair does not work here because we would need to access the nonce N_A).
4. The fourth pair is distinguishable because we can perform an equality test on the components of the pairs: Recall that M and M' may use E-destructors. Since we have pairing in our term signature, we also need to have access to projection functions, which extract specific components from pairs. If we denote the function extracting the first (second) component of a pair with proj_1 (proj_2), then the following test distinguishes the terms:
   – $M = \mathsf{proj}_1(x)$,
   – $M' = \mathsf{proj}_2(x)$.
   Clearly, the test is satisfied by a pair if and only if the two components of the pair contain the same value.
5. The fifth pair is indistinguishable.
6. These are obviously distinguishable.
7. Indistinguishable with the same argument as with asymmetric encryption above.
8. Obviously distinguishable since the attacker has access to $\hat{k}_C$.
9. Even though N_A and N_B are not known by the adversary, these terms are also distinguishable, by testing whether the second component is the encryption of the first component.

Exercise Task: *indistinguishability* 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 Task: *strong secrecy and derivation-based secrecy* 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.

3.8.8 Weak Secrecy

Popular Passwords

passwords		PINs	
1. password	14. abc123	1. 1234	14. 2468
2. 123456	15. mustang	2. 0000	15. 9999
3. 12345678	16. michael	3. 2580	16. 7777
4. 1234	17. shadow	4. 1111	17. 1996
5. qwerty	18. master	5. 5555	18. 2011
6. 12345	19. jennifer	6. 5683	19. 3333
7. dragon	20. 111111	7. 0852	20. 1999
8. pussy	21. 2000	8. 2222	21. 8888
9. baseball	22. jordan	9. 1212	22. 1995
10. football	23. superman	10. 1998	23. 2525
11. letmein	24. harley	11. 6969	24. 1590
12. monkey	25. 1234567	12. 1379	25. 1235
13. 696969		13. 1997	

287

Even people who know about security sometimes choose “weak” passwords [Tre19]. But in this case, it still took 39 years to crack them!

3.8.9 Beyond Secrecy: Correspondence Properties

Exercise Task: *secrecy properties and events* 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))** $\Rightarrow$ **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.

3.8.10 Incompleteness

3.8.11 ProVerif Summary

Exercise Task: *ProVerif modeling of Needham Schroeder* 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?

3.9 Crypto Protocols Summary

4 Part II: Information Flow

4.1 Examples

4.2 Introduction and Motivation

4.3 P-Security

4.3.1 Motivation and Definition

Noninterference: P-Security

definition (P-security)

A system $(S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ is P-secure with respect to a policy $\rightarrowtail$, if for all $u \in D$, $s \in S$, $\alpha_1, \alpha_2 \in A^*$ we have that:

$$\text{If } \texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2), \text{ then } \texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot \alpha_2).$$

intuition

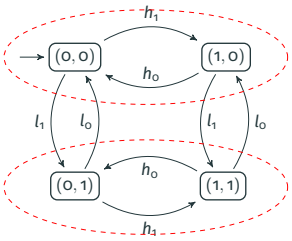
- α_1 and α_2 should "look the same" to u
- performing α_1 or α_2 from s should make no difference for u
- u should receive the same information from the system for both sequences

337

Review questions during semester: The definition of P-Security requires the condition to hold from every state in the system, not only for the initial state. Does requiring the condition to only hold from the initial state s_0 lead to a different notion of security?

P-Security Example

system



specification

- L observation: second component of state name
- h_x, l_x: actions of H, L
- policy: $L \rightarrowtail H$

analysis

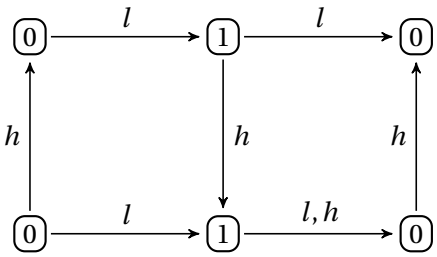
system secure?

- intuitively?
- formally?

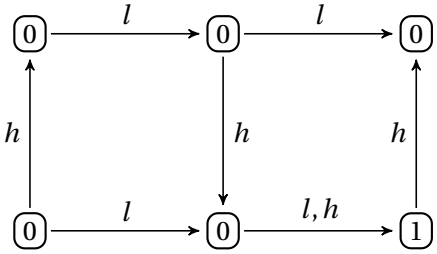
339

Review questions after semester: Which formal concept is indicated by the "circles" in the graphical representation of the system?

Exercise Task: *P-Security Example I* Is the following system P-secure? Justify your answer.



Exercise Task: *P-Security Example II* Is the following system P-secure? Justify your answer.



Exercise Task: *alternative definition of P security I* 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 traces $\alpha \in A^*$, we have that

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

Note: The characterization from this task is in fact the original definition of P-Security, the (equivalent, by the above) definition we work with in the lecture was later user by Ron van der Meyden.

4.3.2 Automatic Verification

Unwindings for P-Security

Definition

A P-unwinding for a system $(S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ and a policy $\rightarrowtail$ is a family of equivalence relations $(\sim_u)_{u \in D}$ on S such that

OC^P	if $s \sim_u t$, then $\texttt{obs}_u(s) = \texttt{obs}_u(t)$	output consistency
SC^P	if $s \sim_u t$, then $s \cdot a \sim_u t \cdot a$	step consistency
LR^P	if $\texttt{dom}(a) \not\rightarrowtail u$, then $s \sim_u s \cdot a$	left respect

theorem (Rushby, [Rus92])

A system M is P-secure with respect to $\rightarrowtail$ if and only if there is a P-unwinding for M and $\rightarrowtail$.

corollary

P-Security can be verified in polynomial time. (proof follows)

346

Review questions during semester: The definition of unwindings requires an equivalence

relation $\sim_u$ for every agent u in the system. However, in the examples covered in lecture and exercise class (with the usual H/L policy), we usually only present an unwinding $\sim_L$ for the "low" agent in the system. Why do we omit discussing an unwinding $\sim_H$ for the "high" agent?

Proof During this proof, we will only write purge instead of $\texttt{purge}_u$. First assume that an unwinding exists. We use the following fact:

Fact 1 *For any $\alpha \in A^*$, we have that $s \cdot \alpha \sim_u s \cdot \texttt{purge}(\alpha)$.*

Proof We show the claim by induction over $|\alpha|$. If $\alpha = \epsilon$, the claim follows since $\sim_u$ is reflexive. Hence assume the claim is true for α, we show that it also holds for αa, where $a \in A$. Due to induction, we know that $s \cdot \alpha \sim_u s \cdot \texttt{purge}(\alpha)$, we need to show that $s \cdot \alpha a \sim_u s \cdot \texttt{purge}(\alpha a)$. Note that $\texttt{purge}(\alpha a) = \texttt{purge}(\alpha)\texttt{purge}(a)$.

- *Case 1:* $\texttt{dom}(a) \rightarrowtail u$. In this case, the claim follows from step consistency: we know that $s \cdot \alpha \sim_u s \cdot \texttt{purge}(\alpha)$, and $\texttt{purge}(\alpha a) = \texttt{purge}(\alpha)a$. Applying step consistency yields: $s \cdot \alpha a \sim_u s \cdot \texttt{purge}(\alpha)a$, and hence $s \cdot \alpha a \sim_u s \cdot \texttt{purge}(\alpha a)$.
- *Case 2:* $\texttt{dom}(a) \not\rightarrowtail u$. We have the following equivalences:

$s \cdot \alpha$	$\sim_u$	$s \cdot \alpha a$	(left respect, since $\texttt{dom}(a) \not\rightarrowtail u$)
$s \cdot \alpha$	$\sim_u$	$s \cdot \texttt{purge}(\alpha)$	(induction)
$s \cdot \alpha a$	$\sim_u$	$s \cdot \texttt{purge}(\alpha)$	(transitivity)
$s \cdot \alpha a$	$\sim_u$	$s \cdot \texttt{purge}(\alpha a)$	($\texttt{purge}(\alpha) = \texttt{purge}(\alpha a)$, since $\texttt{dom}(a) \not\rightarrowtail u$).

This proves the claim. □

Now let $s \in S$, let $\alpha_1, \alpha_2 \in A^*$ with $\texttt{purge}(\alpha_1) = \texttt{purge}(\alpha_2)$. We need to show that $\texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot \alpha_2)$. Since $\texttt{purge}(\alpha_1) = \texttt{purge}(\alpha_2)$, and since $\sim_u$ is transitive, from the above fact we get $s \cdot \alpha_1 \sim_u s \cdot \texttt{purge}(\alpha_1) = s \cdot \texttt{purge}(\alpha_2) \sim_u s \cdot \alpha_2$. Due to output consistency, $s \cdot \alpha_1 \sim_u s \cdot \alpha_2$ implies $\texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot \alpha_2)$ as required.

For the converse, assume that M is P-secure. For each $u \in D$, we define the relation $\sim_u$ as follows:

$$s \sim_u t \text{ iff } \forall \alpha_1, \alpha_2 \in A^* \text{ with } \texttt{purge}(\alpha_1) = \texttt{purge}(\alpha_2) : \texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(t \cdot \alpha_2).$$

Since M is P-secure, $\sim_u$ is reflexive. The relation is trivially symmetric and transitive. It satisfies output consistency by definition (with $\alpha_1 = \alpha_2 = \epsilon$). We show LR^P and SC^P.

Left Respect (LR^P). Let $s \in S$ and $a \in A$ with $\texttt{dom}(a) \not\rightarrowtail u$. We need to show that $s \sim_u s \cdot a$. Hence let $\alpha_1, \alpha_2 \in A^*$ with $\texttt{purge}(\alpha_1) = \texttt{purge}(\alpha_2)$. We need to show that $\texttt{obs}_u(s \cdot \alpha_1) = \texttt{obs}_u(s \cdot a\alpha_2)$. Since $\texttt{dom}(a) \not\rightarrowtail u$, we have that $\texttt{purge}(a\alpha_2) = \texttt{purge}(\alpha_2)$. Therefore,

$$\begin{aligned} \texttt{obs}_u(s \cdot \alpha_1) &= \texttt{obs}_u(s \cdot \alpha_2) && \text{as } s \sim_u s \text{ and } \texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2) \\ &= \texttt{obs}_u(s \cdot a\alpha_2) && \text{as } s \sim_u s, \text{ and } \texttt{purge}_u(\alpha_2) = \texttt{purge}(a\alpha_2), \end{aligned}$$

as required.

Step Consistency (SC^P). Let $s \sim_u t$, and $a \in A$. We need to show that $s \cdot a \sim_u t \cdot a$. Hence let $\alpha_1, \alpha_2 \in A^*$ with $\mathtt{purge}_u(\alpha_1) = \mathtt{purge}_u(\alpha_2)$. We need to show that $\mathtt{obs}_u(s \cdot a\mathtt{purge}_u(\alpha_1)) = \mathtt{obs}_u(t \cdot a\mathtt{purge}_u(\alpha_2))$. Since $s \sim_u t$, it suffices to show that $\mathtt{purge}_u(a\mathtt{purge}_u(\alpha_1)) = \mathtt{purge}_u(a\mathtt{purge}_u(\alpha_2))$. This easily follows since

$$\begin{aligned} \mathtt{purge}(a\mathtt{purge}(\alpha_1)) &= \mathtt{purge}(a)\mathtt{purge}(\alpha_1) \\ &= \mathtt{purge}(a)\mathtt{purge}(\alpha_2) \quad = \mathtt{purge}(a\mathtt{purge}(\alpha_2)), \end{aligned}$$

completing the proof. □

Video Lecture The following slides are presented in the video "Characterization of P-Security with Unwindings" (`https://cloud.rz.uni-kiel.de/index.php/s/6k9DT475qW9NcQg`).

***Exercise Task:** uniqueness of unwindings* 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 $\rightarrowtail$.

Required: "Easier" Unwinding

lemma
If M is P-secure, then this algorithm constructs unwinding:

Input: $(S, A, \mathtt{step}, D, \mathtt{dom})$
  for each $u \in D$ do
    $\sim_u := \{(s,s) \mid s \in S\}$
    while elements added to $\sim_u$ do
      close $\sim_u$ under transitivity
      close $\sim_u$ under symmetry
      close $\sim_u$ under left respect
      close $\sim_u$ under step consistency
    end while
  end for

corollary
P-Security can be verified in polynomial time.

proof
Algorithm:

- construct $(\sim_u)_{u \in U}$ as in algorithm
- accept iff each relation satisfies output consistency

365

We now prove the lemma showing that if M is P-secure, then the family obtained by, for each agent $u \in D$, closing the relation $\{(s,s) \mid s \in S\}$ under the relevant properties is an unwinding. (The converse is trivial, since the presence of any unwinding implies P-security.) We refer to this relation as the "unwinding closure" in the following proof.

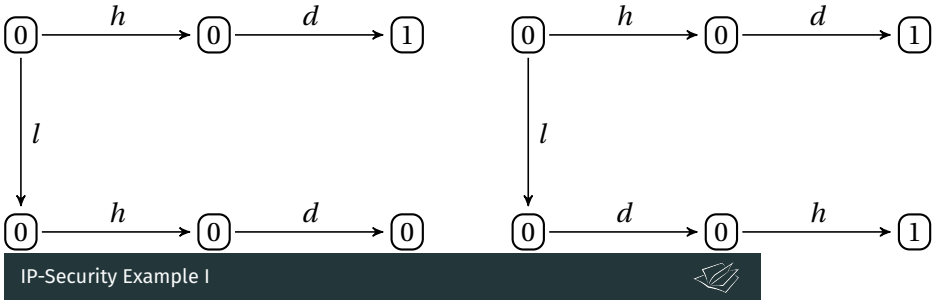
Proof Assume that M is P-secure. We then know, due to the characterization of P-security, that there is some P-unwinding $(\sim_u)_{u \in D}$ for M. By construction, $\sim_u$ is a superset of the unwinding closure for each $u \in D$. Since $\sim_u$ satisfies output consistency, it follows that the unwinding closure satisfies output consistency as well. Since the closure properties are satisfied

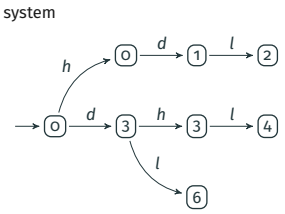
by construction, it follows that the unwinding closure is in fact an unwinding. □

4.4 IP-Security

4.4.1 Motivation and Definition

Exercise Task: *IP-Security examples* Which of the following systems are IP-secure? Assume that as usual, the state names indicate the observations made by L, that lowercase letters denote actions performed by agents with the corresponding higher-case letter name, and the policy $H \rightarrowtail D \rightarrowtail L$. Additionally, assume that H and D make the same observation in each state of the system.





specification
- intransitive policy: $H \rightarrowtail D \rightarrowtail L$
- actions h / d / l of agent H / D / L
- L's observations: indicated numbers

analysis
system secure?
- intuitively?
- formally?

375

The example does not satisfy IP-security. To show this, we first compute the sets of intransitve sources for the involved traces:

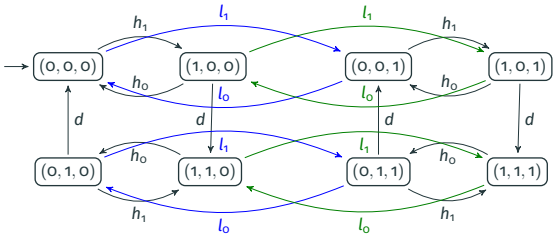
- $\texttt{sources}_L(\epsilon) = \{L\}$,
- $\texttt{sources}_L(l) = \{L\}$,
- $\texttt{sources}_L(dl) = \{D, L\}$,
- $\texttt{sources}_L(hdl) = \{H, D, L\}$,
- $\texttt{sources}_L(hl) = \{L\}$,
- $\texttt{sources}_L(dhl) = \{D, L\}$.

Using these results, we can now compute the intransitive purge of the relevant action sequences (traces):

- $\texttt{ipurge}_L(dhl) = dl$,
- $\texttt{ipurge}_L(dl) = dl$.

In particular, $\texttt{ipurge}_L(dhl) = \texttt{ipurge}_L(dl)$. However, with q_0 denoting the initial state, it follows that $\texttt{obs}_L(q_0 \cdot dl) = 6 \neq 4 = \texttt{obs}_L(dhl)$. Therefore, the system is not IP-secure.

IP-Security Example II



system

- intransitive policy $H \rightarrowtail D \rightarrowtail L$
- actions h_x, d, l_x of agents H / D / L
- $\texttt{obs}_L(a, b, c) = (b, c)$
- system secure? intuitively, formally?

376

By definition, we need to show that for traces α_1 and β_1, if $\texttt{ipurge}_L(\alpha_1) = \texttt{ipurge}_L(\alpha_2)$, then $\texttt{obs}_L(q_0 \cdot \alpha_1) = \texttt{obs}_L(q_1 \cdot \alpha_2)$. Note that in the example system, the following is true for any trace α. In the following, we say that a sub-sequence (or single action) of α is *purged*, if it is removed from α when computing $\texttt{ipurge}_L(\alpha)$.

- every purged subsequence only contains H actions, since $D \rightarrowtail L$ and $L \rightarrowtail L$.
- every purged subsequence appears after the last D action, since otherwise D downgrades all actions in the subsequence.

We claim that for a trace α, the observations of L can be read off $\texttt{ipurge}_L(\alpha)$ as follows: The observation is (b, c), where

- c is 0 or 1, depending on whether the last l-action performed was l_0 or l_1 (and 0 if no such pair of actions was performed),
- b is 0 or 1, depending on whether the last h-action performed before the last d-action was h_0 or h_1 (and 0 if no such action was performed).

We prove the claim by induction over α. The claim is clearly true if $\alpha = \epsilon$. We now consider each possible action performed:

- If h_0 or h_1 is played, then L's observation does not change, but the index of h's action is stored in the first component of the state.
- If d is played, the first component of the state (i.e., the index of H's last action) is copied to b.
- Playing l_0 or l_1 simply stores the index of the action as the last component of the state.

Due to the above, we know that the last H-action before the last D-action and all of L's actions are contained in $\texttt{ipurge}_L\alpha$. Therefore, L's observations in the state $(0,0,0)\cdot\alpha$ are in fact a function of $\texttt{ipurge}_L(\alpha)$.

P-Security and IP-Security

question
- two security properties: P-security, IP-security
- does either implication hold? guesses?

intuition
- IP-security is "relaxation" of P-security
- agents are allowed to have more information
- leads to less-strict security property

fact
If system M is P-secure wrt. $\rightarrowtail$, then also IP-secure wrt. $\rightarrowtail$.

converse?

377

We now prove that if a system is P-secure, then it is also IP-secure (with respect to the same policy).

Proof Assume that the system M is P-secure, let $\alpha_1 = \alpha_2$ be traces with $\texttt{ipurge}(\alpha_1) = \texttt{ipurge}(\alpha_2)$, and let u be an agent of the system. We need to show that $\texttt{obs}_u(s\cdot\alpha_1) = \texttt{obs}_u(s\cdot\alpha_2)$. Since M is P-secure, it suffices to show that $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$. This easily follows from the following:

For any trace α, we have that $\texttt{purge}_u(\alpha) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha))$: $\texttt{purge}_u(\alpha)$ contains exactly those actions from α for whose domain v we have $v \rightarrowtail u$. All of these actions are also still present in $\texttt{ipurge}_u$.

Therefore, we obtain

$$\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha_1)) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha_2)) = \texttt{purge}_u(\alpha_2),$$

as required. □

P-Security implies IP-Security

fact
If a system M is P-secure with respect to $\rightarrowtail$, then M is IP-secure with respect to $\rightarrowtail$. The converse is true for transitive policies.

proof
- assume M is P-secure
- agent u, state s, traces α_1, α_2 with $\texttt{ipurge}_u(\alpha_1) = \texttt{ipurge}_u(\alpha_2)$
- need to show: $\texttt{obs}_u(s\cdot\alpha_1) = \texttt{obs}_u(s\cdot\alpha_2)$
- enough to show: $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\alpha_2)$, since M is P-secure
- general: $\texttt{purge}_u(\alpha) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha))$
- so: $\texttt{purge}_u(\alpha_1) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha_1)) = \texttt{purge}_u(\texttt{ipurge}_u(\alpha_2)) = \texttt{purge}_u(\alpha_2)$

completes proof (see exercise for transitive case).

378

Review questions after semester: P-Security and IP-Security are both defined very similarly, where equality between $\texttt{purge}_u(\alpha_1)/\texttt{ipurge}_u(\alpha_1)$ and $\texttt{purge}_u(\alpha_2)/\texttt{ipurge}_u(\alpha_2)$ implies equality of the observations in the states reached by these sequences. The third notion of nonintererence we will consider will follow the same pattern with $\texttt{ta}_u$ instead of $\texttt{purge}_u$ or $\texttt{ipurge}_u$. Proving implications between these three security notions therefore consists of proving relationships between the three involved functions. Which relationship would we need to prove to show, e.g., that TA-Security implies IP-Security? (You can answer this question *without* having studied the definition of the `ta`-function.)

Exercise Task: *implications between security properties* 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 $\rightarrowtail$.

Exercise Task: *equivalence for transitive policies* 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 $\rightarrowtail$,

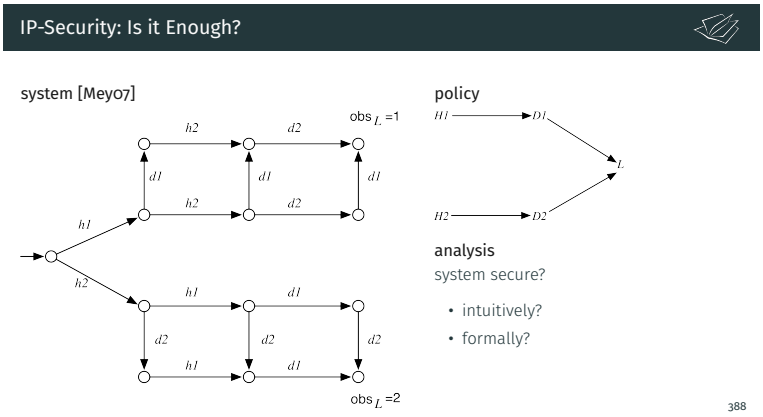
Exercise Task: *P-security and non-transitive policies* Prove or disprove the following: If $M = (S, s_0, A, \texttt{step}, D, O, \texttt{obs}, \texttt{dom})$ is a system and $\rightarrowtail$ is a policy for M, then the following are equivalent:

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

4.4.2 Automatic Verification

4.5 TA-Security

4.5.1 Motivation and Definition



It can easily be verified that the system is in fact IP-secure. However, intuitively, the system can be regarded as insecure, by considering these two traces:

- $\alpha_1 = h_1 d_1 h_2 d_2$
- $\alpha_2 = h_2 d_2 h_1 d_1$

With the indicated policy, it follows that

- $\texttt{ipurge}_L(\alpha_1) = h_1 d_1 h_2 d_2$
- $\texttt{ipurge}_L(\alpha_2) = h_2 d_2 h_1 d_1$

i.e., the `ipurge`-values of these traces are different. IP-security therefore does not require the states reached by these two traces to have the same observations. However, by observing either the output 1 or 2, the agent L learns whether H_1 or H_2 performed the first action. Intuitively, L should not have this knowledge, since neither D_1 nor D_2 know which of the H-agents performed the first action.

References

[18] *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018. ISBN: 978-1-5386-6680-7. URL: https://ieeexplore.ieee.org/xpl/conhome/8428826/proceeding.

[AC06] Martın Abadi and Véronique Cortier. “Deciding knowledge in security protocols under equational theories”. In: *Theoretical Computer Science* 367.1-2 (2006), pp. 2–32.

[AR02] Martín Abadi and Phillip Rogaway. “Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)”. In: *Journal of Cryptology* 15.2 (2002), pp. 103–127.

[Bla11] Bruno Blanchet. “Using Horn Clauses for Analyzing Security Protocols”. In: *Cryptology and Information Security Series* 5 (2011), pp. 86–111.

[BP03] Michael Backes and Birgit Pfitzmann. “Intransitive Non-Interference for Cryptographic Purpose”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2003, pp. 140–. ISBN: 0-7695-1940-7.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. “A General Composition Theorem for Secure Reactive Systems”. In: *TCC*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Springer, 2004, pp. 336–354. ISBN: 3-540-21000-8.

[Bra+18] Oliver Bracevac, Richard Gay, Sylvia Grewe, Heiko Mantel, Henning Sudbrock, and Markus Tasch. “An Isabelle/HOL Formalization of the Modular Assembly Kit for Security Properties”. In: *Archive of Formal Proofs* 2018 (2018). URL: https://www.isa-afp.org/entries/Modular%5C_Assembly%5C_Kit%5C_Security.html.

[Cam+19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. “iUC: Flexible Universal Composability Made Simple”. In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Springer, 2019, pp. 191–221. ISBN: 978-3-030-34617-1. DOI: 10.1007/978-3-030-34618-8_7. URL: https://doi.org/10.1007/978-3-030-34618-8%5C_7.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *FOCS*. 2001, pp. 136–145.

[CCT18] Vincent Cheval, Véronique Cortier, and Mathieu Turuani. “A Little More Conversation, a Little Less Action, a Lot More Satisfaction: Global States in ProVerif”. In: *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 344–358. ISBN: 978-1-5386-6680-7. DOI: 10.1109/CSF.2018.00032. URL: https://doi.org/10.1109/CSF.2018.00032.

[CFL19] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. “BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device”. In: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 367–381. ISBN: 978-1-7281-1407-1. DOI: 10.1109/CSF.2019.00032. URL: https://doi.org/10.1109/CSF.2019.00032.

[Cha+16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. “BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme”. In: *ACM Conference on Computer and Communications Security*. ACM, 2016, pp. 1614–1625.

[Chr+20] Rémy Chrétien, Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. “Typing Messages for Free in Security Protocols”. In: *ACM Trans. Comput. Log.* 21.1 (2020), 1:1–1:52. DOI: 10.1145/3343507. URL: https://doi.org/10.1145/3343507.

[CNN14] CNN. *E-voting experiments end in Norway amid security fears*. June 2014. URL: http://www.bbc.com/news/technology-28055678.

[Cor+18] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. “Machine-Checked Proofs for Electronic Voting: Privacy and Verifiabi-

lity for Belenios". In: *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 298–312. ISBN: 978-1-5386-6680-7. DOI: 10.1109/CSF.2018.00029. URL: https://doi.org/10.1109/CSF.2018.00029.

[Cor14] Véronique Cortier. "Electronic Voting: How Logic Can Help". In: *IJCAR*. 2014, pp. 16–25.

[CW17] Véronique Cortier and Cyrille Wiedling. "A formal analysis of the Norwegian E-voting protocol". In: *Journal of Computer Security* 25.1 (2017), pp. 21–57. DOI: 10.3233/JCS-15777. URL: https://doi.org/10.3233/JCS-15777.

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* IT-22.6 (1976), pp. 644–654.

[Dre+17] Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. "Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols". In: *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Ed. by Matteo Maffei and Mark Ryan. Vol. 10204. Lecture Notes in Computer Science. Springer, 2017, pp. 117–140. ISBN: 978-3-662-54454-9. DOI: 10.1007/978-3-662-54455-6.

[Dre+18] Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. "Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR". In: *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 359–373. ISBN: 978-1-5386-6680-7. URL: https://ieeexplore.ieee.org/xpl/conhome/8428826/proceeding.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–207.

[Egg+11] Sebastian Eggert, Ron van der Meyden, Henning Schnoor, and Thomas Wilke. "The Complexity of Intransitive Noninterference". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011, pp. 196–211. ISBN: 978-1-4577-0147-4.

[Egg+13] Sebastian Eggert, Ron van der Meyden, Henning Schnoor, and Thomas Wilke. "Complexity and Unwinding for Intransitive Noninterference". In: *CoRR* abs/1308.1204 (2013). URL: http://arxiv.org/abs/1308.1204.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. "A Practical Secret Voting Scheme for Large Scale Elections." In: *AUSCRYPT*. 1992, pp. 244–251.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. ISBN: 978-1-60558-506-2. DOI: 10.1145/1536414.1536440. URL: http://doi.acm.org/10.1145/1536414.1536440.

[Gjø10] Kristian Gjøsteen. "Analysis of an internet voting protocol". In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 380. URL: http://eprint.iacr.org/2010/380.

[GM82] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models". In: *IEEE Symposium on Security and Privacy*. 1982, pp. 11–20.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28420. URL: http://doi.acm.org/10.1145/28395.28420.

[Ham+18] Tobias Hamann, Mihai Herda, Heiko Mantel, Martin Mohr, David Schneider, and Markus Tasch. “A Uniform Information-Flow Security Benchmark Suite for Source Code and Bytecode”. In: *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*. Ed. by Nils Gruschka. Vol. 11252. Lecture Notes in Computer Science. Springer, 2018, pp. 437–453. ISBN: 978-3-030-03637-9. DOI: 10.1007/978-3-030-03638-6_27. URL: https://doi.org/10.1007/978-3-030-03638-6%5C_27.

[HLS03] James Heather, Gavin Lowe, and Steve Schneider. “How to Prevent Type Flaw Attacks on Security Protocols”. In: *Journal of Computer Security* 11.2 (2003), pp. 217–244. URL: http://content.iospress.com/articles/journal-of-computer-security/jcs162.

[HY87] J. Thomas Haigh and William D. Young. “Extending the Noninterference Version of MLS for SAT”. In: *IEEE Trans. on Software Engineering* SE-13.2 (Feb. 1987), pp. 141–150.

[JW18] Richard J. Lipton and Kenneth W. Regan. *Timing Leaks Everything*. 2018. URL: https://rjlipton.wordpress.com/2018/01/12/timing-leaks-everything.

[KEB18] Robert Künnemann, Ilkan Esiyok, and Michael Backes. “Automated Verification of Accountability in Security Protocols”. In: *CoRR* abs/1805.10891 (2018). arXiv: 1805.10891. URL: http://arxiv.org/abs/1805.10891.

[KK16] Steve Kremer and Robert Künnemann. “Automated analysis of security protocols with global state”. In: *Journal of Computer Security* 24.5 (2016), pp. 583–616. DOI: 10.3233/JCS-160556. URL: https://doi.org/10.3233/JCS-160556.

[KKT07] Detlef Kähler, Ralf Küsters, and Tomasz Truderung. “Infinite State AMC-Model Checking for Cryptographic Protocols”. In: *LICS*. IEEE Computer Society, 2007, pp. 181–192.

[KM17] Ralf Küsters and Johannes Müller. “Cryptographic Security Analysis of E-voting Systems: Achievements, Misconceptions, and Limitations”. In: *Electronic Voting - Second International Joint Conference*. Ed. by Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann. Vol. 10615. Lecture Notes in Computer Science. Springer, 2017, pp. 21–41. ISBN: 978-3-319-68686-8. DOI: 10.1007/978-3-319-68687-5. URL: https://doi.org/10.1007/978-3-319-68687-5.

[KR05] Steve Kremer and Mark Ryan. “Analysis of an Electronic Voting Protocol in the Applied Pi Calculus”. In: *ESOP 2005*. Ed. by Shmuel Sagiv. Vol. 3444. Lecture Notes in Computer Science. Springer, 2005, pp. 186–200. ISBN: 3-540-25435-8.

[KT07] Ralf Küsters and Tomasz Truderung. “On the Automatic Analysis of Recursive Security Protocols with XOR”. In: *STACS*. Ed. by Wolfgang Thomas and Pascal Weil. Vol. 4393. Lecture Notes in Computer Science. Springer, 2007, pp. 646–657. ISBN: 978-3-540-70917-6.

[KT08] Ralf Küsters and Tomasz Truderung. “Reducing protocol analysis with XOR to the XOR-free case in the horn theory based approach”. In: *ACM Conference on Computer and Communications Security*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM, 2008, pp. 129–138. ISBN: 978-1-59593-810-7.

[KT09] Ralf Küsters and Tomasz Truderung. “Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation”. In: *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*. IEEE Computer Society, 2009, pp. 157–171. ISBN: 978-0-7695-3712-2. DOI: 10.1109/CSF.2009.17. URL: https://doi.org/10.1109/CSF.2009.17.

[Küs06] Ralf Küsters. “Simulation-Based Security with Inexhaustible Interactive Turing Machines”. In: *CSFW*. IEEE Computer Society, 2006, pp. 309–320. ISBN: 0-7695-2615-2.

[KW11] Ralf Küsters and Thomas Wilke. *Moderne Kryptographie - Eine Einführung*. Vieweg + Teubner, 2011. ISBN: 978-3-519-00509-4.

[Lin17] Yehuda Lindell. “How to Simulate It - A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography*. Ed. by Yehuda Lindell. Springer International Publishing, 2017, pp. 277–346. ISBN: 978-3-319-57047-1. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8%5C_6.

[Lip+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. “Meltdown”. In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.01207.

[Low96] Gavin Lowe. “Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR”. In: *TACAS*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 1055. Lecture Notes in Computer Science. Springer, 1996, pp. 147–166. ISBN: 3-540-61042-1.

[Low99] Gavin Lowe. “Towards a Completeness Result for Model Checking of Security Protocols”. In: *Journal of Computer Security* 7.1 (1999), pp. 89–146. URL: http://content.iospress.com/articles/journal-of-computer-security/jcs118.

[Mey07] Ron van der Meyden. “What, Indeed, Is Intransitive Noninterference?” In: *European Symposium On Research In Computer Security (ESORICS)*. Ed. by Joachim Biskup and Javier Lopez. Vol. 4734. Lecture Notes in Computer Science. Springer, 2007, pp. 235–250. ISBN: 978-3-540-74834-2.

[Mey08] Ron van der Meyden. “On Notions of Causality and Distributed Knowledge”. In: *KR*. Ed. by Gerhard Brewka and Jérôme Lang. AAAI Press, 2008, pp. 209–219. ISBN: 978-1-57735-384-3.

[Mil99] Jonathan K. Millen. “A Necessarily Parallel Attack”. In: *In Workshop on Formal Methods and Security Protocols*. 1999.

[MPR19] Andrea Marin, Carla Piazza, and Sabina Rossi. “A Process Algebra for (Delimited) Persistent Stochastic Non-Interference”. In: *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*. Ed. by Alberto Casagrande and Eugenio G. Omodeo. Vol. 2396. CEUR Workshop Proceedings. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2396.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. "A Calculus of Mobile Processes, I and II". In: *Inf. Comput.* 100.1 (1992), pp. 1–77. DOI: 10.1016/0890-5401(92)90008-4. URL: http://dx.doi.org/10.1016/0890-5401(92)90008-4.

[NS78] Roger M. Needham and Michael D. Schroeder. "Using Encryption for Authentication in Large Networks of Computers". In: *Communications of the ACM* 21.12 (1978), pp. 993–999.

[NSC18] Thanh Binh Nguyen, Christoph Sprenger, and Cas Cremers. "Abstractions for security protocol verification". In: *Journal of Computer Security* 26.4 (2018), pp. 459–508. DOI: 10.3233/JCS-15769. URL: https://doi.org/10.3233/JCS-15769.

[Pos46] Emil L. Post. "A variant of a recursively unsolvable problem". In: *Bull. Amer. Math. Soc.* 52.4 (Apr. 1946), pp. 264–268. URL: https://projecteuclid.org:443/euclid.bams/1183507843.

[Rot16] Jörg Rothe, ed. *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*. Springer, 2016. ISBN: 978-3-662-47903-2. DOI: 10.1007/978-3-662-47904-9. URL: https://doi.org/10.1007/978-3-662-47904-9.

[RT03] Michaël Rusinowitch and Mathieu Turuani. "Protocol insecurity with a finite number of sessions, composed keys is NP-complete". In: *Theoretical Computer Science* 1-3.299 (2003), pp. 451–475.

[Rus92] John Rushby. *Noninterference, Transitivity, and Channel-Control Security Policies*. Tech. rep. CSL-92-02. SRI International, Dec. 1992. URL: http://www.csl.sri.com/papers/csl-92-2/.

[Sch12] Henning Schnoor. "Deciding Epistemic and Strategic Properties of Cryptographic Protocols". In: *ESORICS*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. Lecture Notes in Computer Science. Springer, 2012, pp. 91–108. ISBN: 978-3-642-33166-4.

[Sch78] T. J. Schaefer. "The complexity of satisfiability problems". In: *Proceedings 10th Symposium on Theory of Computing*. ACM Press, 1978, pp. 216–226.

[SM03] Andrei Sabelfeld and Andrew C. Myers. "Language-based information-flow security". In: *IEEE Journal on Selected Areas in Communications* 21.1 (2003), pp. 5–19. DOI: 10.1109/JSAC.2002.806121. URL: http://dx.doi.org/10.1109/JSAC.2002.806121.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001. ISBN: 978-0-521-78177-0.

[Tre19] Sylvester Tremmel. *UNIX-Prominenz wählte Schach-Eröffnung: 39 Jahre alte BSD-Passwörter geknackt*. 2019. URL: https://www.heise.de/newsticker/meldung/UNIX-Prominenz-waehlte-Schach-Eroeffnung-39-Jahre-alte-BSD-Passwoerter-geknackt-4554180.html.

[VP17] Mathy Vanhoef and Frank Piessens. *Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2*. 2017.

[War05] Bogdan Warinschi. "A computational analysis of the Needham-Schroeder-(Lowe) protocol". In: *Journal of Computer Security* 13.3 (2005), pp. 565–591.

[Win18] Christof Windeck. *PC und Notebook senden auf Mittelwelle - ohne Zusatz-Hardware*. 2018. URL: https://www.heise.de/ct/artikel/PC-und-Notebook-senden-auf-Mittelwelle-ohne-Zusatz-Hardware-3948910.html.

[WL92] Thomas Y. C. Woo and Simon S. Lam. "Authentication for Distributed Systems". In: *Computer* 25.1 (Jan. 1992), pp. 39–52. ISSN: 0018-9162. DOI: 10.1109/2.108052. URL: http://dx.doi.org/10.1109/2.108052.