

Engineering Secure Software Systems

Winter 2020/21: Complete Lecture

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

You hear me?

1, 2, 3, ...

Echo fail?

- Chat (I'll be half-watching)
- “Raise Hand”
- use your voice
- EMail

fast feedback

Participants (1)

Software Engineering (Host, me) muted

yes no go slower go faster more clear all

Mute All Unmute All More ▾



Introduction: Me

Bio Henning Schnoor

2004 diploma mathematics / computer science in Hannover

2004-7 PhD in Hannover advised by Prof. Heribert Vollmer
Algebraic Techniques for Satisfiability Problems

2007-8 Postdoc in Rochester, NY, USA

2008-15 Habilitation in Kiel, advised by Prof. Thomas Wilke
Knowledge-based, Strategic, and Temporal Security Properties

s. 2016 Privatdozent, AG Software Engineering, Prof. Wilhelm Hasselbring

Research interests

- formal methods for IT security
- metrics for software quality
- complexity theory
- logic
- computational social choice



Introduction: You

study program

- bachelor computer science
- master computer science
- Wirtschaftsinformatik
- mathematics
- physics
- others?



Admin

Overview

Admin

Accounts, Materials

Distance Learning in ESSS 20/21



Administrative

time and place

lecture tuesday, 12:15-13:45

exercise thursday, 9:00-9:45

url <https://uni-kiel.zoom.us/j/84262484452?pwd=bVNkeDJiWXk1c0ZpaE5sWnB0eUhDUT09>

passcode 699272

exercise and exam

- as usual: work in pairs
- exam: depending on number of students after \approx 6 weeks
 - oral exam
 - oral exam for active students
 - written exam
- no bonus points for exam grade
- dates depend on exam period (Prüfungszeitraum)

ESSS Exercise Class

exercise class goals

- apply concepts and formalisms from lecture yourself
- time for details and clarifications
- I hope: discussions and questions

exercise content: tasks

- tasks presented in lecture, also published as exercise sheet
- review questions from lecture
- work on tasks before discussion in exercise class
- hand-in/feedback using git repositories
- not required to solve tasks for good grade
- discussion in exercise class after hand-in
- first sheet: released today



materials: my repository

- slides, lecture notes, exercise sheets, modeling scripts ...
- *notes*: additional material for “marked” slides
 - background material, formal proofs, discussion
 - review questions

further reading

- no textbook
- references: original research papers

materials repo clone command

```
git clone https://hs:e3eLXb8-iLb3hoG7LRGo@git.informatik.uni-kiel.de/hs/esss-ws2021-common.git
```

exercise solutions: your repository

- form groups of 2 students to work together
- create GitLab project to hand-in exercises



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 abbreviation for the lecture, (e.g., **SEPV5** for Software Engineering für Parallele und Verteilte Systeme or **ESS5** for Engineering Secure Software Systems), **SEM** is an abbreviation 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.

Overview

Admin

Accounts, Materials

Distance Learning in ESSS 20/21



Distance Learning: My Experience

on-campus

- direct interaction with students
- judge your reactions, see what works and what does not
- make corresponding adjustments
- ideally: interested, but critical audience

distance learning

- very limited interaction beyond multiple choice polls, chat questions
- full Zoom participants lists
- lectures go exactly as planned
- very uncritical audience: my wall!

two possibilities

1. my teaching is perfect, nothing to adjust
2. something's missing ...



Distance Learning: IfI Experience

experience IfI

- very little interaction
- feedback mostly when using polls / surveys in class, some questions in chat
- students are essentially names in Zoom participant lists
- **Zoom-teaching almost the same as video recording**

summary

- teaching in summer 2020: essentially video tutorials plus homework, exam grading
- works for basic programming courses, but not here



Adjustments

then and now

- summer 2020: used on-campus methods in remote setting
- now: we're better prepared, try "real" remote teaching methods

usage of class time

- goal: less prepared presentation, more interaction
- your participation:
 - questions
 - discussion of review questions
 - presentation of exercise solutions
- consequence: no video recordings of "live lectures"



Distance Learning in ESSS 20/21: Half a Plan!

weeks ≈ 1-3: introduction, motivation, formal model introduction

traditional lecture

- “live” presentation here in class
- discussion of review questions in (exercise) class

weeks ≈ 4-6: formal model application, main (un)decidability results

flipped classroom

- I provide materials (videos, lecture slides, lecture notes) beforehand, including “schedule”
- use “Wilke model” in class: meet in smaller groups to discuss questions

after ≈ week 6

review what worked with your feedback



Video Lectures

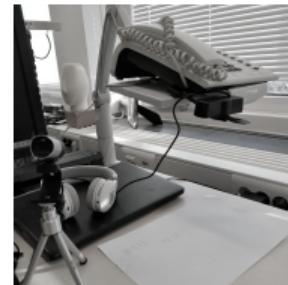


motivation

- non-interactive presentations do not need to be “live”
- technical content like proofs better presented in videos than on slides

video content

- slides with voiceover
- hand-written proofs on paper/whiteboard
- voice-over



alternative to videos

- slides from videos appear in lecture slides (gray background for distinction)
- detailed proofs contained in lecture notes



Workload?



Workload?

your perspective

- = regular lecture meetings (some only partial with split groups)
- = work on exercise tasks
- + work on review questions
- + watch video lectures

issue

- more work compared with traditional course?
- apples/oranges comparison: in on-campus iteration, you also study material on your own!
- distance learning can't just be "classroom relocated to Zoom"
- my hypothesis/goal: workload for **active participants** similar as in on-campus iteration

measure to limit workload for you

- we cover the same topics as in last iteration of the course (also 13 lecture sessions)
- lecture progress: comparable to last time



How can we make this work?

cameras

- experience: people are more focused in online meetings when cameras are on
- but: we're many people
- suggestion: turn your camera on at least during “discussion parts” of the lecture

Zoom

- use real names
- use “raise hand” function, talk if I don’t “see you”
- questions in chat: I try to see and answer them all, but usually with delay

general

your feedback is essential!



Distance Learning: Key Points

none of this works without your participation!

also on-campus: active students are more successful in this course

we're still learning!

let's experiment and try some stuff!

"I'm just here for a quick look"

if you just want to watch/listen to get a superficial overview of the topic: fine too!



Overview and Motivation

Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview



Computer Security

relevance

- topic gets (relatively) much media attention
- usual reporting: current breaches and attacks
- high-profile examples
 - RAMBleed, Meltdown, Spectre, Heartbleed
 - WPA/2 attack KRACK
 - Loss of Credit Card / Hotel Customer Data
 - malware (e.g., crypto miners in audio files)
 - concrete security issues in specific software
 - ...



WPA/2 KRACK

summary

- Wi-Fi Protected Access 2 (IEEE 802.11i): WLAN security
- designed to ensure privacy over WLAN
- widespread deployment

KRACK (October 2017)

- Mathy Vanhoef and Frank Piessens. [Key Reinstallation Attacks: ForcingNonceReuseinWPA2.](#) 2017 (presentation based on, images taken from this paper)
- crucial bug in the protocol
- allows attacker to read all messages
- crucial for this lecture: conceptual issue, not only bug in concrete implementation



Attack Outline

vulnerability

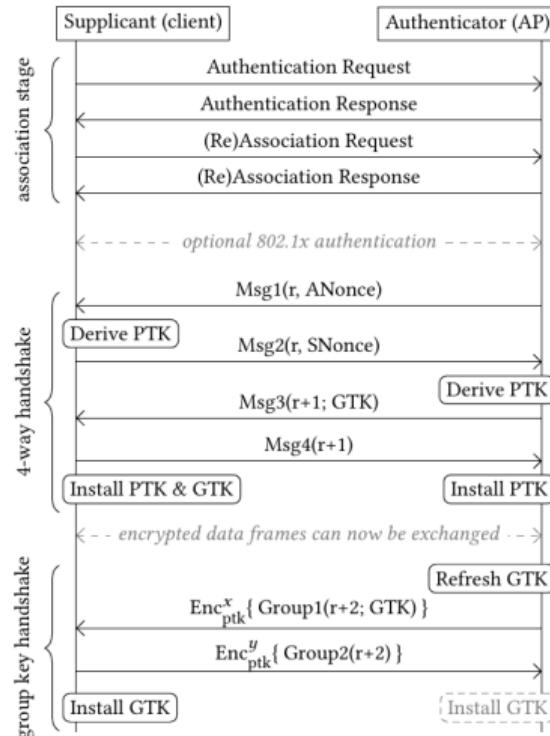
- WPA/2 uses a sub-protocol for encrypted communication
 - CCMP (Counter-Mode/CBC-MAC Protocol), based on AES
 - secure as long as “initialization vectors” (some random numbers) are not reused
- key to attack: force protocol to re-use initialization vectors
- then CCMP does not guarantee any security anymore

question

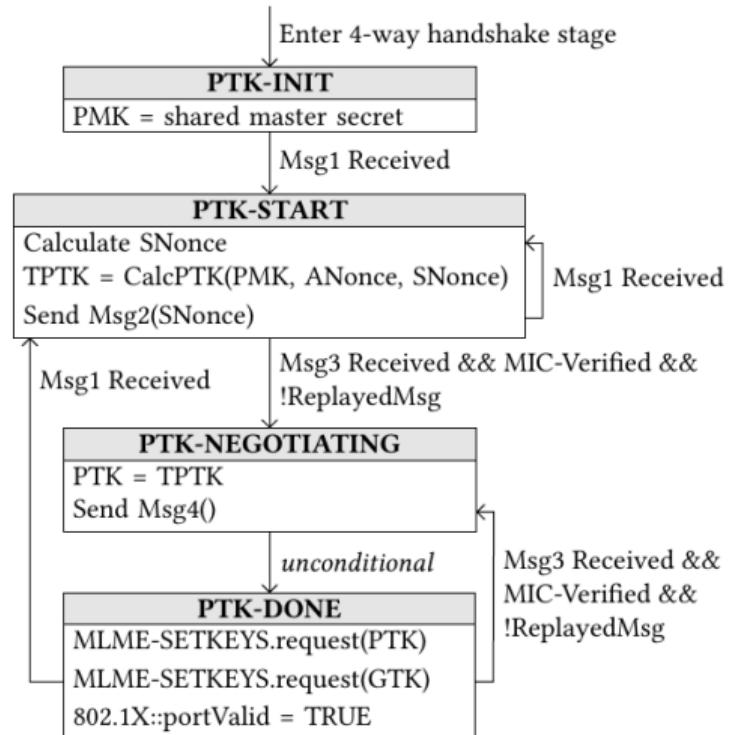
how does an attacker force re-use of initialization vectors?



Handshake Specification I



Handshake Specification II



Bug in Protocol

approach

- protocol allows retransmission/acceptance of message from handshake
- needed in case of message loss
 - can occur with unreliable network
 - or be forced by attacker
- effect: CCMP used with same “initialization vector”
- completely breaks WPA/2

worst case

- Linux systems (including Android): key reinstallation leads to “zero key”
- trivial to “crack”



Affected Systems

Implementation	Re. Msg3	Pt. EAPOL	Quick Pt.	Quick Ct.	4-way	Group
OS X 10.9.5	✓	✗	✗	✓	✓	✓
macOS Sierra 10.12	✓	✗	✗	✓	✓	✓
iOS 10.3.1 ^c	✗	N/A	N/A	N/A	✗	✓
wpa_supplicant v2.3	✓	✓	✓	✓	✓	✓
wpa_supplicant v2.4-5	✓	✓	✓	✓ ^a	✓ ^a	✓
wpa_supplicant v2.6	✓	✓	✓	✓ ^b	✓ ^b	✓
Android 6.0.1	✓	✗	✓	✓ ^a	✓ ^a	✓
OpenBSD 6.1 (rum)	✓	✗	✗	✗	✗	✓
OpenBSD 6.1 (iwn)	✓	✗	✗	✓	✓	✓
Windows 7 ^c	✗	N/A	N/A	N/A	✗	✓
Windows 10 ^c	✗	N/A	N/A	N/A	✗	✓
MediaTek	✓	✓	✓	✓	✓	✓

^a Due to a bug, an all-zero TK will be installed, see Section 6.3.

^b Only the group key is reinstalled in the 4-way handshake.

^c Certain tests are irrelevant (not applicable) because the implementation does not accept retransmissions of message 3.



But there was a Proof?

The 4-way handshake was mathematically proven as secure. How is your attack possible?

Our attacks do not violate the security properties proven in formal analysis of the 4-way handshake. In particular, these proofs state that the negotiated encryption key remains private, and that the identity of both the client and Access Point (AP) is confirmed. Our attacks do not leak the encryption key. Additionally, although normal data frames can be forged if TKIP or GCMP is used, an attacker cannot forge handshake messages and hence cannot impersonate the client or AP during handshakes. Therefore, the properties that were proven in formal analysis of the 4-way handshake remain true. However, the problem is that the proofs do not model key installation. Put differently, the formal models did not define when a negotiated key should be installed. In practice, this means the same key can be installed multiple times, thereby resetting nonces and replay counters used by the encryption protocol (e.g. by WPA-TKIP or AES-CCMP).



Lessons Learned

take-aways

- bugs appear in standard, widely used protocols
- composition of security protocols not trivial
 - approaches see [Cano1], [BPWo4], [Kuso6], [Lin17]
- formal methods
 - methods only verify specified (security) properties
 - specifying security properties itself is highly nontrivial
 - approaches see [Sch12] (and references therein), [NSC18]
- **definitions of security** critical
 - no “single security definition”
 - consequence: security specification should be part of an analysis algorithm’s input



Engineering Secure Software Systems

Wikipedia

Engineering is the application of **mathematics**, as well as **scientific**, economic, social, and **practical** knowledge, to invent, innovate, **design**, build, maintain, **research**, and improve structures, machines, **tools**, **systems**, components, materials, processes, solutions, and organizations.

lecture focus: topics that

- are parts of a unified theory
- tell us how to **build** secure (parts of) systems and **tools** to analyse them
- analyse security of one **aspect** (level of abstraction)



Engineering: We love Tools!

software engineering

- IDEs
 - refactoring capabilities
 - type-aware code completion
 - ...
- static analysis
 - checkstyle
 - PMD
 - ...
- dynamic analysis
 - monitoring
 - trace analysis
- ...

security engineering

?

this lecture

- what do we **want** tools to do?
- what can—and cannot—be done?

theory lecture!

- study theory behind tools
- algorithms, hardness and impossibility results
- practical aspect: ProVerif



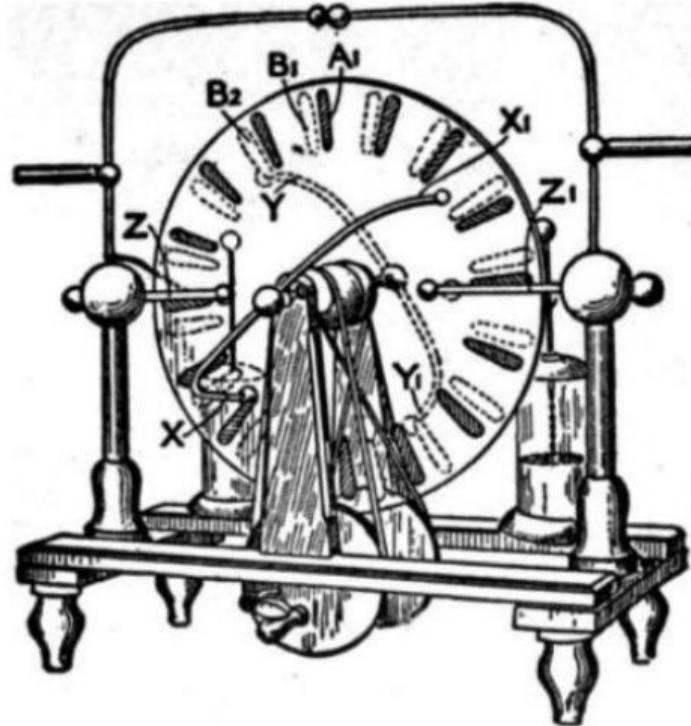
Ideal Situation



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!



Key Areas in Lecture

key topics

1. cryptographic protocols
 - formal model
 - definitions of security
 - automatic analysis
 - real examples: voting protocols, (Bitcoin,) ...
2. information-flow security
 - formal model
 - transitive and intransitive policies
 - automatic analysis
 - real example: Meltdown
3. language-based security
 - if we get around to it!

theory lecture

security: success story for formal methods

- abstract systems view
 1. Alice-and-Bob protocols
 2. system as state diagram
 3. “toy examples” for languages
- results
 - security proofs (to a point)
 - algorithms, (im-)possibility results
 - complexity results
- techniques
 - formal models and proofs



Caveat: Theory Lecture

theory lecture

- formal models
- formal definitions
- formal results and proofs.

compromise approach

- theory in detail
- examples: a bit of handwaving
- additional details are in the notes!

motivation from practice

- need theory that can express real systems
- want to express real systems in the theory
- formal models of real systems tend to be large

issues

- there is no “correct” level of formality
 - lecture feedback goes both ways, I promise to err on both sides!
- if you want more formal details / more intuitive examples, let me know!



Caveat: Theory Lecture

prerequisites

- foundations of computer science theory (TGI)
 - complexity: nondeterministic algorithms, reductions, NP-completeness
 - decidability: reductions, undecidability
- logic for computer science (LogInf)
 - terms and signatures
 - Horn clauses, first-order logic
- data structures and algorithms (ADS)
 - data structures, trees, (directed acyclic) graphs



Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview



Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview





what are crypto protocols?

- application of crypto primitives to provide “secure” services
- examples: Diffie-Hellman, Internet Key Exchange, TLS, ...

Needham-Schroeder protocol

$$\begin{aligned} A \rightarrow B & \quad \text{enc}_{k_B}^a(A, N_A) \\ B \rightarrow A & \quad \text{enc}_{k_A}^a(N_A, N_B) \\ A \rightarrow B & \quad \text{enc}_{k_B}^a(N_B) \end{aligned}$$



Analysis of Protocols

goal: automatic analysis

analyse security properties of protocols (semi-)automatically

feasibility

- protocols are “small,” so complete analysis possible
- successful application of formal methods
 - model checking
 - (interactive) theorem proving
 - type systems

question

how far can we go?

difficulty

protocols “small,” but attacker strategies unrestricted



Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview



Information Flow Security

motivation

- components with different security levels on one system
- `top secret > classified > public`
- requirement: no information about “higher level” data can be derived from “lower level” data

questions

- what does “secure” mean?
- how can we check / guarantee security?

points of view

local system as finite automaton

global architecture view





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?



Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview



Learning Goals

protocols

- specify protocols and security properties
- evaluate security
- automatic analysis and its limits
- best practice for protocol design
- transfer of abstract results to real protocols
- model and analyse protocols with ProVerif

information flow

- know security notions and relationships
- choose matching security notion for application scenario
- evaluate systems, know basic algorithms

language based security

- formal and “practical” security



Overview

Overview and Motivation

Introduction

Topic Overview: Three Areas in a
Nutshell

Crypto Protocols

Information Flow

Learning Goals

Lecture Overview



Lecture Plan (based on last year)

categories

- introduction
- examples
- formal model
- algorithms and proofs
- tool application

lecture 1 intro, overview, crypto primitives	November 3, 2020	
lecture 3 protocol model, attack definition	November 17, 2020	
lecture 5 RT algorithm and proof	December 1, 2020	
lecture 7 logic modeling, ProVerif basics, equational theories	December 15, 2020	
lecture 9 ProVerif: events and incompleteness	January 12, 2021	
lecture 11 infoflow basics, transitive infoflow	January 26, 2021	
lecture 13 unwindings	February 9, 2021	
lecture 2 examples, protocol model, adversary capabilities	November 10, 2020	
lecture 4 RT algorithm and proof	November 24, 2020	
lecture 6 RT proof II, negative results	December 8, 2020	
lecture 8 ProVerif: secrecy and beyond	January 5, 2021	
lecture 10 Voting Protocols	January 19, 2021	
lecture 12 transitive and intransitive infoflow	February 2, 2021	

skipped in winter 19/20
abstractions, BitCoin



Lecture Plan (update 1)

categories

- introduction
- examples
- formal model
- algorithms and proofs
- tool application

lecture 1 intro, overview, crypto primitives	November 3, 2020
lecture 3 protocol model, video: dy computation	November 17, 2020
lecture 5 RT algorithm and parsing lemma	December 1, 2020
lecture 7 negative results, logic modeling	December 15, 2020
lecture 9 ProVerif: secrecy and beyond	January 12, 2021
lecture 11 infoflow basics, transitive infoflow	January 26, 2021
lecture 13 intransitive infoflow	February 9, 2021
lecture 2 examples, protocol model, adversary capabilities	November 10, 2020
lecture 4 formal attack definition	November 24, 2020
lecture 6 RT algorithm proof, NP-hardness	December 8, 2020
lecture 8 ProVerif basics, equational theories	January 5, 2021
lecture 10 ProVerif: events and incompleteness	January 19, 2021
lecture 12 transitive infoflow and unwindings	February 2, 2021

skipped in winter 20/21
voting protocols



Part I: Crypto Protocols

Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



What are Cryptographic Protocols?

protocols

- fix steps of communication
- sequence of messages
- examples

TCP “low-level” communication

HTTP website delivery

SMTP email transport



assumption for application

- parties are honest
- reliable channels (TCP protects against non-malicious errors)

internet: both assumptions unrealistic!



Well-Known Crypto Protocols

examples

SSL/TLS encryption, authentication of web communication

IPSec virtual private networks

IKE internet key exchange

SSH secure (remote) shell

BitCoin digital currency

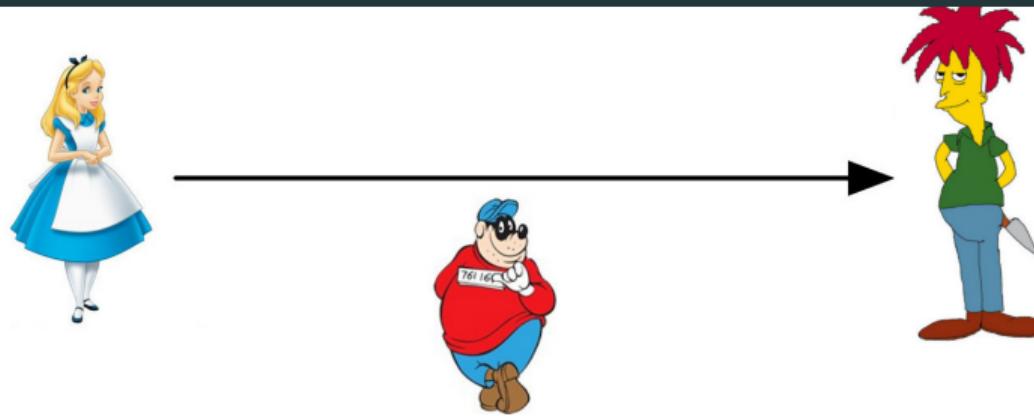


security

- Are these protocols “secure”?
- What does “secure” even mean?
recall: definitions central!



Communication over the Internet



ideal assumption

secure channel between Alice and Bob

crypto assumption

Adversary completely controls network! Can read/manipulate/drop all messages

reality

which assumption is realistic? which one is useful?



Crypto Protocols: Scenarios and Goals

protocol security goals

- privacy (secrecy)
- authentication
- message exchange
- online banking
- contract signing
- online shopping
- key exchange
- electronic elections
- crypto currencies
- secure remote access

...

clearly, there are overlaps

key questions

- what are the **precise** security requirements here?
- how do we formalize these?



Crypto Protocols: Special Issues

quote

Security protocols are three line programs that people still manage to get wrong.

Roger Needham

reasons

- protocols must be “successful” against adversary
- bugs hard to find: how do we “debug” security holes?

bug in protocol $\hat{=}$ possible attack!

consider

- security against all **possible** attacks needed
- security against all **known** attacks is not enough!





- logical errors

C++

- buffer overruns, length checks ...



- compiler errors, library bugs ...



- operation system bugs



- hardware design error, short-term failures



“complete analysis:” obviously undecidable!

Distinction: In this Lecture

abstraction level we consider

- protocols on “Alice-and-Bob” level
- finding “logical attacks” on this level

motivation

- reality: attacks not on RSA, but on protocol/application
 - RSA security: discussed in crypto lecture!
- implementation errors “similar” to usual software development
 - buffer overflows
 - SQL injection
 - “normal” bugs

as always: exceptions!

analysis on logical level does not guarantee system security!



Distinction: Not in this lecture



Adversary: Two Aspects



dishonest components

network attacker completely controls the network

parties parties do not follow protocol

no clear separation

- similar consequences: messages may be fake
- possible reasons:
 1. network delivers wrong message
 2. Bob acts dishonestly

treat both aspects uniformly

assumption: there is a single adversary \mathcal{A} controlling the network
and all dishonest parties (who all work together)

security tradition

“maximally pessimistic assumptions”



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



Tools Against Powerful Adversaries

assumption

- attacker controls network
- dishonest parties

defence?

use cryptography! encryption, signatures, hash functions ...

caveats

- encryption alone does not ensure security
 - need shared keys, PKI, ...
- cryptographic infrastructure
- now: brief discussion of cryptographic primitives



Crypto in this Lecture: A Black Box

important

- no prior knowledge about cryptography assumed
- see crypto lecture by Prof. Wilke, or [KW11], or vast literature

cryptography aspects

- encryption (symmetric and asymmetric)
- signatures (symmetric (MAC) und asymmetric)
- hash functions
- random numbers

application

- “abstract” view: no concrete algorithms!
- security properties: “idealized”!



Cryptographic Primitives: Encryption



two cases

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

- encrypt: $X \times K \rightarrow Y$ $\text{dec}_k^s(\text{enc}_k^s(x)) = x$
- decrypt: $Y \times K \rightarrow X$

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

- encrypt: $X \times K \rightarrow Y$
- decrypt: $Y \times \hat{K} \rightarrow X$ $\text{dec}_{\hat{k}_A}^a(\text{enc}_{k_A}^a(x)) = x$
- mapping: $\hat{\cdot}: K \rightarrow \hat{K}$

security property in both cases

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



Cryptographic Primitives: Signatures



two cases

symmetric (MAC) (CMAC, ...) single key k_{AB} for “signing” and verification

- sign: $mac: X \times K \rightarrow T$ $test(x, k, mac_k(x)) = \text{ok}$
- verify: $test: X \times K \times T \rightarrow \{\text{ok}, \text{error}\}$

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

- sign: $sig: X \times \hat{K} \rightarrow S$
- verify: $test: X \times K \times S \rightarrow \{\text{ok}, \text{error}\}$ $test(x, k, sig_{\hat{k}}(x)) = \text{ok}$
- mapping: $\hat{\cdot}: K \rightarrow \hat{K}$

security property in both cases

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





perfect hash function

computation $\text{hash}: X \rightarrow T$

collision resistance if $x \neq y$, then $\text{hash}(x) \neq \text{hash}(y)$

information about message

- naive approach: $\text{hash}(x)$ gives no information about x
- problem?
- attacker capabilities?: see later (equational theories)





ideal properties

- random generators always return fresh values
- random bitstrings cannot be guessed

consequences

- session ids are perfectly random and unpredictable
- randomisation for encryption and other primitives is always perfect
- key generators are perfect



Exercise

Task (WhatsApp Authentication)

The instant messaging 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.



Summary: Public and Secret Keys

Priv.-Doz. Dr. Henning Schnoor



wiss. Mitarbeiter

Christian-Albrechts-Platz 4, R.1215 (CAP 4)

Phone: ☎ +49 431 880-4467

Telefax: ☎ +49 431 880-7617

✉ henning.schnoor@email.uni-kiel.de

PGP key: ☎ [henning-schnoor-pgp-key.asc](#)

with public key k_{HS} , you can ...

- send encrypted emails to me

$$x \rightarrow \text{enc}_{k_{HS}}^a(x)$$

- verify whether I signed a message

$$\text{test } \text{sig}_{k_{HS}}(x)$$

with secret key \hat{k}_{HS} , I can ...

- decrypt mails encrypted with my public key

$$\text{enc}_{k_{HS}}^a(x) \rightarrow x$$

- sign messages that will successfully verify against my public key

$$x \rightarrow \text{sig}_{k_{HS}}(x)$$



Assumptions too strong?

impossible with adversary-controlled network

- reply after $\leq t$ seconds
- reliable emergency call system
- delivery guarantee for messages
- ...

in general

“liveness” properties cannot be guaranteed
when network is completely unreliable

protocol design futile

If all others “maximally dishonest:” communication not reasonable

consequence: assumptions (depend on scenario), examples:

- at least Alice and Bob are honest
- existence of a trusted third party (TTP)
- Alice and Bob share secret key
- availability of PKI (public-key infrastructure)
- ...



Hopeless Situations?

cryptography can only help you so far ...

political electronic elections in Germany

Der Zweite Senat hat entschieden, dass der Einsatz elektronischer Wahlgeräte voraussetzt, dass die **wesentlichen Schritte der Wahlhandlung und der Ergebnisermittlung vom Bürger zuverlässig und ohne besondere Sachkenntnis überprüft werden können**. Dies ergibt sich aus dem Grundsatz der Öffentlichkeit der Wahl (Art. 38 in Verbindung mit Art. 20 Abs. 1 und Abs. 2 GG), der gebietet, dass alle wesentlichen Schritte der Wahl öffentlicher Überprüfbarkeit unterliegen, soweit nicht andere verfassungsrechtliche Belange eine Ausnahme rechtfertigen.



Example: Authentication

goal

Bob expects “authenticated” message from Alice

problems

- attacker can always send message in Alice’s name!
- Bob needs way to check authenticity of message

cannot require that message arrives (liveness)

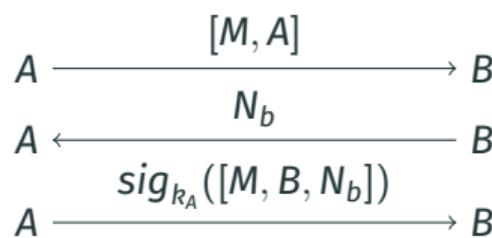
require only: if Bob “accepts,” then message is from Alice.

need infrastructure: Alice “can do” something the attacker “can’t”

- Alice has private key, Bob knows public key
- Alice and Bob share private secret
- Alice can authenticate herself using a certificate
- ...

A Secure Protocol: Simple Authentication

protocol



too complicated?

- why three messages?
- why is N_b needed?
- why must B be signed?

Bob's guarantees?

What can Bob be sure of after the protocol has successfully completed?



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 $\text{sig}_{k_A}([M, B, N_B])$.

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?



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



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

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

secure protocol?

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



The Needham-Schroeder Protocol



goal

authentication and key exchange

protocol

$A \rightarrow B \quad \text{enc}_{k_B}^a(A, N_A)$

$B \rightarrow A \quad \text{enc}_{k_A}^a(N_A, N_B)$

$A \rightarrow B \quad \text{enc}_{k_B}^a(N_B)$

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]



Attack on Needham-Schroeder



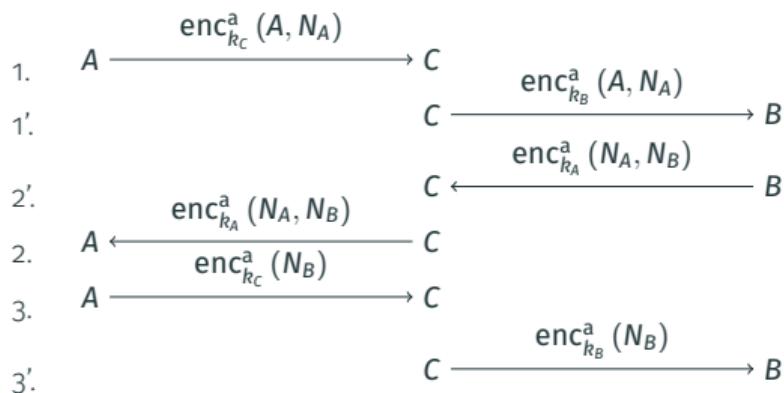
protocol

1. $A \rightarrow B \text{ } C \quad \text{enc}_{k_B}^a (A, N_A)$
2. $B \text{ } C \rightarrow A \quad \text{enc}_{k_A}^a (N_A, N_B \text{ } N_C)$
3. $A \rightarrow B \text{ } C \quad \text{enc}_{k_B}^a (N_B \text{ } N_C)$

situation

- Alice starts protocol as initiator with C (attacker)
- Bob starts protocol as responder with Alice
- adjust protocol for this situation

attack (Charlie controlled by \mathcal{A})



consequence

- who is attacked?
- Bob “thinks” only Alice knows N_A and N_B
- C knows N_A and N_B
- what about Alice’s point of view?
- suggestions to fix protocol?



The Needham-Schroeder-Lowe Protocol

protocol

$A \rightarrow B \quad \text{enc}_{k_B}^a(A, N_A)$

$B \rightarrow A \quad \text{enc}_{k_A}^a(N_A, N_B, B)$

$A \rightarrow B \quad \text{enc}_{k_B}^a(N_B)$

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

intuition

- attack “mixes” messages from different protocol sessions
- consequence: B “talks to” C instead of A
- change: A realizes that message does not come from C



Attack on Needham-Schroeder-Lowe?

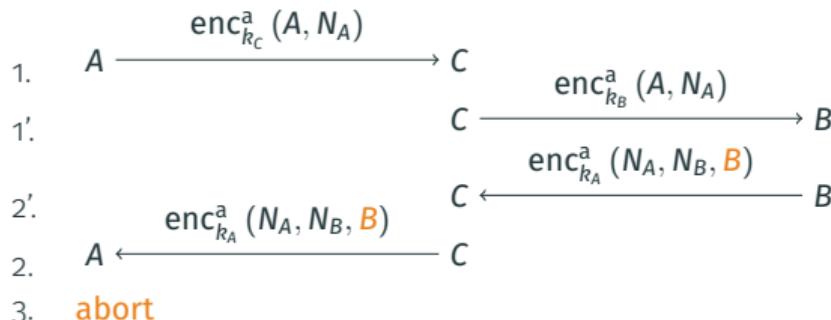
protocol

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

consequence

- Alice “talks to C” receives message with “sender” B
- Alice aborts
- good practice: sender and receiver in messages

attack attempt



Exercise

Task (Fixing Broken Authentication Protocols)

Consider the two authentication protocols presented in the exercise class:

a)

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

b)

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

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.



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



Example: Woo-Lam Authentication Protocol

prerequisites

k_{AS}, k_{BS} : symmetric keys shared between Alice (Bob) and Server

protocol

1. $A \rightarrow B \quad A$
2. $B \rightarrow A \quad N_B$
3. $A \rightarrow B \quad \text{enc}_{k_{AS}}^s(N_B)$
4. $B \rightarrow S \quad \text{enc}_{k_{BS}}^s([A, \text{enc}_{k_{AS}}^s(N_B)])$
5. $S \rightarrow B \quad \text{enc}_{k_{BS}}^s(N_B)$

idea

- only Alice can encrypt N_B with k_{AS}
- server can check correctness

issues?

- server is “decryption oracle”
- Alice does not “know” that she “talks to Bob”

reference

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>



Woo-Lam Protocol is insecure



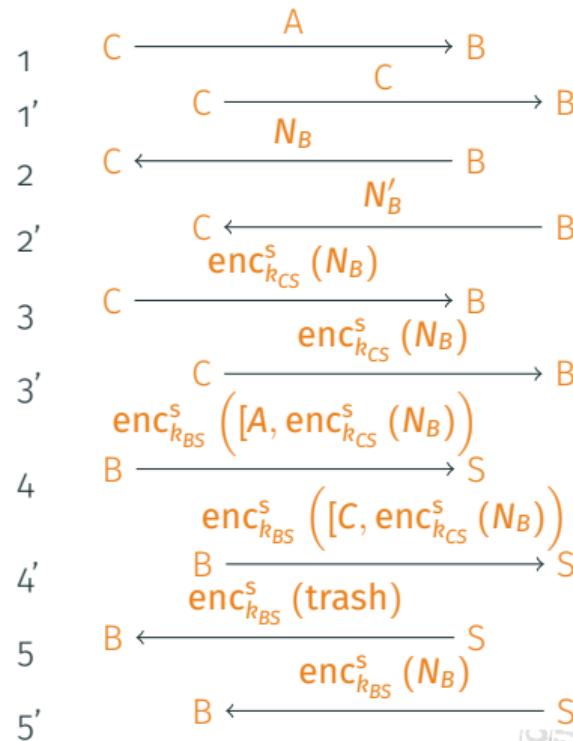
protocol

1. $A \rightarrow B \quad A$
2. $B \rightarrow A \quad N_B$
3. $A \rightarrow B \quad \text{enc}_{k_{AS}}^s(N_B)$
4. $B \rightarrow S \quad \text{enc}_{k_{BS}}^s([A, \text{enc}_{k_{AS}}^s(N_B)])$
5. $S \rightarrow B \quad \text{enc}_{k_{BS}}^s(N_B)$

analysis

- trash: result of decrypting $\text{enc}_{k_{CS}}^s(N_B)$ with K_{AS}
- B believes A participated in protocol run
- assumptions?

attack: C controlled by \mathcal{A} , B honest



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



Finding Attacks

seen up to now: manual analysis

issues found by

- construction of message sequence: reachability of “bad state”
- argument that the attacker “knows” each message she sends: attacker gains sufficient knowledge to trigger required events

goal: automatic analysis

want algorithm that comes up with attack, or “proves” that there is no attack

required

formal model in which we can express protocols, security, and attacks



Model Requirement: Express Needham Schroeder

minimal requirement for formal model

must be able to formalize Needham-Schroeder(-Lowe) protocol, attack, security.

attacker actions

- | | |
|--|--------------------|
| 1. C "talks to Bob in Alice's name" | steps 1', 2', 3' |
| 2. C makes Alice accept N_B instead of N_C | step 2 |
| 3. C exchanges data between sessions | all steps |
| 4. C lets Alice and Bob wait | steps 1', 2', 2, 3 |

consequences for model

1. untrusted message delivery
2. Alice's protocol specification must not mention N_C
3. attacker can send arbitrary terms, limited only by cryptography
4. attacker controls scheduling





untrusted message delivery

messages delivered by network without meta-information

Alice's protocol specification must not mention N_c

expected terms cannot be hard-coded, model steps as receive/send-rules with variables instead

attacker can send arbitrary terms, limited only by cryptography

adversary controls network, uses “message construction” rules precisely defined using so-called Dolev-Yao closure

attacker controls scheduling

scheduling (execution order) explicitly done by adversary



Roadmap: Formal Model

features

1. untrusted message delivery
2. Alice's protocol specification must not mention N_C
3. attacker can send arbitrary terms, limited only by cryptography
4. attacker controls scheduling

components of formal model

messages

message construction, delivery, parsing

protocol specifications

sessions, scheduling

protocol security: no combination of
adversary actions breaks protocol goal

formal terms

Dolev-Yao closure,

receive/send actions, substitutions, matching

protocol instance, protocol

execution order

protocol runs, (successful) attacks

Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy





implemented protocol

- messages are bitstrings
- constructed by crypto algorithms
- attacker: arbitrary probabilistic polynomial-time algorithm

formal model

- messages are terms
- algorithms represented by function symbols
- attacker: nondeterministic choice of messages

why?

advantages of term model?





definition: terms

\mathcal{T} : smallest set with

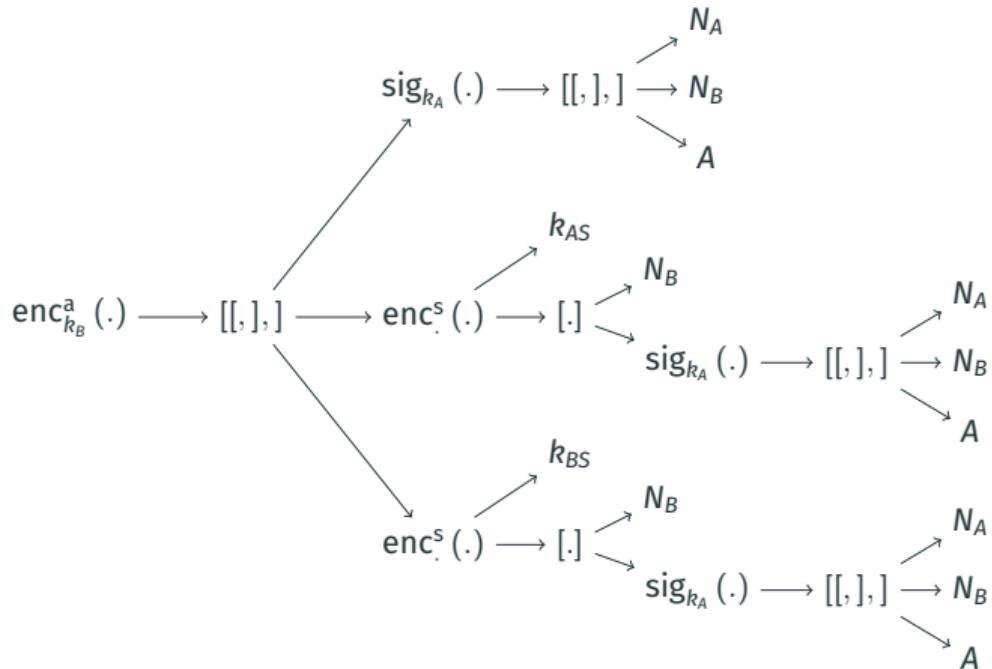
- $\{\epsilon\} \cup \mathcal{C} \cup \mathcal{V} \cup \text{IDs} \subseteq \mathcal{T}$, empty message, constants, variables, names
- for all $i \in \mathbb{N}$, all $a \in \text{IDs}$: $N_i, k_a, \hat{k}_a \in \mathcal{T}$, random values, keys
- if $t_1, t_2 \in \mathcal{T}$, then $[t_1, t_2] \in \mathcal{T}$, pairs/sequences
- if $t, t_k \in \mathcal{T}$, then $\text{enc}_{t_k}^s(t) \in \mathcal{T}$, symm. encryption
- if $t \in \mathcal{T}$, $a \in \text{IDs}$, then $\text{enc}_{k_a}^a(t) \in \mathcal{T}$, asymm. encryption
- if $t, t_k \in \mathcal{T}$, then $\text{MAC}_{t_k}(t) \in \mathcal{T}$, symm. signature (MAC)
- if $t \in \mathcal{T}$, $a \in \text{IDs}$, then $\text{sig}_{k_a}(t) \in \mathcal{T}$, asymm. signature
- if $t \in \mathcal{T}$, then $\text{hash}(t) \in \mathcal{T}$. hash function

messages

term without variable: **ground term**, message.



Terms: Tree Representation



remarks

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



Definition: Subterms



definition: subterms

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

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

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



Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



Model Requirement: Express Cryptographic Limitations

situation in protocol run: \mathcal{A} knows messages in set S

- own keys
- keys of dishonest parties
- “common knowledge” terms
`init, request, ...`
- messages sent by participants so far
- ...

cryptographic operations to cover

- asymmetric encryption
- symmetric encryption
- decryption (both cases)
- signature / MAC
- apply hash function
- ...

always possible

only with key

only with key

only with key

always possible

question

which messages can \mathcal{A} send?

formally

define set $DY(S)$ of messages that \mathcal{A} can derive from S



Key Concept: Dolev-Yao Closure

reference

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

simple attacker modeling

- standard model, many extensions
- consider primitives in isolation
- only derivations, no indistinguishability
(see later)
- actual cryptography abstracted away

too simple?

- assume “perfect cryptography”
- practice: do RSA, AES, ElGamal satisfy this?
- abstraction step must be justified!

abstraction soundness

nontrivial topic, subtle issues – (possibly) later in the lecture!





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

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

note

model allows composed keys for symmetric cryptosystems

Dolev-Yao Closure Examples I

situation: PKI, shared keys, look at Charlie

$$S = \left\{ \hat{k}_C, k_A, k_B, k_C, k_{AC}, k_{BC}, N_C^1, N_C^2, \text{enc}_{k_{BC}}^s \left(\text{enc}_{k_C}^a \left(\text{enc}_{k_{AB}}^s (N_A) \right) \right) \right\}$$

derivable?

- $\text{sig}_{k_C} \left(\text{enc}_{k_{AB}}^s (N_A) \right)$? yes
- $\text{sig}_{k_C} (N_A)$? no
- $\text{sig}_{k_A} \left(\text{enc}_{k_{AB}}^s (N_A) \right)$? no



Dolev-Yao Closure Examples II

initial adversary knowledge

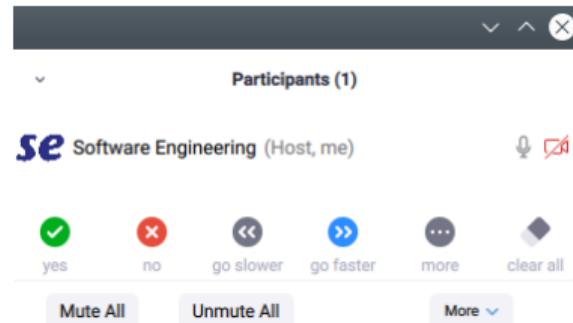
$$I = \{A, B, \hat{k}_I, k_{AI}, k_{BI}, 0, 1, \text{yes}, \text{no}\},$$

knowledge grows with each message

messages

\mathcal{A} receives	goal	derivable?
$\text{enc}_{k_A}^a(\text{secret})$	secret	✗
$\text{enc}_{k_I}^a(\text{secret})$	secret	✓
$\text{enc}_{k_{AB}}^s(\text{yes})$	yes	✓
$\text{enc}_{k_{AB}}^s(N_A)$	N_A	✗
$\text{enc}_{k_{AI}}^s(k_{AB})$	N_A	✓
$\text{enc}_{k_A}^s([0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1])$	$[0, \dots, 1]$	✓
$\text{enc}_{[0,1,1,0,0,1,1,\dots,0,1,1]}^s(N_B)$	N_B	✓

can adversary derive terms?



consequence

- arbitrarily long bit sequences always “known”
- do not model: adversary knows that **this message** contains “yes”
- \rightsquigarrow generalization later

Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy





Computing the Dolev-Yao Closure

<https://cloud.rz.uni-kiel.de/index.php/s/LXBHnCergZ35sfF>

video content

- characterization of $DY(S)$ with *derivation rules*
- properties of “minimal derivations”
- a fixpoint algorithm for “computing” $DY(S)$

study

- watch video—feedback welcome!
- video slides contained in slide set (gray background), additional material in lecture notes
- next week: discussion of content (in small groups), bring questions!



Goal: Compute Dolev-Yao Closure

Dolev-Yao

- proofs of insecurity (security): argue that adversary can (not) send message m
- need formal criterion of messages that adversary can send
- $\text{DY}(S)$: set of messages the adversary can derive from S

long-term goal: automatic security analysis

need algorithm for $\text{DY}(S)$

obstacle to computation

- $\text{DY}(S)$ is infinite: $\epsilon, [\epsilon, \epsilon], [\epsilon, [\epsilon, \epsilon]], \dots$
- algorithm cannot “write down” $\text{DY}(S)$

way out

- we do not need to enumerate $\text{DY}(S)$
- suffices to algorithmically answer question
can adversary send m ?



decision problem

Problem: DERIVE

Input: set of terms S , term m

Question: is $m \in \text{DY}(S)$?

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

Technique: Proof Overview

steps

- characterization of Dolev-Yao Closure with **derivation rules**
- deciding whether $m \in \text{DY}(S)$ is deciding whether there is a derivation of m from S
- issue: infinite search space of derivations
- solution:
 - if there is a derivation of m from S , then there is a shortest one
 - a shortest derivation contains no unnecessary steps
 - this restricts the search space

simplification

- to simplify case distinctions: only encryption, pairing, nonces, constants in this proof
- arguments suffice to also cover signatures, MACs and hash functions, ...
- see exercise task for generalization



rules

for a message m , rule $L_d(m)/L_c(m)$ describes how m can be decomposed/composed
this potentially needs prerequisites:

- composing $\text{sig}_{k_B}(m)$ needs m and \hat{k}_B
- decomposing $\text{enc}_{k_{AB}}^s(m)$ needs k_{AB}

a rule consists of a set R of required and and a set O of obtained terms, written $R \rightarrow O$. In the specific rules, we omit set brackets.

composition rules

$$\begin{array}{ll} L_c([a, b]) & a, b \rightarrow [a, b] \\ L_c(\text{enc}_{k_A}^a(m)) & m \rightarrow \text{enc}_{k_A}^a(m) \\ L_c(\text{enc}_{t_k}^s(m)) & m, t_k \rightarrow \text{enc}_{t_k}^s(m) \end{array}$$

decomposition rules

$$\begin{array}{ll} L_d([a, b]) & [a, b] \rightarrow a, b \\ L_d(\text{enc}_{k_A}^a(m)) & \text{enc}_{k_A}^a(m), \hat{k}_A \rightarrow m \\ L_d(\text{enc}_{t_k}^s(m)) & \text{enc}_{t_k}^s(m), t_k \rightarrow m \end{array}$$

Application of Rules: Derivations

definition

derivation: sequence $S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} \dots S_{n-1} \rightarrow_{L_{n-1}} S_n$, such that, for all $i \in \{0, \dots, n-1\}$,

- L_i is a rule of the form $S \rightarrow S'$
- $S \subseteq S_i$
- $S_{i+1} = S_i \cup S'$

intuition

S_{i+1} obtained from S_i with L_i

Characterization: Dolev-Yao Closure Captured by Derivation Rules

lemma & definition

If $m \in \text{DY}(S)$ where $\text{IDs} \cup \{k_a \mid a \in \text{IDs}\} \cup \{\epsilon\} \subseteq S$, then there is a **derivation of m from S** :

$$S = S_0 \rightarrow_{L_0} S_1 \rightarrow_{L_1} \dots S_{n-1} \rightarrow_{L_{n-1}} S_n$$

with $m \in S_n$. We call n the **length** of the derivation.

proof

see exercise

definition

For $m \in \text{DY}(S)$, let $D_S(m)$ be a (fixed) **shortest** derivation of m from S . We write $L \in D_S(m)$, if L is a rule applied in $D_S(m)$.

Exercise

Task (DY closure and derivations)

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

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

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

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

Derivation Rules Property: One-Step Effects Only

composition

$$\begin{array}{ll} L_c([a, b]) & a, b \rightarrow [a, b] \\ L_c(\text{enc}_{k_A}^a(m)) & m \rightarrow \text{enc}_{k_A}^a(m) \\ L_c(\text{enc}_{t_k}^s(m)) & m, t_k \rightarrow \text{enc}_{t_k}^s(m) \end{array}$$

decomposition

$$\begin{array}{ll} L_d([a, b]) & [a, b] \rightarrow a, b \\ L_d(\text{enc}_{k_A}^a(m)) & \text{enc}_{k_A}^a(m), \hat{k}_A \rightarrow m \\ L_d(\text{enc}_{t_k}^s(m)) & \text{enc}_{t_k}^s(m), t_k \rightarrow m \end{array}$$

notation

$t \in \mathcal{T}$, then $\text{Sub}^1(t)$ set of direct subterms of t

- $\text{Sub}^1([t_1, t_2]) = \{t_1, t_2\}$
- $\text{Sub}^1(\text{enc}_k^s(t)) = \{k, t\}$
- $\text{Sub}^1(\text{hash}(t)) = \{t\}$
- ...

direct successors in tree representation

observation

rules only work on $\text{Sub}^1(\cdot)$ -level

- **composition rule** $L_c(m)$ has all terms from $\text{Sub}^1(m)$ as prerequisites
- **decomposition rules** $L_d(m)$ obtains only terms from $\text{Sub}^1(m)$



lemma

$D_S(m)$ shortest derivation of m from S , then:

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

relevance

to derive m from S , we only need

1. decompositions of subterms from S
2. compositions of subterms of S or subterms of m

let's prove this!

written proof also contained in lecture notes

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?



Input: set $S \neq \emptyset$ of messages, message m

$S_{old} = \emptyset$

while $S_{old} \neq S$ do

$S_{old} = S$

if ex. rule $S \rightarrow_L S \cup \{t\}$, $t \in \text{Sub}(S \cup \{m\}) \setminus S$ then

$S = S \cup \{t\}$

end if

end while

if $m \in S$ then

accept

end if

reject

algorithm uses previous results

- uses result on steps appearing in minimal derivations
- fixpoint algorithm: expands set S until fix point reached
- terminates in polynomial time since there are only polynomially many choices for t

covered in exercise

- algorithm correctness
- cannot “decompose first, compose later”

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).

Video Lecture: Feedback wanted



questions

- audio/video quality?
- proof presentation as screenshots, or “live writing?”
- better as video or “live Zoom session?”
- any suggestions?

feedback crucial

- your perspective very different from mine!
- constructive criticism always welcome
- review after week 6!

remember

- we're all still learning this
- new tools, concepts
- big playground :-)

Plan for Review Sessions

purpose, timing

- used after self-study material (videos)
- purpose: discussions / questions about content (usually proofs)
 - mainly: your questions
 - some: review questions
 - **no prepared material**, that's the point!
- length/time: full or partial next session
 - synchronize schedule with last course iteration

„Wilke model“: meet in smaller groups

- 2-3 groups, depending on number of participants (OLAT registration)
- groups for strong theory background / more basic theory knowledge
- please choose “fitting group,” otherwise discussed questions might not match your needs

Overview

Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



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 $r \rightarrow s$.

example from Needham-Schroeder

Bob's rule: $\text{enc}_{k_B}^a(A, x) \rightarrow \text{enc}_{k_A}^a(x, N_B)$

- 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)

$A \rightarrow B \quad \text{enc}_{k_B}^a(A, N_A)$

$B \rightarrow A \quad \text{enc}_{k_A}^a(N_A, N_B)$

$A \rightarrow B \quad \text{enc}_{k_B}^a(N_B)$

recall

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



definition: substitutions

- **substitution**: function $\sigma: \mathcal{V} \rightarrow \mathcal{T}$ 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 $x \in \mathcal{V}$
- $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$

examples

~~~ lecture notes

# Receiving and Parsing a Message

central step: react to incoming message

- state:  $\sigma(x) \neq x$  for some  $x$ , next r/s action is  $r \rightarrow s$
- incoming message:  $m$
- reaction: updated substitution  $\sigma'$ , reply message

Alice

- substitution:  $\sigma(x) = N_B^1, \sigma(y) = y$
- next step:  $(x, y, N_A^1) \rightarrow \text{enc}_{k_B}^a(y, N_A^2)$
- incoming message:  $(N_B^1, (\text{ok}, N_B^2), N_A^1)$

action:

- set  $\sigma'(y) = (\text{ok}, N_B^2)$
- send  $\text{enc}_{k_B}^a((\text{ok}, N_B^2), N_A^2)$

Bob

- substitution:  $\sigma(z) = N_A^1 \neq N_A^2$
- next step:  $\text{enc}_{k_B}^a((\text{ok}, N_B^2), z) \rightarrow \text{enc}_{k_A}^a(N_B^3)$
- incoming message:  $\text{enc}_{k_B}^a((\text{ok}, N_B^2), N_A^2)$

action:

incoming message cannot be parsed with receive/send rule, no action taken



## situation in protocol run

- memory: substitution  $\sigma$
- next action:  $r \rightarrow s$
- incoming message:  $m$

## parsing $m$ with $r \rightarrow s$

- update substitution to  $\sigma'$
- outgoing term:  $\sigma'(s)$

## definition: matching

a term  $r$  matches with message  $m$  and substitution  $\sigma$  via substitution  $\sigma'$ , if

- $\sigma'(r) = m$ , and
- $\sigma'(x) = \sigma(x)$  for all  $x$  with  $\sigma(x) \neq x$ .

$\sigma'$  consistent with incoming message

$\sigma'$  consistent with state

# 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  $\sigma$

## example situation

- next receive/send rule:

$$(\text{enc}_{k_A}^a(x_A^1, N_A^1), \text{sig}_{k_B}(x_A^2, y)) \rightarrow \\ \text{sig}_{k_A}(y, x_A^1, x_A^2, N_A^1, N_A^2)$$

- 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?

- $(\text{enc}_{k_A}^a(N_B^1), \text{sig}_{k_B}(N_B^2, N_C))$
- $(\text{enc}_{k_A}^a(N_B^2, N_A^1), \text{sig}_{k_B}(N_B^1, N_C))$
- $(\text{enc}_{k_A}^a(N_B^1, N_A^1), \text{sig}_{k_B}(N_B^2, N_C))$
- $(\text{enc}_{k_A}^a(N_B^1, N_A^1), \text{sig}_{k_B}(N_B^2, N_B^2))$

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

## Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



definition: protocol instance  $\mathcal{I}$

sequence of actions

- $r_0 \rightarrow s_0$ ,
- $r_1 \rightarrow s_1$ ,      with  $\mathcal{V}(s_i) \subseteq \cup_{j \leq i} \mathcal{V}(r_j)$  for all  $i$ .  
 $(\mathcal{V}(t)$ : variables in term  $t$ )
- ...,
- $r_{n-1} \rightarrow s_{n-1}$

example role

1.  $\epsilon \rightarrow \text{enc}_{k_B}^a(A, N_A)$
2.  $\text{enc}_{k_A}^a(N_A, x) \rightarrow \text{enc}_{k_B}^a(x)$

consequences for modeling

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

definition: protocol

protocol consists of

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

# Formal Representation of Needham-Schroeder I



## example

formal representation of the Needham-Schroeder protocol

### protocol

$A \rightarrow B \quad \text{enc}_{k_B}^a(A, N_a)$   
 $B \rightarrow A \quad \text{enc}_{k_A}^a(N_a, N_b)$   
 $A \rightarrow B \quad \text{enc}_{k_B}^a(N_b)$

### formalization

Alice

$$\begin{array}{ccc} \epsilon & \rightarrow & \text{enc}_{k_B}^a(A, N_A) \\ \text{enc}_{k_A}^a(N_A, y) & \rightarrow & \text{enc}_{k_B}^a(y) \end{array}$$

Bob

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

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

## Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Modeling Protocols: Attack?

## components: instances

- contain r/s actions
- example NSL:  $\text{enc}_{k_B}^a(A, x) \rightarrow \text{enc}_{k_A}^a(x, N_B)$
- hard-coded assumption: Bob replies to Alice

## fixed by protocol

- actual “protocol” (r/s actions)
- participants and “roles”
  - Alice: initiator
  - Bob: responder

## issue

- protocol as formalized cannot be attacked!
- need different situation to show attack ...



## example

formal representation of the Needham-Schroeder protocol with attacker

messages by A, B

$$\begin{array}{ll} A \rightarrow C(A) & \text{enc}_{k_C}^a(A, N_A) \\ B \rightarrow A(A) & \text{enc}_{k_A}^a(N_A, N_B) \\ A \rightarrow C(A) & \text{enc}_{k_C}^a(N_B) \end{array}$$

formalization

Alice

$$\begin{array}{ccc} \epsilon & \rightarrow & \text{enc}_{k_C}^a(A, N_A) \\ \text{enc}_{k_A}^a(N_A, y) & \rightarrow & \text{enc}_{k_C}^a(y) \end{array}$$

Bob

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



saw

formal protocol model, example for protocol run

crucial question

when is a protocol secure?

intuition

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

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

consequence?

need: attacker model

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

# Executing Instances I

situation: instances given

1. Alice as initiator with Charlie ( $\mathcal{A}$ )
2. Bob as responder with Alice (played by  $\mathcal{A}$ )

adversary:

- $\mathcal{A}$  controls  $C$  (knows  $C$ 's private key)
- $\mathcal{A}$  impersonates  $A$  in session with Bob

execution steps

1.  $A \rightarrow C$
2.  $A \rightarrow B$
3.  $B \rightarrow A$
4.  $C \rightarrow A$
5.  $A \rightarrow C$
6.  $A \rightarrow B$

attack works only with this order

allow  $\mathcal{A}$  to control order

# Execution Order

## intuition

$P = \{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$  protocol

- each instance  $\mathcal{I}_j$  has  $|\mathcal{I}_j|$  steps
- instance  $\mathcal{I}_j$  activated “at most  $|\mathcal{I}_j|$  times”
- order inside  $\mathcal{I}_j$ : fixed by protocol

## definition: execution order

$P = \{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$  protocol. An **execution order** for  $P$  is a sequence  $\mathbf{o}$  over  $\{0, \dots, n-1\}$  such that each  $j \in \{0, \dots, n-1\}$  appears at most  $|\mathcal{I}_j|$  times.

## notation

- | $\mathbf{o}(t)$ : $t$ -th element in $\mathbf{o}$                                              | position $t$                                                         |
|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| • $\mathbf{o}(t)$ : $t$ -th element in $\mathbf{o}$                                            | # $\mathbf{o}(t)$ -th step of instance $\mathcal{I}_{\mathbf{o}(t)}$ |
| • $\#\mathbf{o}(t)$ : $ \{\ell \mid \ell < t \text{ and } \mathbf{o}(\ell) = \mathbf{o}(t)\} $ |                                                                      |

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

## Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



## situation

- attacker controls scheduling, network

assumption:  $\mathcal{A}$  controls everything 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!



## definition: attack

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

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

$$\sigma(r_{\#o(k)}^{o(k)}) \in \text{DY} \left( I \cup \left\{ \sigma(s_{\#o(\ell)}^{o(\ell)}) \mid o \leq \ell < k \right\} \right).$$

## 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)

# Successful Attack

next step

definition of secure protocol / successful attack

difficulty: different goals

- **authentication**: Bob only accepts when he “talked” to Alice
- **key exchange**: adversary obtains no information about the key
- **secure channel**: adversary has no information about, and cannot influence, messages exchanged on the channel
- **electronic voting**: votes authenticated, counted correctly, “secret”
- ...





## 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  $(o, \sigma)$  is **successful**, if:

$$\text{FAIL} \in \text{DY} \left( I \cup \left\{ \sigma(s_{\#o(\ell)}^{o(\ell)}) \mid o \leq \ell < |o| \right\} \right).$$

literature: secret instead of FAIL



## 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

# Needham Schroeder Formalization with Attacker

Alice's r/s rules

$$\begin{array}{lll} r_o^A \rightarrow s_o^A & \epsilon & \rightarrow \text{enc}_{k_c}^a(A, N_A) \\ r_1^A \rightarrow s_1^A & \text{enc}_{k_A}^a(N_A, y) & \rightarrow \text{enc}_{k_c}^a(y) \end{array}$$

Bob's r/s rules

$$\begin{array}{lll} r_o^B \rightarrow s_o^B & \text{enc}_{k_B}^a(A, x) & \rightarrow \text{enc}_{k_A}^a(x, N_B) \\ r_1^B \rightarrow s_1^B & [\text{BREAK}, N_B] & \rightarrow \text{FAIL} \end{array}$$

attack

execution order  $o = ABAB$ , substitution:  $\sigma(x) = N_A, \sigma(y) = N_B$

actual messages

| step | message sent by $\mathcal{A}$  | recipient of $\mathcal{A}$ message | message received as reply      |
|------|--------------------------------|------------------------------------|--------------------------------|
| 0    | $\epsilon$                     | $A$                                | $\text{enc}_{k_c}^a(A, N_A)$   |
| 1    | $\text{enc}_{k_B}^a(A, N_A)$   | $B$                                | $\text{enc}_{k_A}^a(N_A, N_B)$ |
| 2    | $\text{enc}_{k_A}^a(N_A, N_B)$ | $A$                                | $\text{enc}_{k_c}^a(N_B)$      |
| 3    | $[\text{BREAK}, N_B]$          | $B$                                | FAIL                           |

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

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.)



## examples

- key exchange  $\rightsquigarrow$  key  $k$ 
  - $\mathcal{A}$ :  $[\text{BREAK}, k]$  to Alice
  - Alice replies with **FAIL**
- electronic voting  $\rightsquigarrow$  Bob votes  $vote_B$ 
  - $\mathcal{A}$ :  $[\text{BREAK}, vote_B]$  to Bob
  - Bob replies with **FAIL**

## careful

voting example does not work like this! How can we fix this?

## important

**BREAK**-messages distinguishable from “normal” messages: **FAIL** value used nowhere else

## issues

- “bug” in the examples?
- limits of this approach?

## 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?

# (IN)SECURE Examples and Security Proofs

seen

definition of security for protocols

## examples

- Needham-Schroeder protocol is insecure
- Needham-Schroeder-Lowe protocol is secure

really? missing?

- NSL formalization with BREAK/FAIL (easy)
- security proof

## security proofs in the lecture

- past: proof of example protocol as discussion on blackboard
- in class: no complete security proof (time/benefit tradeoff)
- example: see solution of exercise 2.2, contains “handwaving”

## key point in lecture

- manual analysis: expensive and error-prone
- alternative: automatic analysis

## consequence

focus on automatic analysis

# 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.

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Goal: Automatic Analysis

## security definition

A protocol  $P$  is secure if there is no successful attack on  $P$ .

(Recall complex definition of successful attack)

## computational decision problem

*Problem:* INSECURE

*Input:* protocol  $P$  (including initial adversary knowledge  $I$ )

*Question:* is there a successful attack on  $P$ ?

## algorithm for INSECURE?

is the problem decidable?

# Obstacles to Decidability

## related problems

- undecidable problems for term rewriting systems
- term unifiers can be exponentially large (exercise)
- can express **if/then** – encode halting problem?

## simpler setting

- simple scenario: fixed number of sessions
- fixed number of r/s actions
- simple term replacement rules

# The Rusinowitch-Turuani Theorem

## theorem

INSECURE is NP-complete.

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

## consequences

- problem is decidable
- (probably) no efficient algorithm, but
  - possible for small instances (“three line programs”)
  - approach with SAT-solver, constraint solver, ...
- many extensions proved since then, see later

## proof (today & next week)

- presentation based on original paper
- consider only encryption and nonces

# P and NP: A Brief Reminder

machine model

Turing Machine (TM): abstraction of (e.g.) Java programs

P: determinism

- problems solvable in polynomial time on **deterministic machines**
- “normal” efficient algorithms

NP: nondeterminism

- problems solvable in polynomial time on **non-deterministic machines**
- “magic guess” algorithms

examples for NP problems

- satisfiability (NP-c)
- clique (NP-c)
- graph isomorphism
- subgraph isomorphism (NP-c)
- integer factorization
- bin-packing (NP-c)
- scheduling problems (NP-c)
- subset-sum (NP-c)
- traveling salesman (NP-c)
- ...

## Example NP algorithms

### Satisfiability for Propositional Logic

**Input:** propositional formula  $\varphi$   
**guess** assignment  $\Pi: \mathcal{V}(\varphi) \rightarrow \{0, 1\}$   
**verify** that  $\Pi \models \varphi$

### Clique Search in Graphs

**Input:** graph  $G = (V, E)$ , number  $k$   
**guess**  $C \subseteq V$  with  $|C| = k$   
**verify** that  $C \times C \subseteq E$

### Generic NP problem

**Input:** instance  $I$   
**guess** poly-length “witness” for  $I$   
**verify** that witness is correct in polynomial time

generic witness? path through nondeterministic configurations  
theorem NP is class of “efficiently verifiable” problems

# Proof of Rusinowitch-Turuani Theorem

theorem

INSECURE is NP-complete.

two parts

1. INSECURE  $\in$  NP
2. INSECURE is NP-hard

more interesting part: INSECURE  $\in$  NP

approach: guess & verify

**Input:** protocol  $P$ , initial knowledge  $I$

**guess** attack

**verify** that attack is successful

need to show

1. if  $P$  is insecure, there is a “short successful attack”
2. attack can be verified efficiently

# Rusinowitch-Turuani Algorithm

## NP-algorithm for INSECURE

input: protocol  $P = \{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$ , with initial knowledge  $I$

1. guess execution order  $\mathbf{o}$  of  $P$
2. guess short representation of substitution  $\sigma$  for variables in  $P$
3. verify  $\sigma(r_{\#o(k)}^{o(k)}) \in \text{DY} \left( I \cup \left\{ \sigma(s_{\#o(\ell)}^{o(\ell)}) \mid o \leq \ell < k \right\} \right)$  for all  $k$
4. verify  $\text{FAIL} \in \text{DY} \left( I \cup \left\{ \sigma(s_{\#o(\ell)}^{o(\ell)}) \mid o \leq \ell < |o| \right\} \right)$
5. accept if all checks successful

(nondeterministic) polynomial time?

- length of  $\mathbf{o}$  and representation of  $\sigma$  polynomial in input length?
- verification possible in (deterministic) polynomial time?

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Short Representation of $\sigma$

fact

“exponentially long attacker terms” needed for some protocols

↔ exercise

needed: short representation of terms

- required terms “long,” but structurally simple
- “many copies” of identical subterms

idea: “compress” by avoiding repetitions

definition (DAG representation)

DAG representation of  $S \subseteq \mathcal{T}$ : edge-labeled graph  $G = (V, E)$  with

- $V = \text{Sub}(S)$ ,
- $E = \left\{ v_s \xrightarrow{\text{left}} v_e \mid \exists b, v_s = [v_e, b] \text{ or } v_s = \text{enc}_{v_e}^s(b) \text{ or } v_s = \text{enc}_{v_e}^a(b) \right\}$   
 $\cup \left\{ v_s \xrightarrow{\text{right}} v_e \mid \exists b, v_s = [b, v_e] \text{ or } v_s = \text{enc}_b^s(v_e) \text{ or } v_s = \text{enc}_b^a(v_e) \right\}$

$|S|_{\text{DAG}}$ : number of nodes in DAG representation of  $S$

## Term Compression: Example

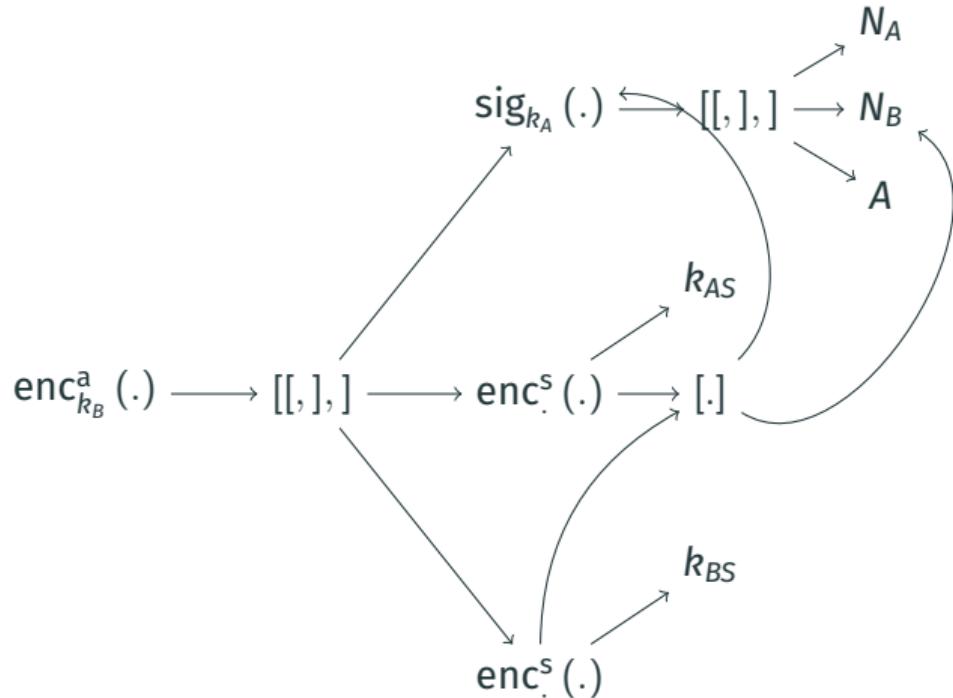
example term

$$\text{enc}_{k_B}^a \left( \text{sig}_{k_A}(N_A, N_B, A), \text{enc}_{k_{AS}}^s(N_B, \text{sig}_{k_A}(N_A, N_B, A)), \text{enc}_{k_{BS}}^s(N_B, \text{sig}_{k_A}(N_A, N_B, A)) \right)$$

term representation as graph

- following slide: complete term / graph representation, compression of repeated elements
- (sets of) terms: DAGS with fan-in 1 (i.e., forests)
- note modeling of symmetric / asymmetric encryption, sequences

# Term Compression: Trees and DAGs



## compression

- tree: 29
- DAG: 14

(3-tuple counts as 2 nodes)



want to show

if there is a successful attack on  $P$ , then there is a successful attack  $(o, \sigma)$  such that

$$|\{\sigma(x) \mid x \text{ variable in } P\}|_{\text{DAG}} \leq p(|P|),$$

for a fixed polynomial  $p$ .

crucial role in NP membership proof

shows: if there is an attack, then there is one with a “short” representation

forgot something?

also need: our algorithms work with DAG representation

- (easy to see, standard techniques)

## Exercise

### Task (exponential attack size)

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

- There are two instances:
  1.  $\mathcal{I}_1$  has a single receive/send action  $[x_1, \dots, x_i] \rightarrow \text{enc}_k^s([t_1, t_2])$ , with
$$t_1 = [x_1, [x_2, [x_3, [x_4, [\dots, [x_{i-1}, [x_i, o]] \dots ]]]]]$$
$$t_2 = [[[[\dots [[o, x_i], x_{i-1}], \dots ], x_4], x_3], x_2], x_1].$$
  2.  $\mathcal{I}_2$  has a single receive/send action  $\text{enc}_k^s(y, y) \rightarrow \text{FAIL}$ .
- 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?



## notation

- $t$  term,  $S$  set of terms,  $\sigma$  substitution, then  $S\sigma = \{\sigma(t) \mid t \in S\}$
- $t$  term,  $x$  variable, then  $[x \leftarrow t]$  substitution  $\sigma$ :  $\sigma(x) = t$ ,  $\sigma(y) = y$  for  $y \neq x$

## lemma

$S \subseteq \mathcal{T}$ ,  $x$  variable,  $t$  message, then  $|S[x \leftarrow t]|_{\text{DAG}} \leq |S \cup \{t\}|_{\text{DAG}}$ .

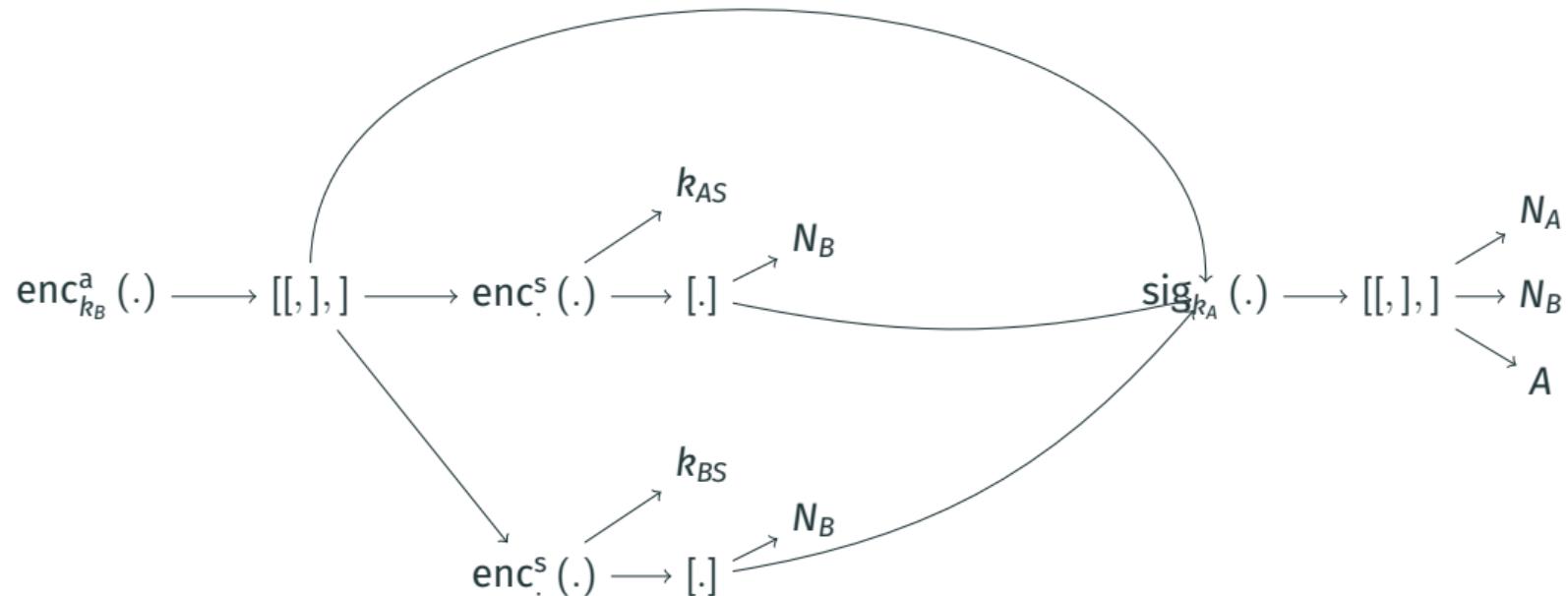
## corollary

$S \subseteq \mathcal{T}$ ,  $\sigma$  ground substitution on variables  $x_1, \dots, x_k$ , then  
 $|S\sigma|_{\text{DAG}} \leq |S \cup \{\sigma(x_1), \dots, \sigma(x_k)\}|_{\text{DAG}}$ .

### relevance

assigning  $t$  to many occurrences  
is not more expensive than  
adding  $t$  once

## Replacing All Variable Occurrences With Same Term



note something odd? equivalent nodes would be identified.

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



want to show

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

definition

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

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

definition

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

remark

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

## 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.



## simplification

attack consists of

- substitution  $\sigma$
- execution order  $o$

## rule application

- $r_{\#o(0)}^{o(0)} \rightarrow s_{\#o(0)}^{o(0)}$
- $r_{\#o(1)}^{o(1)} \rightarrow s_{\#o(1)}^{o(1)}$
- $\vdots$
- $r_{\#o(n)}^{o(n)} \rightarrow s_{\#o(n)}^{o(n)}$

## simplification

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

# Attacks Against Protocols: Simple Terms in Substitutions

## seen up to now

- substitutions: reference nonces, identities, keys, ... from protocols
- in particular:  $\sigma(x)$  **atomic** in most examples
- atomic case: short attacks given ( $|\sigma(x)|_{\text{DAG}} = 1$ )

## question

when do more complex terms show up for  $\sigma(x)$ ? how complex does it get?

# Example



protocol fragment (cp. Woo Lam Protocol)

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

attack

$$\begin{aligned} \sigma(x) &= \text{enc}_{k_{AS}}^a (A, \text{MAC}_{k_{AS}} (N_A, A, S)) \\ &\dots \end{aligned}$$

represent as receive/send actions

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

term complexity

- Bob cannot build  $\text{enc}_{k_{AS}}^s (\dots)$ , needs variable
- $\sigma(x)$  is a “complex” term
- can be made arbitrarily complex  $\rightarrow$  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?”



## parsing lemma

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

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

## informal justification

- assume  $x$  with  $|\sigma(x)| > 1$  counter-example
- for all  $t \in \text{Sub}(P)$  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  $(\sigma, o)$  minimal attack

## statement

terms in variables represent steps in the protocol



# Parsing Lemma Proof

<https://cloud.rz.uni-kiel.de/index.php/s/WoWy5TScbjFai6b>

## video content

- Proof of Parsing Lemma by explicit construction of a “smaller” attack
- note errata slide!

## study

- watch video—feedback welcome!
- video slides contained in slide set (gray background), additional material in lecture notes
- next week: discussion of content (in small groups), bring questions!



## parsing lemma statement

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

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

## proof structure

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

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

## 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  $(o, \sigma)$  such that there is a variable  $x$  with  $\sigma(x) \neq x$  and  $\sigma(x) \notin \text{DY}(S)$ , where  $S$  is the set of terms available to the adversary at the step where the first term containing  $\sigma(x)$  is sent?

# Errata for Video “Parsing Lemma Proof”

## corrections

- second handwritten page:
  - induction step  $i \rightsquigarrow i - 1$ : should be  $\sigma(\mathbf{x}) \in \text{Sub}(S_i)$
- third handwritten page:
  - last line should be “So,  $(o, \sigma')$  is successful attack”

## additions

- third handwritten page:
  - $T_o = \{\sigma(s_0), \dots, \sigma(s_{j-1})\}, T'_o = \{\sigma'(s_0), \dots, \sigma'(s_{j-1})\}$
  - $\sigma(r_j) \in T_n, \sigma'(r_j) \in T'_n$

# Video Lecture: Feedback wanted



## questions

- audio/video quality?
- proof presentation as screenshots, or “live writing?”
- better as video or “live Zoom session?”
- any suggestions?

## feedback crucial

- your perspective very different from mine!
- constructive criticism always welcome
- review after week 6!

## remember

- we're all still learning this
- new tools, concepts
- big playground :-)



## theorem

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

### 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  
 $\rightsquigarrow$  small DAG size

# Proof of Rusinowitch Turuani Theorem

## theorem

$(\sigma, o)$  minimal successful attack on  $P$ , then  $|\sigma(x)|_{\text{DAG}} \leq |\{r_o, \dots, r_n, s_o, \dots, s_n\}|_{\text{DAG}}$ .

## proof

$U$  set of variables, then  $\overline{U} = \{\sigma(x) \mid x \in U\}$ .

- construct  $S_i \subseteq \text{Sub}(\{r_o, \dots, s_n\})$ ,  $V_i \subseteq \mathcal{V}$ :

$$|\sigma(x)|_{\text{DAG}} \leq |S_i \cup \overline{V_i}|_{\text{DAG}}$$

- 1.  $i = o$ : choose  $S_o = \emptyset$ ,  $V_o = \{x\}$ .
- 2.  $i \rightarrow i + 1$ 
  - choose  $y \in V_i$ ,  $t$  from protocol with  $\sigma(y) = \sigma(t)$
  - $S_{i+1} = S_p \cup \{t\}$
  - $V_{i+1} = V_p \setminus \{y\} \cup \mathcal{V}(t)$
  - ind:  $|\sigma(x)|_{\text{DAG}} \leq |S_i \cup \overline{V_i}|_{\text{DAG}}$
  - now:  $|\sigma(x)|_{\text{DAG}} \leq |S_i \cup \overline{V_i}|_{\text{DAG}} \leq |S_{i+1} \cup \overline{V_{i+1}}|_{\text{DAG}}$

### usage

- $S_i$ : references into protocol
- $V_i$ : TODO-list: variables to be replaced by references

### start/end, termination

- $i = o$ :  $|\sigma(x)|_{\text{DAG}} \leq |\overline{\{x\}}|_{\text{DAG}}$
- $V_i = \emptyset$ :  $|x|_{\text{DAG}} \leq |S_i|_{\text{DAG}} \leq |r_o, \dots, s_n|_{\text{DAG}}$
- why do we reach  $V_i = \emptyset$ ?
  - $\sum_{z \in V_i} |\sigma(z)|$  decreases

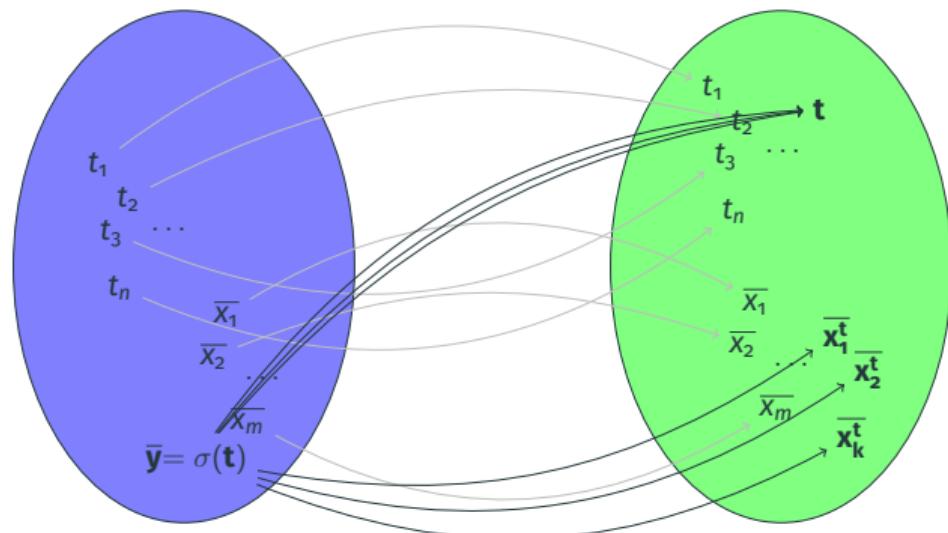
# Proof of Rusinowitch-Turuani Theorem

claim

$$|S_i \cup \bar{V}_i|_{\text{DAG}} \leq |S_{i+1} \cup \bar{V}_{i+1}|_{\text{DAG}} \text{ via injection } f: \text{Sub}(S_i \cup \bar{V}_i) \rightarrow \text{Sub}(S_{i+1} \cup \bar{V}_{i+1})$$

induction

- $S_{i+1} = S_i \cup \{t\}$
- $V_{i+1} = V_i \cup \mathcal{V}(t) \setminus \{y\}$



## 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.

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Proof of Rusinowitch Turuani Theorem

theorem

INSECURE is NP-complete.

two parts

1.  $\text{INSECURE} \in \text{NP}$
2.  $\text{INSECURE}$  is NP-hard

second part: INSECURE is NP-hard

recall TGI: reduce from NP-complete problem

# NP hardness proof

## TGI refresher

$L_1, L_2$  languages,  $L_1$  NP-hard and  $L_1 \leq_m^p L_2$ , then  $L_2$  NP-hard.

## reduction

$L_1 \leq_m^p L_2$  means:

$L_1$ -question can be “efficiently translated” into  $L_2$ -questions

formally

there is a total, P-computable function  $f: \Sigma^* \rightarrow \Sigma^*$  with

$$x \in L_1 \text{ iff } f(x) \in L_2 \text{ for all } x.$$

## NP-complete problem: 3SAT

SAT for formulas  $\varphi = \bigwedge_{i=1}^n (l_1^i \vee l_2^i \vee l_3^i)$ , where  $l_j^i$  literals over  $\{x_1, \dots, x_m\}$

# NP hardness reduction I

## 3SAT

- **nondeterministic step:** guess assignment  $I$  for variables of formula  $\varphi$
- **deterministic step:** check that assignment satisfies  $\varphi$

## INSECURE

- **nondeterministic step:** guess one adversary message (encoding  $I$ )
- **deterministic step:** let honest participants check that assignment satisfies  $\varphi$

## note

$\varphi$  can be hard-coded into INSECURE instance

## issues

- adversary can interfere with communication between honest principals
- use cryptography to ensure secure communication



## 3SAT instance

$$\varphi = \bigwedge_{i=1}^n (l_1^i \vee l_2^i \vee l_3^i), \text{ each } l_j^i \text{ literal over } \{x_1, \dots, x_m\}, l_j^i \in \{x_{r_{i,j}}, \overline{x_{r_{i,j}}}\}$$

crypto

needed for proof?

## protocol instances

- $A$  expects and distributes  $\{x_1, \dots, x_m\}$ -assignment  $I$ :

$$[x_1, x_2, \dots, x_m] \rightarrow [m_1, \dots, m_n], \text{ with } m_i = \text{enc}_k^s([i, x_{r_{i,1}}, x_{r_{i,2}}, x_{r_{i,3}}])$$

- for  $i \in \{1, \dots, n\}$ , satisfying assignment  $(\alpha, \beta, \gamma)$  of clause  $i$ : instance  $B_{(\alpha, \beta, \gamma)}^i$   
 $\text{enc}_k^s([i, \alpha, \beta, \gamma]) \rightarrow k_i$
- final assembly  $F$ : if all clauses satisfied, release FAIL

$$[k_1, \dots, k_n] \rightarrow \text{FAIL}$$

## correctness

$$\begin{array}{llll}
 \text{FAIL} & \leftrightarrow & \text{adv gets all } k_i & \leftrightarrow \text{for each } i, \text{ one } B_{(\alpha, \beta, \gamma)}^i \text{ releases } k_i \\
 & \leftrightarrow & (\alpha, \beta, \gamma) \models \text{clause} & \leftrightarrow \text{all clauses satisfied by } I \\
 & & & \leftrightarrow \varphi \text{ satisfiable}
 \end{array}$$

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

## Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

## Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

## Rusinowitch-Turuani Theorem [RT03]

INSECURE is NP-complete

model

INSECURE: instances given

- theorem only covers fixed number of instances
- instance  $\hat{=}$  protocol session

reality

unbounded number of sessions

- many users for single server
  - different (or same) users at different servers
- number of concurrent TLS sessions?

# Towards Automatic Analysis

## seen in lecture

- formalization of NS protocol must contain sessions to find attack
  - sender instance of  $A \rightarrow C$
  - receiver instance of  $A \rightarrow B$
- unsatisfying: this “tells the algorithm where to look”

## possible way out: over-approximate

- observation: more instances only make the situation worse (more insecure)
- therefore: let algorithm analyze the following:
  - sender instance of  $A \rightarrow B, A \rightarrow C, B \rightarrow C$
  - receiver instance of  $A \rightarrow B, A \rightarrow C, B \rightarrow C$
- issues?

# “parallel” attacks

## Rusinowitch-Turuani analysis

- instances fixed
- hence, protocol sessions fixed

## problem

there are issues in protocols that need an “arbitrary” number of sessions

## reference

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

# The “ffgg” Protocol



protocol

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

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  $y$



# Security of ffgg

## attack

1.  $A \xrightarrow{A} B$
- 1'.  $(A) \xrightarrow{A} B$
- 2a.  $B \xrightarrow{[N_1, N_2]} (A)$
- 2'.  $B \xrightarrow{[N'_1, N'_2]} (A)$
- 2b.  $(B) \xrightarrow{[N_1, N'_1]} A$
3.  $A \xrightarrow{\text{enc}_{k_B}^a ([N_1, N'_1, FAIL])} B$
4.  $B \xrightarrow{[N_1, N'_1, \text{enc}_{k_B}^a ([N'_1, FAIL, N_1])] (A)}$
- 3'.  $(A) \xrightarrow{\text{enc}_{k_B}^a ([N'_1, FAIL, N_1])} B$
- 4'.  $B \xrightarrow{[N'_1, FAIL, \text{enc}_{k_B}^a ([FAIL, N_1, N'_1])] (A)}$

## 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

## Exercise

### Task (the FFGG protocol: 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?



## 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  $\mathcal{I}_{A \rightarrow B}$  may be started arbitrarily often
    - “between A and B”
    - “between A and C”
    - ...

## issue

FAIL-rule may only be contained in “relevant” instance

# Unbounded Version of INSECURE

## approach

- specify instances, initial attacker knowledge as usual
- mark one instance as **goal** (usually contains FAIL constant)

## definition

protocol  $P_{\text{unb}}$  **based** on  $P$ , if  $P_{\text{unb}}$  obtained from  $P$  by

- replicating instances (with fresh variables)
- changing identities in non-goal instances

## issues

- changing identities must “respect” knowledge of keys
- straight-forward for asymmetric keys, more technical for symmetric keys
- see discussion in exercise class

## this lecture

- no formal definition
- follow these ideas in practical security specifications
- case-study (as reading exercise)  
later: modeling of  
Needham-Schroeder in ProVerif

## 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

## Theorem

the following problem is undecidable:

*Problem:* **UNBOUNDED-INSECURE**

*Input:* protocol  $P = (\{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}, I)$

*Question:* is there an insecure protocol  $P_{\text{unb}}$  based on  $P$ ?

?

missing something? we didn't even really define **UNBOUNDED-INSECURE!**

## simplification

result true for very simple modeling of **UNBOUNDED-INSECURE**

# Undecidability Result

## formalization for undecidability

“simplest” formalization of unbounded sessions: result covers more expressive models as well

## “minimal requirements”

- protocol consists of instances  $\{\mathcal{I}_0, \dots, \mathcal{I}_{n-1}\}$ , each instance has a single receive/send rule
- adversary may activate each instance as often as she wishes
- there is only a single symmetric key  $k$  shared by all protocol instances (no PKI, no identities)

## undecidability proof

works for this model

# Undecidability Proof I

## TG1 refresher

$L_1, L_2$  languages,  $L_1$  undecidable and  $L_1 \leq L_2$ , then  $L_2$  undecidable.

## reduction

$L_1 \leq L_2$  means:

$L_1$ -questions can be translated into  $L_2$ -questions.

formally:

*there is a total, computable function  $f: \Sigma^* \rightarrow \Sigma^*$  such that for all  $x$ :*

$$x \in L_1 \text{ iff } f(x) \in L_2.$$

# Post's Correspondence Problem

seen in TGI

halting problem, Rice's theorem

drawback

talk about encodings of Turing machines

**classical problem**

*Problem:* PCP (Post's correspondence problem)

*Input:*  $(x_1, y_1), \dots, (x_n, y_n)$  with  $x_i, y_i \in \{0, 1\}^*$

*Question:* Is there a sequence  $i_1, \dots, i_\ell$  with  $x_{i_1}x_{i_2} \dots x_{i_\ell} = y_{i_1}y_{i_2} \dots y_{i_\ell}$ ?

**theorem**

PCP is undecidable.

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>

# Undecidability Proof

want to show

UNBOUNDED-INSECURE is undecidable

proof (sketch)

- show  $\text{PCP} \leq \text{UNBOUNDED-INSECURE}$
- describe **computable** function  $f: \{0,1\}^* \rightarrow \{0,1\}^*$  such that  
 $x \in \text{PCP} \text{ iff } f(x) \in \text{UNBOUNDED-INSECURE}$

## Undecidability Proof II

PCP

infinite search space: find  $i_1 \dots i_\ell$   
with  $x_{i_1} \dots x_{i_\ell} = y_{i_1} \dots y_{i_\ell}$

UNBOUNDED-INSECURE

infinite search space: choice of instances in

- protocol  $P_{\text{unb}}$  based on  $P$
- execution order of attack

let instances perform “concatenation” of PCP strings

note

$x_1, \dots, x_n, y_1, \dots, y_n$  can be hard-coded into UNBOUNDED-INSECURE instance

issues

- adversary can use “fake PCP substrings”
- use cryptography to authenticate substrings and concatenation from PCP instance



## input

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

## idea

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

## instances

- for each  $i \in \{1, \dots, n\}$ :  $A_{\text{init}}^i \quad \epsilon \rightarrow \text{enc}_k^s \left( [x_{|x_i|}^i, [x_{|x_i|-1}^i, [\dots, x_1^i] \dots]], [y_{|y_i|}^i, [y_{|y_i|-1}^i, [\dots, y_1^i] \dots]] \right)$
- f.e.  $i: A_{\text{step}}^i \quad \text{enc}_k^s ([x, y]) \rightarrow \text{enc}_k^s \left( [x_{|x_i|}^i, [x_{|x_i|-1}^i, [\dots, [x_1^i, x]]]], [y_{|y_i|}^i, [y_{|y_i|-1}^i, [\dots, [y_1^i, y]]]] \right)$
- verification  $B_{\text{check}}$ :  $\text{enc}_k^s ([x, x]) \rightarrow \text{FAIL}$

## correctness

- $A_{\text{init}}^i, A_{\text{step}}^i$ : Adversary gets exactly  $\text{enc}_k^s ([t_1, t_2])$ , where  $t_1, t_2$  constructed by “domino rules”
- $B_{\text{check}}$ : if adversary solves domino puzzle, release FAIL constant
- so: domino solvable iff protocol insecure in unbounded setting

## 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 Edge of Decidability

## lecture results

- Rusinowitch Turuani: **bounded** sessions decidable
- PCP reduction: **unbounded** sessions undecidable

## middle ground?

- “restricted” unbounded sessions?
- simple loops in protocol?
- data structure processing?
- more complex protocol goals?

## results

there is a lot!

# The Edge of Decidability: References I

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

## The Edge of Decidability: References II

- 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>
- 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

## The Edge of Decidability: References III

- 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>
- Robert Künnemann, İlkan 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>
- ...

# Undecidability: Consequences

## result

(in)security with arbitrary many sessions is undecidable

## consequences

- no complete “push-button” analysis of security
  - hardly unexpected
- justification for “user-unfriendly” input for Rusinowitch Turuani algorithm
  - some automatic “preprocessing” possible, but does not solve problem

## analysis still required

what are options for practice?

## approach

- fixed choice of instances
  - fixes identities, roles (e.g., “Alice as initiator in session with Bob”)
  - fixes number of sessions
  - fixes max. number of messages
- attack found: protocol insecure
- no attack found: secure **in this scenario**

## justification

- most attacks found by checking small systems
- unusual for an attack to require “many” sessions

## usual security approach

- worst-case assumptions
- “unusual” attacks are exactly what we do automatic analysis for
- situation not satisfying



## manual approach

- proof using protocol structure
- for every message: *if* accepted,  
*then* earlier ...
- then “protocol run as intended”

expensive and error-prone

## automatic analysis

- problem is undecidable
- cannot have both
  - **soundness** result “protocol secure” is correct
  - **completeness** if protocol secure, this is recognized
- need to look at “incomplete” algorithms

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



## construct security proof

- algorithm searches for security proof
- on failure: abort or endless loop
- algorithm is correct (sound)

## consequence

secure protocols are recursively enumerable  
(semi-decidable)

## construct attack

- algorithm searches for attack
- on failure: abort or endless loop
- algorithm is correct (sound)

## consequence

insecure protocols are recursively enumerable  
(semi-decidable)

## what's wrong?

something does not add up! (*aka don't cite this slide!*)

# Incomplete Algorithms

## seen

- searching for security proofs and attacks can never cover everything
- way out: **heuristics** (cp. NP-complete problems)

## heuristics

- there is always a price!
- what do we give up?

## over-approximate attacker

- simplified attacker model
- gives “too much power” to attacker
- constructs “over-approximated” attack
- user must check attack
- algorithm sound, not complete (for security)

## abstractions

- over-approximation of attacker
- leads to finite model
- apply model checking

lecture: skipped due to time constraints

## logic-based modeling

- models protocol properties in (FO Horn) logic
- leads to Horn theory
- apply satisfiability testing

lecture: cover this in practice (ProVerif), brief look at theory

# Computationally Nice Logics

## propositional logic

- $\varphi = \exists x_1 \forall y_1 \exists x_2 \dots \forall y_n$   
 $(x_1 \vee \overline{x_9} \vee y_4) \wedge \dots \wedge (y_6 \vee \overline{x_3} \vee \overline{y_{44}})$
- relevant algorithmic problems:  
decidable, NP-complete

## first-order logic

- $\varphi = \exists x_1 \forall y_1 \exists x_2 \dots \forall y_n$   
 $R_1(x_1, x_9, y_4) \vee (R_2(x_3, y_1, x_{13}) \wedge \dots)$
- relevant algorithmic problems: undecidable

## complexity reduction

syntactically defined sub-logic with “nicer” complexity? Horn clauses

- allows unit resolution
- “largest” sublogic for which propositional satisfiability is PTIME-solvable [Sch78]
- still undecidable first-order theory, but “better behaved”

# Logic Modeling Outline

facts

- $d(\hat{k}_C)$
- $d(\{0, 1\})$
- ...

DY deductions

- $d(\text{enc}_{k_C}^a(x)) \wedge d(\hat{k}_C) \rightarrow d(x)$
- $d(x) \wedge d(y) \rightarrow d([x, y])$
- $d(x) \rightarrow d(\text{hash}(x))$
- ...

protocol deductions

- $d(\text{enc}_{k_B}^a([A, x])) \rightarrow d(\text{enc}_{k_A}^a([B, x]))$
- $d(\text{enc}_{k_A}^a([B, x, y])) \rightarrow (\text{enc}_{k_B}^a(y))$
- ...

Horn clauses

$$(x_1 \wedge x_2 \wedge \cdots \wedge x_n \rightarrow y) \leftrightarrow (\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee y)$$

target clause

$$\neg d(\text{FAIL})$$

## 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?

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Applying the Theory: Analysis in Practice

## so far in lecture

- formal model
- formal security properties
- decidability and complexity results
- Horn modeling
- abstraction

## now: application of the theory

different analysis tools

- ProVerif (Bruno Blanchet, Vincent Cheval)
- CryptoVerif (Bruno Blanchet, David Cadé)
- FDR (Formal Systems Europe)

# ProVerif in Lecture I

## outline

- ProVerif introduction (syntax, semantics, examples)
- generalized cryptographic primitives in ProVerif
- extended security properties in ProVerif

no detailed theory (literature references)

## key references

**paper** Bruno Blanchet. "Using Horn Clauses for Analyzing Security Protocols". In: *Cryptology and Information Security Series* 5 (2011), pp. 86–111

**update** 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>

**tool** <http://prosecco.gforge.inria.fr/personal/bblanche/proverif/>

## learning goals

practice

analyzing protocols with ProVerif

security properties

study of complex properties (without detailed formal model)

conceptual

undecidable problems in real life

consequences for tool application

## not a learning goal

complete introduction to ProVerif (see reading exercise on Needham-Schroeder modeling)

## non-trivial examples

- certified email protocol, including ssh layer, JFK protocol (candidate for IKE replacement), secure filesystem Plutus, web service verification
- e-voting protocols, authenticated routing, zero knowledge protocols
- TLS (F# implementation for .NET), 5G EAP-TLS
- Telegram, Bitcoin Smart Contracts, Healthcare protocols
- ... (see Google Scholar, ProVerif since 2020)

## original reference

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

## features

- analysis of protocols in symbolic model
- can handle an unbounded number of protocol sessions (necessarily incomplete)
- user-provided specification of cryptographic primitives and security properties

## example security properties

- secrecy
- strong secrecy
- authentication
- correspondence
- observational equivalence

## incompleteness consequences

- “don’t know”
- non-termination

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# ProVerif „Hello, World“

hello.pv (from PV doc)

free c:channel.

free Plain:bitstring [private].

free RSA:bitstring [private].

query attacker(RSA).

query attacker(Plain).

process

out(c,RSA);

o

elements

**free** free algebraic variables

**channel** communication channel

**bitstring** data type

**private**  $\mathcal{A}$  cannot (directly) access

**query** security property

**out** send on channel

execution

- send “private” value RSA on public channel
- query: secrecy for
  - RSA
  - Plain

# ProVerif usage

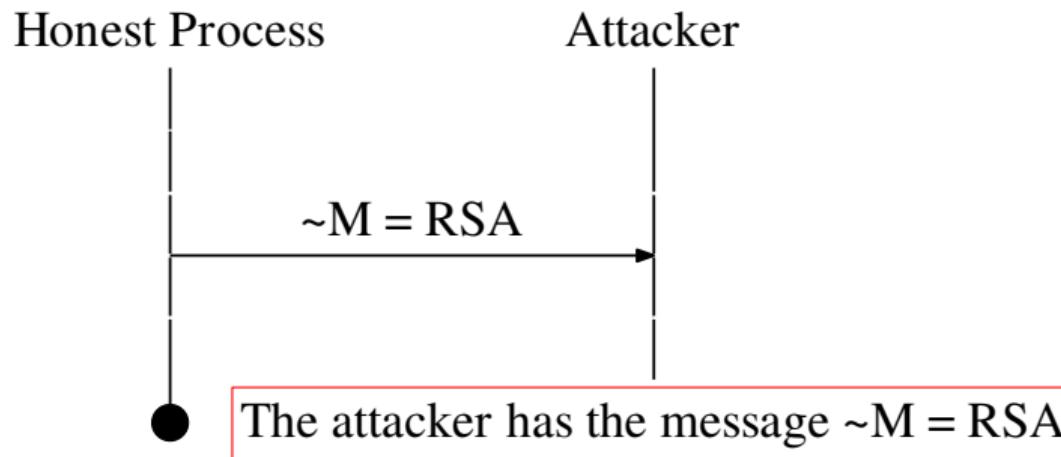
## command line tool

```
$ proverif 2021_01_05_lecture_08/01_hello.pv
$ proverif -graph targetDir 2021_01_05_lecture_08/01_hello.pv
$ proverif -html targetDir 2021_01_05_lecture_08/01_hello.pv
```

## live demo

# ProVerif Analysis Result: Hello, World

A trace has been found.



# The Handshake Protocol



example: handshake (from ProVerif tutorial)

$A \rightarrow B \quad k_A$

$B \rightarrow A \quad \text{enc}_{k_A}^a (\text{sig}_{k_B} ([k_B, k]))$

$A \rightarrow B \quad \text{enc}_k^s (\text{FAIL})$

properties?

- intended security properties?
- protocol secure?

attack

$\mathcal{A} \rightarrow B \quad k_{\mathcal{A}}$

$B \rightarrow \mathcal{A} \quad \text{enc}_{k_{\mathcal{A}}}^a (\text{sig}_{k_B} ([k_B, k]))$

$A \rightarrow B \quad k_A$

$\mathcal{A} \rightarrow A \quad \text{enc}_{k_A}^a (\text{sig}_{k_B} ([k_B, k]))$

$A \rightarrow B \quad \text{enc}_k^s (\text{FAIL})$

fix

add receiver to message

# Analyzing the Handshake Protocol in ProVerif

## steps

1. specify cryptographic primitives
2. specify communication infrastructure
3. specify adversary goal
4. specify Alice & Bob
5. specify “main process”



## 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 (see example scripts)

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy



## 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

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

# Equational Theories: Examples

primitives

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

# “Simple” Equational Theories

## definition

$\rightarrow_E$  is

- **confluent**, if for all  $t, t_1, t_2$  with  $t \rightarrow_E^* t_1$  and  $t \rightarrow_E^* t_2$  there is some  $t'$  with  $t_1 \rightarrow_E^* t'$  and  $t_2 \rightarrow_E^* t'$ .
- **terminating**, if there is no infinite sequence  $t_1, t_2, \dots$  with  $t_i \neq t_{i+1}$  and  $t_i \rightarrow_E t_{i+1}$  for all  $i$ .
- $E$  is **convergent**, if  $\rightarrow_E$  is confluent and terminating
- $E$  is **convergent subterm theory**, if
  - $E$  is convergent, and
  - for all  $(l, r) \in E$ :  $r$  is subterm of  $l$  or constant



## definition

A term  $t$  is in  $E$ -normal-form if  $t = t'$  for all  $t \rightarrow_E t'$ .

## lemma & definition

If  $E$  is convergent, then for every term  $t$ , there is a unique term  $[t]$  with

- $[t]$  is in  $E$ -normal-form,
- $t \equiv_E [t]$ .

## lemma

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

## computation of normal form

How do we compute  $[t]$  from  $t$ ? for a convergent theory?



## primitives

- asym. encryption  $\text{dec}_{\hat{k}_A}^a(\text{enc}_{k_a}^a(x)) = x$
- sym. encryption  $\text{dec}_k^s(\text{enc}_k^s(x)) = x$
- signature
  - $\text{check}(k_A, \text{sig}_{k_A}(x)) = \text{ok}$
  - $\text{extr\_key}(\text{sig}_{k_A}(x)) = k_A$
  - $\text{extr\_msg}(\text{sig}_{k_A}(x)) = x$
- hash function
- bit commitment  $\text{open}(\text{bc}(k, b), k) = b$
- trapdoor commitments
  - $\text{open}(\text{tdc}(m, r, t_d), r) = m$
  - $\text{tdc}(m_2, f(m_1, r, t_d, m_2), t_d) = \text{tdc}(m_1, r, t_d)$

## discussion

- complexity?
- observations?

## remember

modeling primitives at this level of abstraction loses details

## algorithms

algorithms discussed so far do not cover all of these

# 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  $\rightarrow_E$  is not a convergent subterm theory, i.e., one that is not confluent, not terminating, or not a subterm theory.

# Algorithms for Convergent Subterm Theories

## Theorem

For convergent subterm theories, the following problems are polynomial-time decidable:

- given  $E, t, t'$ , does  $t \rightarrow_E t'$  hold?
- given  $E, t, t'$ , is  $t \equiv_E t'$ ?

## Theorem

Also computable in polynomial time: given  $E, t$ , compute  $[t]$  (DAG representation)

## reference

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

# Protocol Notation with Equational Theories



Needham Schroeder

$$\begin{array}{lll} \text{Alice} & \epsilon & \rightarrow \text{enc}_{k_B}^a(A, N_A) \\ & \text{enc}_{k_A}^a(N_A, y) & \rightarrow \text{enc}_{k_B}^a(y) \end{array}$$

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

equational theory

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

“simplification”

- new notation for protocols?
- advantage?

additional equations for pairing

- $\text{split1}(\text{pair}(x, y)) = x$
- $\text{split2}(\text{pair}(x, y)) = y$

Alice

$$\begin{array}{ll} \epsilon & \rightarrow \text{enc}_{k_B}^a(A, N_A) \\ x & \rightarrow \text{enc}_{k_B}^a(\text{split2}(\text{dec}_{\hat{k}_A}^a(x))) \end{array}$$

Bob

$$y \rightarrow \text{enc}_{k_A}^a(\text{pair}(\text{split2}(\text{dec}_{\hat{k}_B}^a(y)), N_B))$$

# Handshake-Protocol in ProVerif: Primitives

**symmetric encryption**

```
type key  
fun senc(bitstring, key): bitstring  
reduc forall m:bitstring, k:key; sdec(senc(m,k),k) = m
```

**asymmetric encryption**

```
type skey  
type pkey  
fun pk(skey): pkey  
fun aenc(bitstring, pkey): bitstring  
reduc forall m:bitstring, sk:skey; adec(aenc(m,pk(sk)),sk) = m
```

**signatures**

```
type sskey  
type pskey  
fun spk(sskey): spkey  
fun sign(bitstring, sskey): bitstring  
reduc forall m:bitstring, ssk:sskey; getmess(sign(m,ssk)) = m  
reduc forall m:bitstring, ssk:sskey; checksign(sign(m,ssk),spk(ssk)) = m
```

# Handshake-Protocol in ProVerif: Communication and Adversary Goal

channel, values, attacker goal

free c:channel

free FAIL:bitstring [private]

query attacker (FAIL)

models

- c is public channel
- FAIL: bitstring, private (not in initial  $\mathcal{A}$  knowledge)
- security property:  $\mathcal{A}$  cannot derive FAIL



## protocol

```
A → B   kA
B → A   encakA (sigkB ([kB, k]))
A → B   encsk (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)



## protocol

```
A → B   kA
B → A   encakA (sigkB ([kB, k]))
A → B   encsk (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

# Handshake-Protocol in ProVerif: Main Process

## previous processes

- clientA(pkA:pkey,skA:skey,pkB:spkey)
- serverB(pkB:spkey,skB:sskey)

## main process

```
new skA:skey  
new skB:sskey  
let pkA=pk(skA) in out(c,pkA)  
let pkB=spk(skB) in out(c,pkB)  
( (!clientA(pkA,skA,pkB) | (!serverB(pkB,skB))) )
```

## features

- key generation, sending of pkA, pkB on public channel
- call of Alice and Bob with matching parameters
- !: “replication operator”

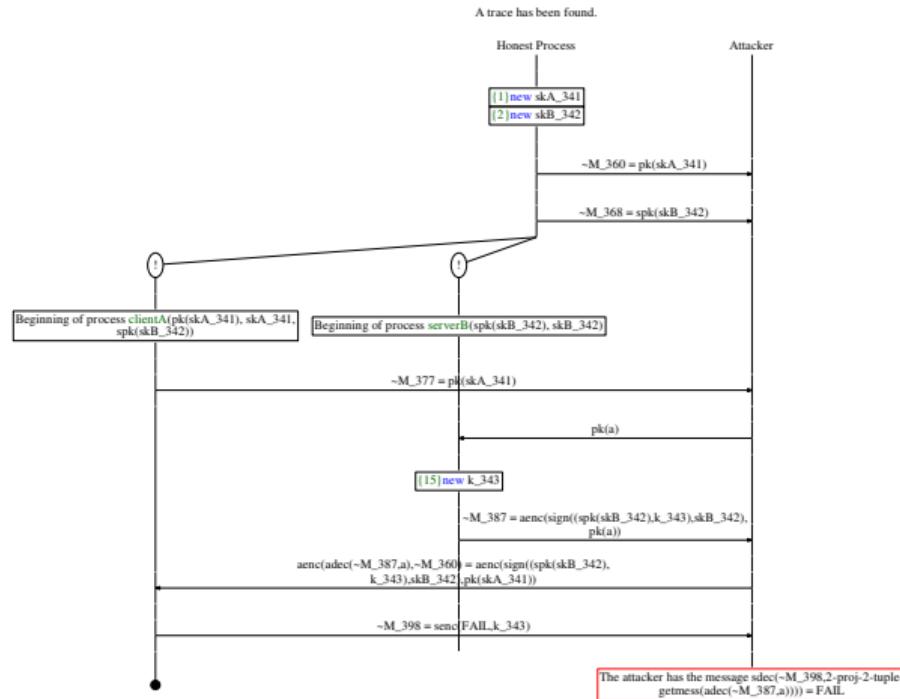
# ProVerif Script: Handshake Protocol

command line tool

```
$ proverif 2021_01_05_lecture_08/02_handshake.pv  
$ proverif -graph targetDir 2021_01_05_lecture_08/02_handshake.pv  
$ proverif -html targetDir 2021_01_05_lecture_08/02_handshake.pv
```

live demo

# ProVerif Analysis Result: Handshake



## Exercise

### Task (ProVerif example I)

Consider the following protocol:

1.  $A \rightarrow B \quad \text{enc}_{k_{AB}}^S(N_A)$
2.  $B \rightarrow A \quad [\text{enc}_{k_{AB}}^S(N_B), N_A]$
3.  $A \rightarrow B \quad N_B$

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).

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

## encryption formally

- up to now:  $\text{enc}_k^s(t)$ ,  $\text{enc}_k^a(k_A)$
- problem? multiple encryptions

## encryption in practice

- encrypting twice: different ciphertexts
- randomized algorithms

fact (formal statement: see cryptography lecture)

secure encryption **must be** randomized

# Randomized Asymmetric Encryption

ProVerif

**type** skey.

**type** pkey.

**type** coins.

**fun** pk (skey): pkey.

**fun** internal\_aenc(bitstring, pkey, coins): bitstring

**letfun** aenc(x:bitstring, y:pkey) = **new** r:coins; internal\_aenc(x,y,r).

**reduc forall** m:bitstring, k:skey, r:coins;  
adec(internal\_aenc(m,pk(k),r),k)=m.

**reduc forall** m:bitstring, pk:pkey, r:coins, getkey(internal\_aenc(m,pk,r))=pk.

# ProVerif Script: Randomized Encryption

command line tool

```
$ proverif 2021_01_12_lecture_09/03_randomized-encryption.pv  
$ proverif -graph targetDir 2021_01_12_lecture_09/03_randomized-encryption.pv  
$ proverif -html targetDir 2021_01_12_lecture_09/03_randomized-encryption.pv
```

live demo

# ProVerif: Abstraction Level?

## symbolic models

- messages as terms
- abstraction of cryptography
- no polynomial-time restriction
- no “real randomness”
- no probabilities

## cryptographic models

- messages as bitstrings
- real cryptographic algorithms
- real randomness
- probabilistic polynomial-time algorithms
- probabilistic security guarantees

What kind of model does ProVerif use?

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

## note

- in ProVerif, each value is **typed**
- allows to distinguish e.g., keys from nonces
- drawback?
  - requires implementation to check types!
- untyped version: use **bitstring**
- typed protocols: annotate messages with types

## type flaw attacks

- attacks that use “wrong types”
- rely on protocols **not** checking types
- seen examples in ffgg protocol,  
Otway-Rees protocol

## reference

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>

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

## process algebra: (applied) pi calculus

- classic technique
- specification of communicating processes
- strong type system

## security-specific aspects in ProVerif

- specification of crypto primitives with equational theories
- algorithms, properties

## references

- 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](http://dx.doi.org/10.1016/0890-5401(92)90008-4)
- Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001. ISBN: 978-0-521-78177-0

## part 1: terms

$M, N =$

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| $a, b, c, k, m, n, s$ | names                                                    |
| $x, y, z$             | variables                                                |
| $(M_1, \dots, M_k)$   | tuples                                                   |
| $h(M_1, \dots, M_k)$  | application of functions (constructors / deconstructors) |
| $M = N$               | equality                                                 |
| $M <> N$              | inequality                                               |
| $M \&\& N$            | conjunction                                              |
| $M    N$              | disjunction                                              |
| <b>not</b> ( $M$ )    | negation                                                 |

# ProVerif: Syntax

## part 2: processes

|                                                   |                                                                           |
|---------------------------------------------------|---------------------------------------------------------------------------|
| $P, Q =$                                          |                                                                           |
| o                                                 | nothing                                                                   |
| $P \mid Q$                                        | parallel execution                                                        |
| $!P$                                              | unbounded replication, $!P = P \mid !P$                                   |
| $\text{new } n : t; P$                            | value $n$ of type $t$ in process $P$                                      |
| $\text{in}(c, x : t); P$                          | store message from channel $c$ in $x$ ,<br>check type to be $t$ , run $P$ |
| $\text{out}(c, N); P$                             | send message $N$ on $c$ , run $P$                                         |
| $\text{if } M \text{ then } P \text{ else } Q$    | conditional                                                               |
| $\text{let } x = M \text{ in } P \text{ else } Q$ | pattern matching and conditional                                          |
| $R(M_1, \dots, M_k)$                              | macro application                                                         |

simplification: o, **else** can often be omitted

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Forward Secrecy

secrecy

up to now black & white view: value either secret or not

reality privacy time-dependent — today's RSA keys secure 20 years from now?

problematic scenario

2021 Alice publishes  $k_A$

2041 Bob computes  $\hat{k}_A$

Bob computes  $\text{sig}_{k_A}(m)$ ,  
 $m$  dated to 2021

2021 Alice and Bob use  $k_A$  and  $k_B$  to compute  $k_{AB}$

2021 Alice sends  $\text{enc}_{k_{AB}}^s(\text{secret})$

2041 Charlie computes  $\hat{k}_A$  and  $\hat{k}_B$ , from this  $k_{AB}$  and  
*secret*

forward security

security against **later** key breaks

# Forward Secrecy in Handshake-Protocol

## protocol (fixed)

1.  $A \rightarrow B \quad k_A$
2.  $B \rightarrow A \quad \text{enc}_{k_A}^a(\text{sig}_{k_B}([k_A, k_B, k_{AB}])))$
3.  $A \rightarrow B \quad \text{enc}_{k_{AB}}^s(\text{FAIL}).$

## asymmetric situation

- always: if  $\hat{k}_A$  and  $\hat{k}_B$  known later, protocol run can be reconstructed
- if  $B$  server: probability that  $\hat{k}_B$  gets known higher (more attractive goal)
- FAIL must stay secret if  $\hat{k}_B$  gets known later

## analysis

- if  $\hat{k}_A$  and  $\hat{k}_B$  secret: FAIL secret
- if  $\hat{k}_A$  known “later:” FAIL derivable
- if  $\hat{k}_B$  known: protocol insecure
- if  $\hat{k}_B$  known “later:” FAIL not derivable

## conclusion

“cost” of revealing keys depends on usage in protocol

## modeling

- split protocol into phases  $0, \dots, n - 1$
- keyword **phase**
- models “global time”
- leaking of  $\hat{k}_B$  in phase  $i$ : **phase  $i$ ; out( $c$ ,  $\hat{k}_B$ )**

# Forward Secrecy in ProVerif II

fixed handshake protocol (informal)

1.  $A \rightarrow B \quad k_A$
2.  $B \rightarrow A \quad \text{enc}_{k_A}^a(\text{sig}_{k_B}([k_A, k_B, k_{AB}]))$
3.  $A \rightarrow B \quad \text{enc}_{k_{AB}}^s(\text{FAIL}).$

protocol: global & Alice

free c:channel.

free FAIL:bitstring [private].

query attacker(FAIL).

let clientA(pkA:pkey,skA:skey,pkB:spkey)=

out (c,pkA);

in (c,x:bitstring);

let y = adec(x,skA) in

let (=pkA,=pkB,k:key) = checksign(y,pkB) in

out (c,senc(FAIL,k)).

note

pattern matching capabilities (cp. formal model)

# Forward Secrecy in ProVerif III

## protocol (informal)

1.  $A \rightarrow B \quad k_A$
2.  $B \rightarrow A \quad \text{enc}_{k_A}^a(\text{sig}_{k_B}([k_A, k_B, k_{AB}]))$
3.  $A \rightarrow B \quad \text{enc}_{k_{AB}}^s(\text{FAIL}).$

## protocol: Bob

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

# Forward Secrecy in ProVerif IV

protocol (informal)

1.  $A \rightarrow B \quad k_A$
2.  $B \rightarrow A \quad \text{enc}_{k_A}^a (\text{sig}_{k_B} ([k_A, k_B, k_{AB}]))$
3.  $A \rightarrow B \quad \text{enc}_{k_{AB}}^s (\text{FAIL}).$

protocol: main process  
process

```
new skA:skey;  
new skB:sskey;  
let pkA = pk(skA) in out(c,pkA);  
let pkB = spk(skB) in out(c,pkB);  
( (!clientA(pkA,skA,pkB)) | (!serverB(pkB,skB,pkA)) | phase 1; out(c, skB) )
```

# ProVerif Script: Forward Secrecy for Handshake Protocol

command line tool

```
$ proverif 2021_01_12_lecture_09/04_forward_secrecy.pv  
$ proverif -graph targetDir 2021_01_12_lecture_09/04_forward_secrecy.pv  
$ proverif -html targetDir 2021_01_12_lecture_09/04_forward_secrecy.pv
```

live demo

# Forward Secrecy and phase in ProVerif

modeling global time

abstract, divided in “phases,” e.g.:

- o. all keys secret real-life security  
How long will an RSA-key remain secret?
- 1.  $\hat{k}_A$  gets known
- 2.  $\hat{k}_B$  gets known
- 3. ...

verifiable properties

- “ if  $k_{AB}$  not used after phase o, publication of  $\hat{k}_A$  does not lead to insecurity”
- “protocol only insecure if both  $\hat{k}_A$  and  $\hat{k}_B$  are known before ...”
- ...

## issue

- Alice and Bob want to perform key exchange
- resulting key  $k$  must remain secret – **even if involved private keys get compromised**
- no possibility to encrypt  $k$  (private keys compromised eventually)
- can use PKI only for signatures: public authenticated channel

## approach

- cannot send  $k$  in exchanged messages
- can only send “parts” of  $k$
- only Alice and Bob can compute  $k$  from “parts”

## impossible?

seems impossible – *at our level of abstraction*

## consequence

treat outside of / enhance model

# 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 infeasible in  $\mathbb{Z}_p^*$

## what have we gained?

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

# Consequence

## modeling

- analyzing perfect forward secrecy requires algebra outside our model
- many extensions of analysis approaches incorporating algebraic properties

## references

- Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: [IEEE Transactions on Information Theory](#) IT-22.6 (1976), pp. 644–654
- 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
- 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](https://doi.org/10.1109/CSF.2009.17). URL: <https://doi.org/10.1109/CSF.2009.17>

# Perfect Forward Secrecy Implementation

## steps

1. generate and distribute private/public keys
2. perform Diffie Hellman key exchange
3. use obtained key  $k$  for communication
4. delete  $k$

## popular implementation



## reference

Paul Rösler, Christian Mainka, and Jörg Schwenk.  
“More is Less: On the End-to-End Security of  
Group Chats in Signal, WhatsApp, and Threema”.  
In: **2018 IEEE European Symposium on Security  
and Privacy, EuroS&P 2018, London, United  
Kingdom, April 24-26, 2018**. IEEE, 2018, pp. 415–429

Github [29, 30]. Since neither WhatsApp nor Threema provide official open source implementations, our analysis of these protocols mainly bases on the traffic that was received by unofficial protocol implementations [15, 16]. The respective messages and operations were sent by the official applications running on different devices and transmitted via the official messenger servers.

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

## Know: DY model insufficient

### modeling of knowledge

- only derivability (DY model)
- not: “knowledge about ciphertext”

### example

Alice sends to Bob:  $\text{enc}_{k_B}^a(t)$ , where  $t \in \{\text{yes}, \text{no}\}$  (with randomized encryption)

### cannot express

- attacker does not know whether Alice sent yes or no (but knows words yes, no)
- similar: voting, attacker knows parties

# Indistinguishability

## situation

- assumption: attacker does not know  $N_A$  or  $\hat{k}_B$
- Alice sends  $\text{enc}_{k_B}^a(N_A, t)$  with  $t \in \{\text{yes}, \text{no}\}$
- want to model: attacker cannot distinguish possibilities “yes” and “no”

## term level

- attacker can distinguish  $\text{enc}_{k_B}^a(\text{yes})$  and  $\text{enc}_{k_B}^a(\text{no})$
- attacker cannot distinguish  $\text{enc}_{k_B}^a(N_A, \text{yes})$  and  $\text{enc}_{k_B}^a(N_A, \text{no})$

# Adversary Knowledge

## approach

- what does the adversary “know?”
- what is the adversary’s “interface?”

## abstraction level

distinguishability of terms

- $\text{enc}_{k_A}^a(N_A, \text{no})$  and  $\text{enc}_{k_A}^a(N_A, \text{yes})$  should “look the same” for adversary
- in particular: the adversary does not “know” that the message is  $\text{enc}_{k_A}^a(N_A, \text{yes})$

## adversary actions

- adversary can perform “term operations” to distinguish messages
- apply constructors, destructors from equational theory
- result interpreted modulo  $\equiv_E$



## 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  $m'$ .

# Indistinguishability Examples

## example

- $t_1 = \text{hash}(\text{yes})$ ,  $t_2 = \text{hash}(\text{no})$ , distinguishable?
- tests:
  - $(M_1, M'_1) = (\text{hash}(\text{yes}), x)$
  - $(M_2, M'_2) = (\text{hash}(\text{no}), x)$
- application:
  - $(M_1[t_1/x], M'_1[t_1/x]) = (\text{hash}(\text{yes}), \text{hash}(\text{yes}))$
  - $(M_1[t_2/x], M'_1[t_2/x]) = (\text{hash}(\text{yes}), \text{hash}(\text{no}))$
  - $(M_2[t_1/x], M'_2[t_1/x]) = (\text{hash}(\text{no}), \text{hash}(\text{yes}))$
  - $(M_2[t_2/x], M'_2[t_2/x]) = (\text{hash}(\text{no}), \text{hash}(\text{no}))$

## consequence

- $t_1$  satisfies first test but not second
- $t_2$  satisfies second test but not first
- so,  $t_1$  and  $t_2$  distinguishable

## more examples (see also lecture notes)

- analogously:  $\text{enc}_{k_A}^a(\text{yes})$  and  $\text{enc}_{k_A}^a(\text{no})$  distinguishable (if  $k_A$  known)
- $\text{hash}([N_A, \text{yes}])$  and  $\text{hash}([N_A, \text{no}])$  indistinguishable (if  $N_A$  unknown)
- $\text{enc}_{k_A}^a([N_A, \text{yes}])$  and  $\text{enc}_{k_A}^a([N_A, \text{no}])$  indistinguishable (if  $N_A, \hat{k}_A$  not known)

# (In)distinguishability Examples



distinguishable?

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

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

# Algorithmic Question

## decision problem

Problem: STATIC-EQUIVALENCE

Input: messages  $m$  and  $m'$ , adversary knowledge  $I$

Question: are  $m$  and  $m'$   $I$ -distinguishable?

## result

polynomial-time decidable for convergent subterm theories

## reference

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

## Exercise

### Task (indistinguishability)

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

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

# Strong Secrecy

example protocol

1.  $A \rightarrow B \quad N_A$
2.  $B \rightarrow A \quad \text{enc}_k^s([N_A, N_B])$

yes/no situation

$A \rightarrow B \text{ enc}_{k_B}^a(N_A, t)$  with  $t \in \{\text{yes}, \text{no}\}$

secrecy (privacy) of  $[N_A, N_B]$

- DY model:  $N_B$  not derivable, so  $[N_A, N_B]$  not derivable

secure in DY model

- indistinguishability:  $[N_A, N_B]$  distinguishable from  $[N_C, N_D]$ .

insecure in SE model

secrecy (privacy) of yes/no

- DY model: yes, no derivable, hence message not “secret”

insecure in DY model

- indistinguishability:  $\mathcal{A}$  has no information about message content

secure in SE model

consequence

“no implication” between security notions

## 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  $I = \{k_{AC}, \underbrace{\text{enc}_{k_{AC}}^s(\text{yes}, N_A)}_{=:u}\}$ . Then  $t = N_A$  is  $E$ -derivable from  $I$  via  $M = \text{proj}_2(\text{dec}_{k_{AC}}^s(u))$ .

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.

# Strong Secrecy in ProVerif

ProVerif: distinguishability on process level

adversary: distinguish processes  $P_1$  and  $P_2$  by interaction

modeling

free c: channel.

free secret1: bitstring[private].

free secret2: bitstring[private].

let clientAlice (which:bool) =

if which then

out (c, secret1)

else

out (c, secret2).

process

clientAlice(choice[true,false])

keyword: choice

- consider processes
  1. clientAlice(true)
  2. clientAlice(false)
- can attacker determine which process is running?
- can attacker distinguish secret1 and secret2?

## Modeling Knowledge: Two Aspects

|                      | expresses                          | ProVerif modeling   |
|----------------------|------------------------------------|---------------------|
| DY closure           | construction / derivation of terms | query attacker FAIL |
| indistinguishability | knowledge about content of terms   | choice              |

### applications

- epistemic security properties: distinguish CDU vote from SPD vote
- strategic security properties (see, e.g., contract signing, voting)

strong secrecy depends on used encryption

- 2021\_01\_19\_lecture\_10/05\_strong\_secrecy.pv  
example from slides
- 2021\_01\_19\_lecture\_10/06\_strong\_secrecy\_deterministic\_encryption.pv  
strong secrecy analysis with deterministic encryption
- 2021\_01\_19\_lecture\_10/07\_strong\_secrecy\_randomized\_encryption.pv  
strong secrecy analysis with randomized encryption

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# “Weak” Secrets

## “Secrets”

### modeling

- nonces
- random, “long enough”
- not derivable

### reality

- often “easy to remember”
- low entropy
- “dictionary attacks” possible

# Popular Passwords



## passwords

1. password
2. 123456
3. 12345678
4. 1234
5. qwerty
6. 12345
7. dragon
8. pussy
9. baseball
10. football
11. letmein
12. monkey
13. 696969
14. abc123
15. mustang
16. michael
17. shadow
18. master
19. jennifer
20. 111111
21. 2000
22. jordan
23. superman
24. harley
25. 1234567

## PINs

1. 1234
2. 0000
3. 2580
4. 1111
5. 5555
6. 5683
7. 0852
8. 2222
9. 1212
10. 1998
11. 6969
12. 1379
13. 1997
14. 2468
15. 9999
16. 7777
17. 1996
18. 2011
19. 3333
20. 1999
21. 8888
22. 1995
23. 2525
24. 1590
25. 1235

# “Weakly Secret Messages”

## scenario: electronic voting

- few candidates (german election 2017:  
42 parties)
- small “plaintext space”

## attack scenario

- attacker knows ciphertext
- knows 42 possible plaintexts
- full search!

# Election Protocol and “Weak Secrets” in ProVerif

```
protocol (docs/ex_weaksecret.pv)
  free c: channel.
  type skey.
  type pkey.
  fun pk(skey): pkey.
  fun aenc(bitstring, pkey): bitstring.
  reduc forall m: bitstring, k: skey; adec(aenc(m,pk(k)),k) = m.
  free v: bitstring [private].
  weaksecret v.
  let V(pkA:pkey) = out(c, aenc(v, pkA)).
  let A(skA:skey) = in(c,x:bitstring); let v' = adec(x, skA) in o.
process
  new skA: skey;
  let pkA = pk(skA) in
    out (c,pkA);
    ! (V(pkA) | A(skA))
```

## situation

- v not derivable
- but: v can be guessed

# ProVerif Script: Weak Secrecy

command line tool

```
$ proverif 2021_01_19_lecture_10/08_weak_secrecy.pv  
$ proverif -graph targetDir 2021_01_19_lecture_10/08_weak_secrecy.pv  
$ proverif -html targetDir 2021_01_19_lecture_10/08_weak_secrecy.pv
```

live demo

# Strong Secrecy and Weak Secrecy I

## strong secrecy motivation

- indistinguishability: adversary performs term operations to find different behavior
- assumes: adversary only wants to distinguish  $t_1$  from  $t_2$
- models: adversary “cannot get any knowledge about the message”

## weak secrets motivation

- “guessing attacks:” adversary has “candidate”  $c$  for secret  $s$  (of type  $t$ )
- adversary interacts with protocol to confirm/refute claim  $s = c$
- models: adversary has “prior knowledge” about message

## similarities, differences?

- difference: two defined processes in strong secrecy case
- similar: “comparison” of terms:  $t_1$  vs.  $t_2$ ,  $c$  vs.  $s$
- similar: adversary interacts with protocol for “comparison”

## Strong Secrecy and Weak Secrecy II

equivalent to weak secrecy

$$P \mid \text{phase 1; out}(chan, s) \approx P \mid \text{phase 1; new } s' : t; \text{out}(chan, s')$$

models

- process  $P$  uses secret  $s$
- adversary interacts with process  $P$
- after  $P$  ends:
  - lhs:  $s$  revealed
  - rhs: unrelated value (of correct type) revealed
- adversary task: determine whether we are in lhs or rhs process
- **important:** offline attack: adversary cannot interact with  $P$  anymore
- intuitively: “can the adversary get any information during process  $P$ ”?

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Security Beyond Secrecy

## security up to now: secrecy

- theory
  - INSECURE: derivability of FAIL
  - Rusinowitch-Turuani: only secrecy

- ProVerif
  - query attacker(term): DY-derivation secrecy
  - variants: strong / forward secrecy, weak secrets

## Needham-Schroeder analysis

- key purpose: authentication
- “reduction” to secrecy  $\rightsquigarrow$  exercise: unnatural modelling in ProVerif

## need: different security properties

- theory: similar to secrecy (reachability)
- can use similar algorithms

## realistic protocol

combination of security properties

# Correspondence Properties

## important class of properties: correspondence

- “if event  $e$  happens, then event  $e'$  happened before”
- required: definition of events in protocol

## possible events

- $S$  generates key  $k$  for Alice and Bob  $\text{gen}(S, A, B, k)$
- Alice accepts  $k$  for communication with Bob  $\text{accept}(A, B, k)$

## security property

- if Alice accepts  $k$ , then  $k$  has been generated by  $S$ .
- event  $\text{accept}(A, B, k)$  is always preceded by event  $\text{gen}(S, A, B, k)$

# Events in ProVerif: Handshake-Protocol I

## declarations

```
event acceptsClient(key).  
event acceptsServer(key,pkey).  
event termClient(key,pkey).  
event termServer(key).
```

## events

- normal instructions in protocol
- may have parameters
- strongly typed

## 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  
            event acceptsClient(k);  
            out (c,senc(FAIL,k));  
            event termClient(k,pkA).
```

# Events in ProVerif: Handshake-Protocol II

## server (Bob)

```
let serverB(pkB:spkey,skB:sskey)=  
    in c,pkX:pkey  
    new k:key  
    event acceptsServer(k,pkX);  
    out c,aenc(sign((pkB,k),skB),pkX)  
    in c,x:bitstring  
    let z=sdec(x,k) in  
        if pkX=pkA then event termServer(k).
```

## specification

- event  $e_1(\dots) \implies e_2(\dots)$ :
  - before  $e_1$ , event  $e_2$  must have happened
  - quantifiers: lhs  $\forall$ , rhs  $\exists$
- inj-event:
  - injective function  $e_1$ -events into  $e_2$ -events

## security property

```
query attacker (FAIL).  
query x:key, y:pkey; event(termClient(x,y))=>event(acceptsServer(x,y)).  
query x:key; inj-event(termServer(x))=>inj-event(acceptsClient(x)).
```

recall earlier question: decryption statement now meaningful  
question: why not always inj-event?

limited event semantics

event  $e_1(\dots) \Rightarrow e_2(\dots)$

- requirement: every  $e_1$  is **preceded** by some  $e_2$
- why no analogous “followed by” requirements?

in general

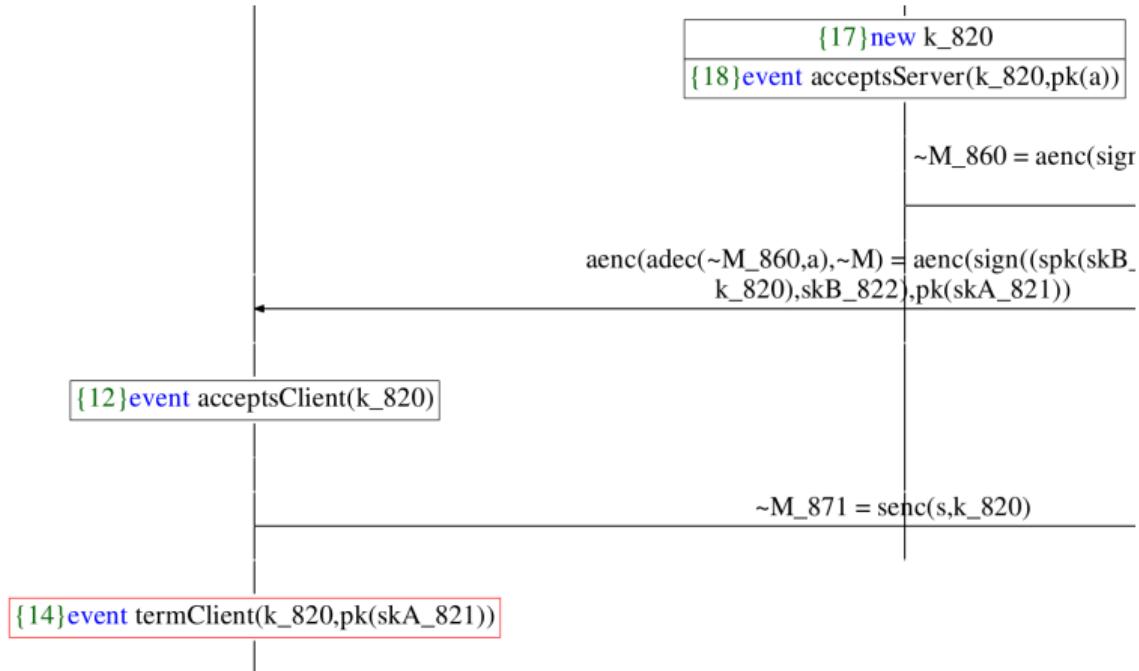
consider only “prefix-closed” properties

## ProVerif Scripts: Handshake with Events

2021\_01\_19\_lecture\_10/09\_original\_handshake\_with\_events.pv  
original (insecure) handshake protocol modelled with events

2021\_01\_19\_lecture\_10/10\_fixed\_handshake\_with\_events.pv  
fixed handshake protocol modelled with events

# ProVerif Analysis with Events



query x:key,y:pkey; event(termClient(x,y))==>event(acceptsServer(x,y)).

# 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 query 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.

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

recall

UNBOUNDED-INSECURE is undecidable

consequence for ProVerif

there are (infinitely many) protocols  $P$ , for which  
ProVerif

1. ~~terminates/secure~~, though  $P$  is not secure, or
2. ~~terminates/insecure~~, though  $P$  is secure, or
3. does not terminate, or
4. returns unknown.

examples?

- for which protocols does this happen?
- how can we avoid this?
  - re-write protocols automatically to ensure decision
  - clearly not possible for every protocol

## Example for Incomplete Analysis

```
protocol (ProVerif)
  free c:channel.
  process
    new k : key;
    out (c, senc(senc(FAIL,k),k));
    in (c, x:bitstring);
    out (c, sdec(x,k))
```

protocol secure?

question

why does ProVerif treat both protocols identically?

“almost equivalent” for ProVerif

```
free c:channel.
process
  new k : key;
  out (c, senc(senc(FAIL,k),k));
  ! (
    in (c, x:bitstring);
    out (c, sdec(x,k))
  )
protocol secure?
```

# Logic Modeling Outline

facts

- $d(\hat{k}_C)$
- $d(\{0, 1\})$
- ...

DY deductions

- $d(\text{enc}_{k_C}^a(x)) \wedge d(\hat{k}_C) \rightarrow d(x)$
- $d(x) \wedge d(y) \rightarrow d([x, y])$
- $d(x) \rightarrow d(\text{hash}(x))$
- ...

protocol deductions

- $d(\text{enc}_{k_B}^a([A, x])) \rightarrow d(\text{enc}_{k_A}^a([B, x]))$
- $d(\text{enc}_{k_A}^a([B, x, y])) \rightarrow (\text{enc}_{k_B}^a(y))$
- ...

Horn clauses

$$(x_1 \wedge x_2 \wedge \cdots \wedge x_n \rightarrow y) \leftrightarrow (\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee y)$$

target clause

$$\neg d(\text{FAIL})$$

# ProVerif Script: Incompleteness Example

command line tool

```
$ proverif 2021_01_19_lecture_10/11_incomplete.pv
$ proverif -graph targetDir 2021_01_19_lecture_10/11_incomplete.pv
$ proverif -html targetDir 2021_01_19_lecture_10/11_incomplete.pv
```

live demo

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# ProVerif Summary

## seen

- ProVerif can analyse examples treated so far in lecture
- analysis of unbounded sessions “possible in practice”
- limitations: seen here only in contrived examples, also occur for complex protocols/properties (e.g., voting)
- ProVerif features beyond formal model:
  - strong/weak secrecy, events

## caveats

- protocols can be “secure” for trivial reasons:  
 $\text{event}(a) ==> \text{event}(b)$   
is satisfied if  $a$  never happens
- also add “liveness” tests to protocol

## 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?

# Overview

## Part I: Crypto Protocols

Foundations

Cryptography

An Example and an Attack

More Examples

Formal Protocol Model

Motivation and Requirements

Messages: Formal Terms

Message Construction: Dolev-Yao Closure

Algorithm: Computing the Dolev-Yao Closure

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

DAGs

Short Attacks

NP hardness

Automatic Analysis: Undecidability

Arbitrarily Many Sessions

Incomplete Algorithms

Automatic Analysis in Practice: ProVerif

Hello World and simple Examples

Equational Theories

Randomized Encryption

Typing

Syntax: Pi-Calculus

Forward Secrecy

# Crypto Protocols Summary

covered in lecture

1. theoretical foundations
  - terms and cryptographic primitives (DY, tests)
  - protocol model and security (instances, execution order, derivability of FAIL)
  - results
    - insecurity is NP-complete [RTo3]
    - “parallel analysis” necessary and undecidable
2. approaches to undecidability: incomplete algorithms
  - abstraction, Horn approach
3. tool: ProVerif
  - examples
  - strong secrecy, indistinguishability
4. complex examples
  - ~~voting protocols: Norway protocol, FOO92~~
  - BitCoin

## Part II: Information Flow

---

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

IP-Security

Motivation and Definition

Automatic Verification

TA-Security

Motivation and Definition

## lecture up to now

- attacker model: network attacker
  - models attacks on communication
  - protection: cryptography
- protection at network level

## alternative: internal point of view

attacks “inside” one system

- buffer overflows
- format strings
- RPC vulnerabilities
- malware
- covert channels

also need protection at system/application level

# Level of Abstraction

## cryptographic protocols

- high level of abstraction with respect to cryptography
  - term model
  - idealized security properties
- low level of abstraction with respect to processing
  - structure of messages modeled precisely
  - pattern-matching steps fixed completely

## information-flow modeling

- scope: all system components
- basic model: FSMs
- high/low level of abstraction depending on semantics of states
- fewer details in model, more modeling work

# Information-Flow in the News

recent high-profile security issues

- Meltdown
- Spectre

(one of the) core issue(s)

information leakage via timing

reference

Richard J. Lipton and Kenneth W. Regan. [Timing Leaks Everything](#). 2018. URL:  
<https://rjlipton.wordpress.com/2018/01/12/timing-leaks-everything>

# Background: Speculative Execution

recall

microprocessor design: pipelining

**reasons why code is not executed**

- unauthorized memory access
- branching in “unexpected” direction

**speculative execution**

- execute code anyway
- roll-back if code not to be executed (backtrack)

# Attack Outline

## attack

- attacker wants to learn value  $b$  at location  $x$  of memory map  $K$
  - creates array  $A$  of objects  $Q$  with width equal to cache page size
  - array only **created**, not read or initialized
- content of  $A$  not in cache

```
object Q;      //loaded into chip memory
byte b = 0;
while (b == 0) {
    b = K[x];    //violates privilege---so raises an exception
}
Q = A[b];      //should not be executed but usually is

//continue process after subprocess dies or exception is caught:
int T[256];
for (int i = 0; i < 256; i++) {
    T[i] = the time to find A[i];
}
if T has a clear minimum T[i] output i, else output 0.
```

## cases

- $b \neq 0$  while-loop exits,  $A[b]$  cached → accessing  $A[b]$  faster
- $b = 0$  race condition handling (possibly no fetch)

## reference

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

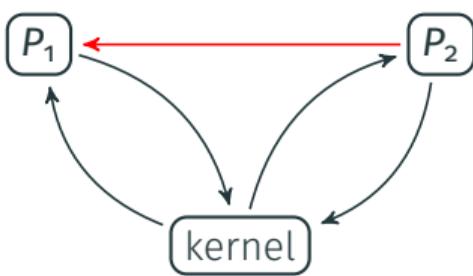
## highlights

- operating systems: Linux, Windows
- Docker
- Intel: read speed 503 KB/sec
- only “toy examples” for AMD, ARM

# Meltdown as Information Flow Issue

security policy

process isolation



communicating processes

- two user processes and kernel
- both processes may communicate with kernel
- communication between processes forbidden

meltdown

- allows direct communication between  $P_1$  and  $P_2$ : system does not respect security policy
- uses covert channel: timing information

# Side Channel Attack: Project System Bus Radio

## approach

- electronic systems emit electromagnetic radiation
- approach: choose processor workload so that radiation is AM signal (amplitude modulation, “Mittelwelle”)



## references

- <https://github.com/fulldecent/system-bus-radio>
- 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>

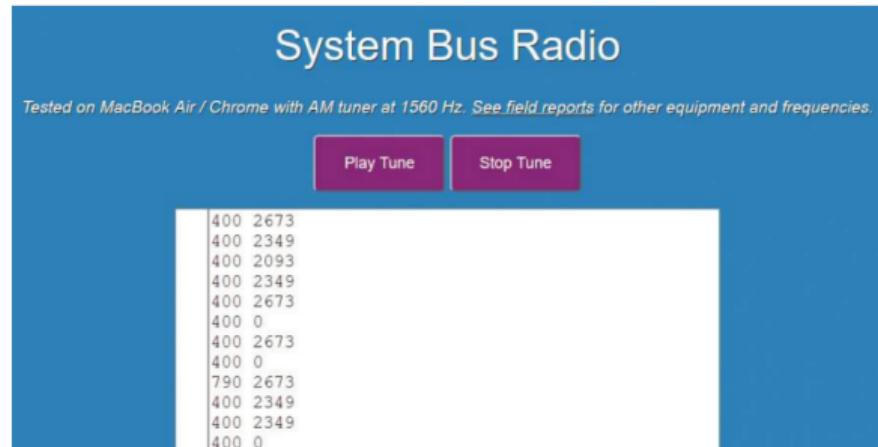
# System Bus Radio in Practice

Kurztest | PC sendet per Mittelwelle

c't 5/2018 S. 48

## PC sendet Mittelwelle

Die freie Software System Bus Radio verwandelt einen PC in einen Mittelwellensender – per JavaScript im Browser, ohne weitere Hardware.



System Bus Radio demonstriert eine Sicherheitslücke, die Hacker nutzen könnten, um Daten aus einem PC völlig ohne Netzwerkverbindungen abzugreifen. Dabei geht es um elektromagnetische Abstrahlungen, wel-

# Overview

Part II: Information Flow

- Examples
- Introduction and Motivation
- P-Security
  - Motivation and Definition

- Automatic Verification
- IP-Security
  - Motivation and Definition
  - Automatic Verification
- TA-Security
  - Motivation and Definition

## information security

crucial aspect: protection against unauthorized access or manipulation

## noninterference

- introduced by Goguen and Meseguer [GM82]
- general approach to capture security
- information flows and covert channels
- confidentiality and integrity
- goal: detect undesired information flows

# Security Policies

scenarios: different “security levels” on single system

- different processes running on the same system,
- different users interacting with the same system,
- different tabs in a browser

security policies

- policies: govern “what may be done” with information
- can be arbitrarily complex (see later)
- suffices for start: H/L policy

H high-security data (and users), must be protected

L low-security data (and users), considered public

## variations of noninterference

- classical: **transitive** noninterference
- policy generalizations: **intransitive** noninterference
- system generalizations: **dynamic** noninterference
- timing assumptions: (a)synchronous systems

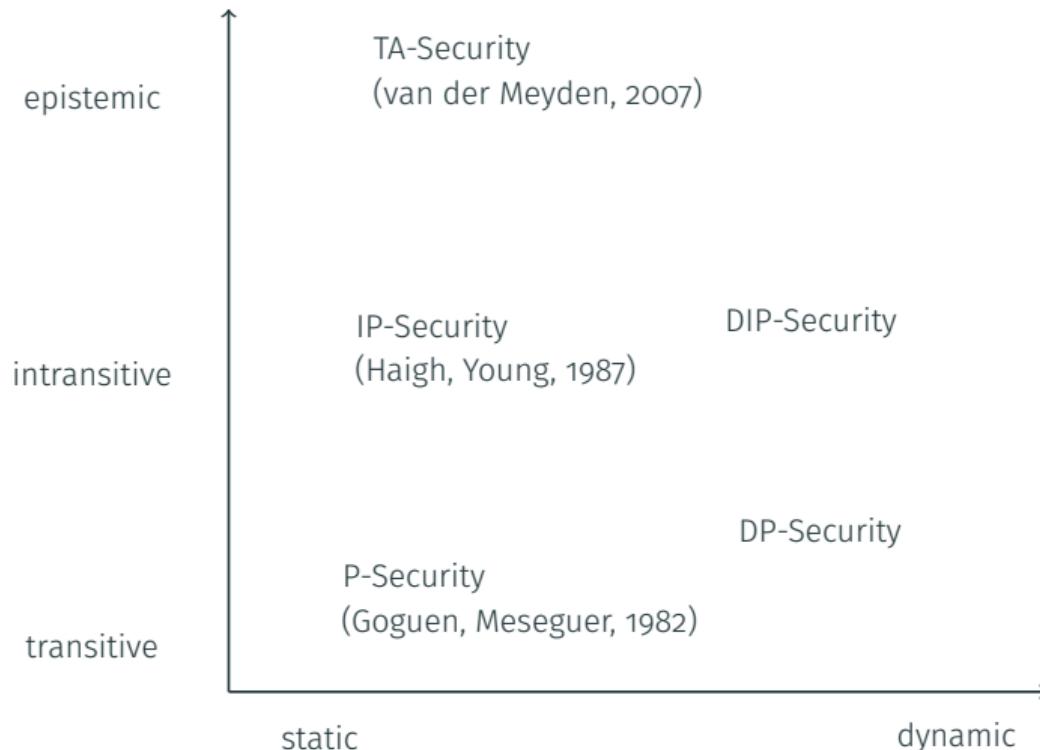
## aspects of noninterference

- definitions and relationships
- characterizations
- verification algorithms and complexity results

as usual: no “one-size-fits-all” approach

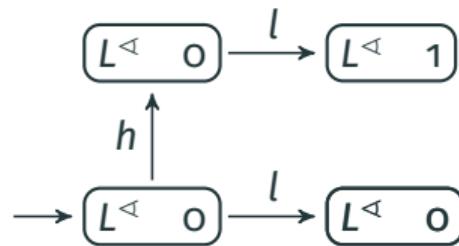
- choice of “correct” definition depends on situation
- covered definitions share basic structure
- similarities lead to common algorithmic approach

# Non-Interference Notions



# Information-Flow Example

system



- $L^<$  : output to  $L$  in state
- users  $H$  and  $L$  perform actions  $h, l$
- goal:  $L$  must not learn anything about which actions  $H$  performs

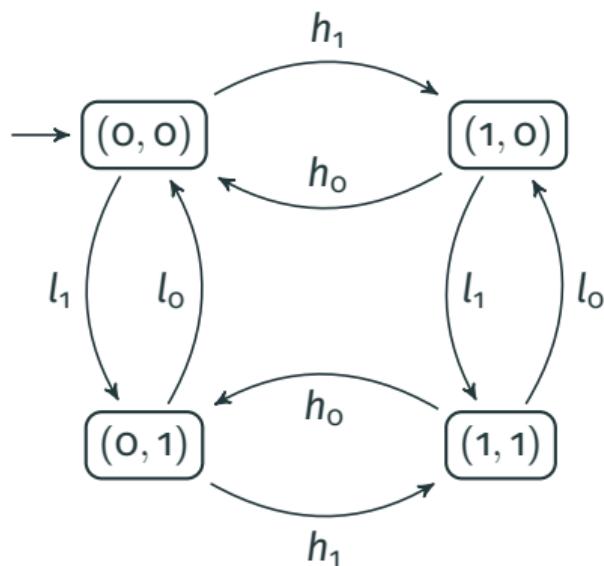
analysis

system secure?

- intuitively?
- formally?

# Information-Flow Example

system



specification

- output to  $L$ : second component of pairs
- users  $H, L$  perform actions  $h_0, h_1, l_0, l_1$
- goal:  $L$  must not learn anything about  $H$ 's actions

analysis

system secure?

- intuitively?
- formally?

# Noninterference: Formal Model

## systems

- finite automata, actions change states
- agents, domains: users of system
- $\text{obs}_L(s)$ : observation of agent  $L$  in state  $s$

## policy

→ indicates allowed information flow:

- $L \rightarrow H$ : information may flow from  $L$  to  $H$
- $H \not\rightarrow L$ : **not** from  $H$  to  $L$

$H, L$ : agents (users) or processes in a system

## central question

what does “information flows from  $H$  to  $L$ ” mean?

## noninterference

formalize this!

# Noninterference: System Model

system: tuple  $(S, s_0, A, \text{step}, D, O, \text{obs}, \text{dom})$  with

- $S$  set of states
- $s_0 \in S$  initial state
- $A$  set of actions
- $\text{step}: S \times A \rightarrow S$  deterministic  
step function
- $D$  set of security domains (agents)
- $O$  set of possible observations
- $\text{obs}: S \times D \rightarrow O$  observation function
- $\text{dom}: A \rightarrow D$  domain function

notation

- $s \in S, \alpha \in A^*$ , then  $s \cdot \alpha$ : state obtained by “performing  $\alpha$  from  $s$ ”
  - $s \cdot \epsilon = s$
  - $s \cdot \alpha a = \text{step}(s \cdot \alpha, a)$  (for  $\alpha \in A^*, a \in A$ )
- write  $\text{obs}_u(s)$  for  $\text{obs}(s, u)$

# Security Policies (formal)

## definition (security policy)

For a set of domains  $D$ , a **security policy** is a set  $\rightarrow \subseteq D \times D$ .

## properties

- $\rightarrow$  is usually reflexive
- $\rightarrow$  is often transitive (why?)

## reference

examples from 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>

## Recall: Indistinguishability

### crypto protocols

- secrecy on term level
- indistinguishability: **tests** (operations on terms)
- security:  $t_1$  and  $t_2$  indistinguishable
  - e.g.,  $t_1$  and  $t_2$  Alice's messages in voting protocol

### information-flow security

- “data:” performed actions
- indistinguishability: from observations ( $q_1 \equiv q_2$  iff  $\text{obs}_L(q_1) = \text{obs}_L(q_2)$ )
- security:  $q_1$  and  $q_2$  indistinguishable
  - if “same public data” in  $q_1$  and  $q_2$

# Information-Flow Security Approach

“required” and “achieved” indistinguishability

traces  $\alpha_1, \alpha_2 \in A^*$  should be indistinguishable if they have same “public data”

states  $s \cdot \alpha_1, s \cdot \alpha_2$  are indistinguishable, if they have same observations

security: system achieves required indistinguishabilities

- state-equivalence relation “includes” trace-equivalence relation
- traces that should be indistinguishable lead to indistinguishable states

three instances

- P-security [GM82],
- IP-security [HY87],
- TA-security [Mey07].

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

IP-Security

Motivation and Definition

Automatic Verification

TA-Security

Motivation and Definition

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

IP-Security

Motivation and Definition

Automatic Verification

TA-Security

Motivation and Definition

# Noninterference: P-Security

## overview

- simple notion of security [GM82]
- assumes  $H/L$  policy.  $L \rightarrowtail H$
- intuition: “low” users may not “see” anything that “high users” do

## definition (purge function)

$E \subseteq D$  set of domains, sequence  $\alpha \in A^*$ , policy  $\rightarrowtail$

- $\alpha|E$ : subsequence of actions  $a$  from  $\alpha$  with  $\text{dom}(a) \in E$
- $\text{purge}(\alpha, u) = \alpha| \{v \in D \mid v \rightarrowtail u\}$
- often write  $\text{purge}_u(\alpha)$ , omit  $\rightarrowtail$

## intuition

$\text{purge}(\alpha, u)$  contains actions from  $\alpha$  that  $u$  may “learn about”



## definition (P-security)

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

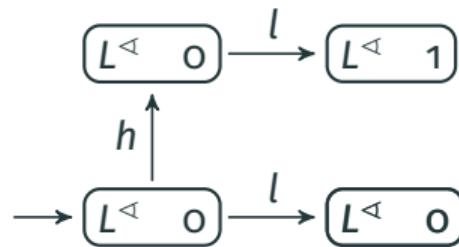
$$\text{If } \text{purge}_u(\alpha_1) = \text{purge}_u(\alpha_2), \text{ then } \text{obs}_u(s \cdot \alpha_1) = \text{obs}_u(s \cdot \alpha_2).$$

## intuition

- $\alpha_1$  and  $\alpha_2$  should “look the same” to  $u$
- performing  $\alpha_1$  or  $\alpha_2$  from  $s$  should make no difference for  $u$
- $u$  should receive the same information from the system for both sequences

# P-Security Example

system



- $L^<$  :  $\text{obs}_L$
- $h, l$ : actions of  $H, L$
- policy:  $L \rightarrowtail H$

analysis

system secure?

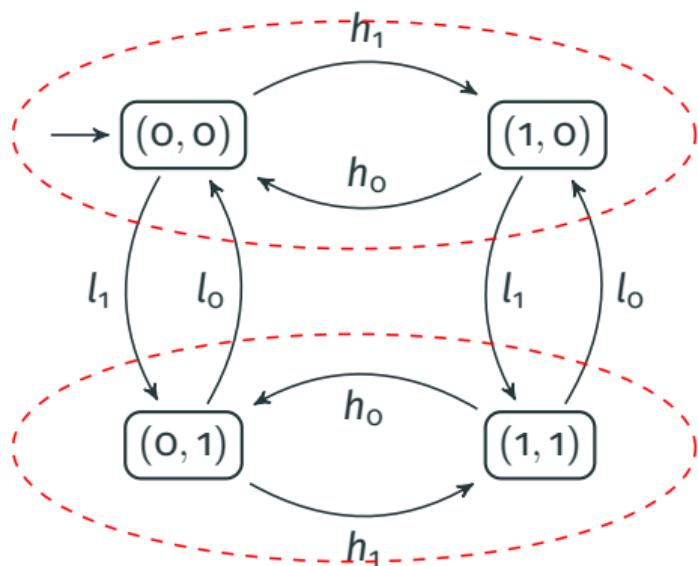
- intuitively?
- formally?

insecure

- $\alpha_1 : l$ , then  $\text{purge}_L(\alpha_1) = l$
- $\alpha_2 : hl$ , then  $\text{purge}_L(\alpha_2) = l$
- $\text{obs}_L(q_0 \cdot \alpha_1) = o \neq 1 = \text{obs}_L(q_0 \cdot \alpha_2)$



## system



## specification

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

## analysis

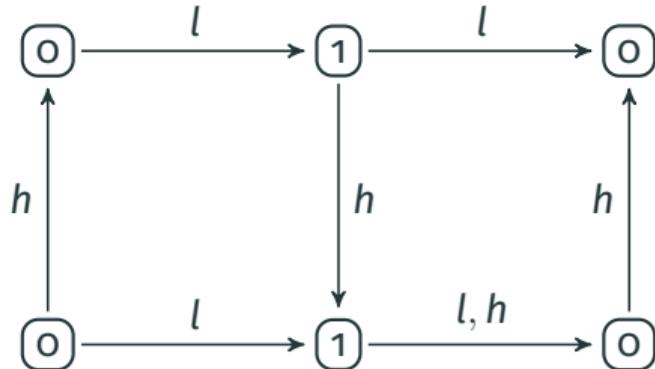
system secure?

- intuitively?
- formally?

## Exercise

### Task (P-Security Example I)

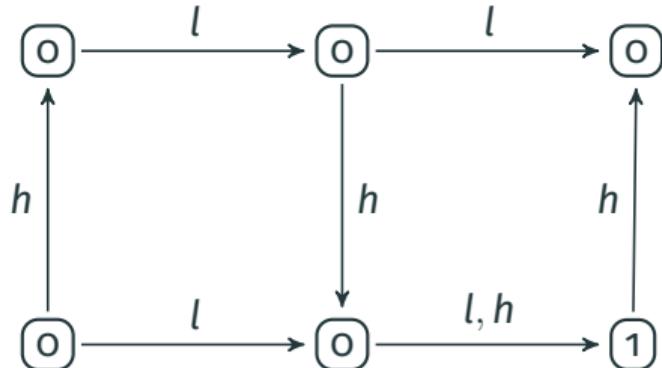
Is the following system P-secure? Justify your answer.



## Exercise

### Task (P-Security Example II)

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



## Exercise

### Task (alternative definition of P security I)

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

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

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

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 used by Ron van der Meyden.

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

IP-Security

Motivation and Definition

Automatic Verification

TA-Security

Motivation and Definition

# Proving (in-)Security

## methods

- prove insecurity: counter-example
- prove security: manual proof

## consequence

- need proof technique: short “arguments” why we should believe in system’s security
- need automatic security analysis

## comparison to protocols

- approach similar
- model of (realistic) systems: much larger!

## verifying P (later: also IP/TA-)security

- if  $\text{purge}_u(\alpha_1) = \text{purge}_u(\alpha_2)$ , then  $\text{obs}_u(s \cdot \alpha_1) = \text{obs}_u(s \cdot \alpha_2)$  (or ipurge, or ta)
- only finitely many pairs  $(s_1, s_2)$  with  $\text{obs}_u(s_1) = \text{obs}_u(s_2)$  required
- security proof needs list of all these  $(s_1, s_2)$
- infinitely many  $\alpha_1, \alpha_2$  to consider

## algorithmic approach

- complete set of pairs  $(s_1, s_2)$  with  $\text{obs}_u(s_1) = \text{obs}_u(s_2)$  required
- start with  $\{(s, s) \mid s \in S\}$
- add pairs for “suitable” sequences  $\alpha_1, \alpha_2$  until fixpoint reached



## Definition

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

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

## theorem (Rushby, [Rus92])

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

## corollary

P-Security can be verified in polynomial time.

(proof follows)



# Characterization of P-Security with Unwindings

<https://cloud.rz.uni-kiel.de/index.php/s/6k9DT475qW9NcQg>

## video content

- proof: a system is P-secure if and only if there is an unwinding
- “canonical” choice of unwindings

## study

- watch video—feedback welcome!
- video slides contained in slide set (gray background), additional material in lecture notes
- next week: discussion of content (in small groups), bring questions!

# Characterizing P-Security with Unwindings

## Theorem

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

## reference

John Rushby. [Noninterference, Transitivity, and Channel-Control Security Policies](#). Tech. rep. CSL-92-02. SRI International, Dec. 1992. URL: <http://www.csle.sri.com/papers/csle-92-2/>

## relevance

- classic result, many (more complex) generalizations
- captures “intuitive” reasons for security
- motivation: proof technique, verification

## Recall: Definition P-Unwinding

### Definition

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

$OC^P$  if  $s \sim_u t$ , then  $\text{obs}_u(s) = \text{obs}_u(t)$  output consistency

$SC^P$  if  $s \sim_u t$ , then  $s \cdot a \sim_u t \cdot a$  step consistency

$LR^P$  if  $\text{dom}(a) \not\ni u$ , then  $s \sim_u s \cdot a$  left respect

# Simplification

## notation

fix user  $u$ : write `purge` instead of  $\text{purge}_u$ ,  $\sim$  instead of  $\sim_u$ , `obs` instead of  $\text{obs}_u$

## possible because P-security “simple”:

- (proof of) unwinding for user  $u_1$  does not depend on unwinding for user  $u_2$
- P-security does not model “interaction” between users
- contrast to IP-security (see later)

## Part 1: Unwinding → P-Security overview

### Proof Structure

- assume unwinding exists
- prove key fact:
  - for all  $\alpha \in A^*$ , we have  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$ .
- with key fact and output consistency:
  - if  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , then  $\text{obs}(s\alpha_1) = \text{obs}(s\alpha_2)$ .
- this is P-Security.

## Proof of Key Fact (Part I)

Claim (Key Fact)

if  $\sim$  unwinding,  $\alpha \in A^*$ , then  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$

recall step consistency

if  $s \sim t$ , then  $s \cdot a \sim t \cdot a$

proof: induction over  $|\alpha|$

$\alpha = \epsilon$   $s \cdot \alpha = s \cdot \epsilon = s \sim s = s \cdot \epsilon = s \cdot \text{purge}(\epsilon)$ , since  $\sim$  reflexive

$\alpha \rightarrow \alpha a$  induction:  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$ , must show:  $s \cdot \alpha a \sim s \cdot \text{purge}(\alpha a)$

induction step case 1:  $\text{dom}(a) \rightarrow u$

from step consistency and  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$ :  $s \cdot \alpha a \sim s \cdot \text{purge}(\alpha)a$

since  $\text{purge}(a) = a$ :  $= s \cdot \text{purge}(\alpha)\text{purge}(a)$

since  $\text{purge}(\alpha a) = \text{purge}(\alpha)\text{purge}(a)$ :  $= s \cdot \text{purge}(\alpha a)$

## Proof of Key Fact (Part II)

Claim (Key Fact)

if  $\sim$  unwinding,  $\alpha \in A^*$ , then  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$

recall left respect

if  $\text{dom}(a) \not\rightarrow u$ , then  $s \sim s \cdot a$

proof: induction over  $|\alpha|$

$\alpha = \epsilon$   $s \cdot \alpha = s \cdot \epsilon = s \sim s = s \cdot \epsilon = s \cdot \text{purge}(\epsilon)$ , since  $\sim$  reflexive

$\alpha \rightarrow \alpha a$  induction:  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$ , must show:  $s \cdot \alpha a \sim s \cdot \text{purge}(\alpha a)$

induction step case 2:  $\text{dom}(a) \not\rightarrow u$

from left respect  $s \cdot \alpha \sim s \cdot \alpha a$

induction, transitivity  $s \cdot \alpha a \sim s \cdot \text{purge}(\alpha)$

since  $\text{dom}(a) \not\rightarrow u$   $\text{purge}(\alpha a) = \text{purge}(\alpha)$

so  $s \cdot \alpha a \sim s \cdot \text{purge}(\alpha a)$

proof of key fact complete

next: use this to show security

# Proof of Security with Key Fact

## Claim

If there is an unwinding, system is P-secure: if  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , then  $\text{obs}(s \cdot \alpha_1) = \text{obs}(s \cdot \alpha_2)$

## Key Fact

If  $\sim$  unwinding,  $\alpha \in A^*$ , then  $s \cdot \alpha \sim s \cdot \text{purge}(\alpha)$

## proof

- choose  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$
- $s \cdot \alpha_1 \sim s \cdot \text{purge}(\alpha_1) = s \cdot \text{purge}(\alpha_2) \sim s \cdot \alpha_2$
- output consistency:  $\text{obs}(s \cdot \alpha_1) = \text{obs}(s \cdot \alpha_2)$

recall output consistency

if  $s \sim t$ , then  $\text{obs}(s) = \text{obs}(t)$

## completes proof of first direction

If there is an unwinding, system is P-secure.

## Part 2: P-Security → Unwinding overview

### Proof Structure

- assume system secure:  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$  implies  $\text{obs}(s \cdot \alpha_1) = \text{obs}(s \cdot \alpha_2)$
- need to define equivalence relation  $\sim$  (for agent  $u$ ) that satisfies:
  - $\text{OC}^P$  if  $s \sim t$ , then  $\text{obs}(s) = \text{obs}(t)$  output consistency
  - $\text{SC}^P$  if  $s \sim t$ , then  $s \cdot a \sim t \cdot a$  step consistency
  - $\text{LR}^P$  if  $\text{dom}(a) \not\rightarrow u$ , then  $s \sim s \cdot a$  left respect
- candidate:  $s \sim t$ , if “equivalent actions lead to indistinguishably states”

### Choice of $\sim$

$s \sim t$  iff for all  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , we have  $\text{obs}(s \cdot \alpha_1) = \text{obs}(t \cdot \alpha_2)$

# Proof of Unwinding Properties (Part 1)

## Relation

$s \sim t$  iff for all  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , we have  $\text{obs}(s \cdot \alpha_1) = \text{obs}(t \cdot \alpha_2)$

## Claim

if system secure,  $\sim$  is an unwinding

## proof

- $\sim$  is an equivalence relation
  - $\sim$  reflexive: due to  $P$ -security, if  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , then  $\text{obs}(s \cdot \alpha_1) = \text{obs}(s \cdot \alpha_2)$ . So,  $s \sim s$ .
  - symmetry, transitivity: trivial
- output consistency: let  $s \sim t$ , choose  $\alpha_1 = \alpha_2 = \epsilon$ :  $\text{obs}(s) = \text{obs}(t)$

## Proof of Unwinding Properties (Part 2)

### Relation

$s \sim t$  iff for all  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , we have  $\text{obs}(s \cdot \alpha_1) = \text{obs}(t \cdot \alpha_2)$

proof: if system secure,  $\sim$  is an unwinding (here: left respect)

- choose  $a$  with  $\text{dom}(a) \not\rightarrow u$ , need to show:  $s \sim s \cdot a$
- choose  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , need to show:  
 $\text{obs}(s \cdot \alpha_1) = \text{obs}(s \cdot a\alpha_2)$
- since  $\text{dom}(a) \not\rightarrow u$ , we have  $\text{purge}(a\alpha_2) = \text{purge}(\alpha_2)$
- so:  
$$\begin{aligned}\text{obs}(s \cdot \alpha_1) &= \text{obs}(s \cdot \alpha_2) && (\text{since } s \sim s \text{ and } \text{purge}(\alpha_1) = \text{purge}(\alpha_2)) \\ &= \text{obs}(s \cdot a\alpha_2) && (\text{since } s \sim s \text{ and } \text{purge}(\alpha_2) = \text{purge}(a\alpha_2))\end{aligned}$$

# Proof of Unwinding Properties (Part 3)

## Relation

$s \sim t$  iff for all  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , we have  $\text{obs}(s \cdot \alpha_1) = \text{obs}(t \cdot \alpha_2)$

proof: if system secure,  $\sim$  is an unwinding (here: step consistency)

- choose  $s, t$  with  $s \sim t$ ,  $a \in A$ , show:  $s \cdot a \sim t \cdot a$ .
- choose  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , show:  $\text{obs}(s \cdot a \text{purge}(\alpha_1)) = \text{obs}(t \cdot a \text{purge}(\alpha_2))$ .
- since  $s \sim t$  and definition of  $\sim$ : enough to show that  $\text{purge}(a \text{purge}(\alpha_1)) = \text{purge}(a \text{purge}(\alpha_2))$ .
- this follows:

$$\begin{aligned}\text{purge}(a \text{purge}(\alpha_1)) &= \text{purge}(a) \text{purge}(\alpha_1) \\ &= \text{purge}(a) \text{purge}(\alpha_2) = \text{purge}(a \text{purge}(\alpha_2)).\end{aligned}$$

completes proof of second direction

If system is secure, there is an unwinding relation.

# Conclusion and Outlook

## Result

P-Security is completely characterized by unwindings

## Consequences

- an unwinding is a formal proof for P-security of a system
- unwindings (or bisimulations) are popular proof techniques for various security notions

## Application: Automatic Analysis

How do we determine whether a system has an unwinding?

- “canonical unwinding”:  
 $s \sim t$  iff for all  $\alpha_1, \alpha_2$  with  $\text{purge}(\alpha_1) = \text{purge}(\alpha_2)$ , we have  $\text{obs}(s \cdot \alpha_1) = \text{obs}(t \cdot \alpha_2)$
- how is computing this simpler than deciding P-security by the original definition?

# Video Lecture: Feedback wanted



## questions

- audio/video quality?
- proof presentation as screenshots, or “live writing?”
- better as video or “live Zoom session?”
- any suggestions?

## feedback crucial

- your perspective very different from mine!
- constructive criticism always welcome
- review after week 6!

## remember

- we're all still learning this
- new tools, concepts
- big playground :-)

# Plan for Review Sessions

## purpose, timing

- used after self-study material (videos)
- purpose: discussions / questions about content (usually proofs)
  - mainly: your questions
  - some: review questions
  - **no prepared material**, that's the point!
- length/time: partial next session
  - synchronize schedule with last course iteration

this time: only one group

probably  $\approx$  half of next week's session

# Unwinding Examples

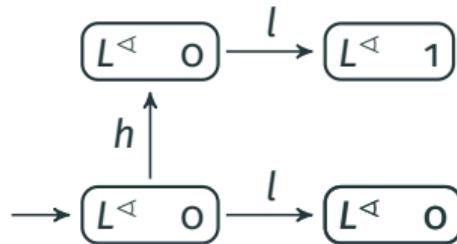
conditions

OC  $s \sim_u t$ , then  $\text{obs}_u(s) = \text{obs}_u(t)$

SC  $s \sim_u t$ , then  $s \cdot a \sim_u t \cdot a$

LR  $\text{dom}(a) \not\rightarrow u$ , then  $s \sim_u s \cdot a$

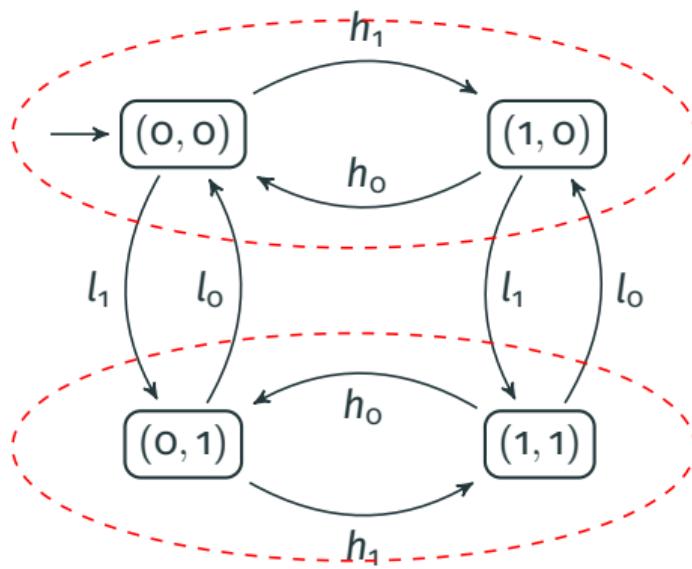
system 1



insecure

- $\alpha_1 = l$
- $\alpha_2 = hl$

system 2



## Exercise

### Task (uniqueness of unwindings)

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

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

# Algorithm for P-Security

seen

P-security is characterized by unwindings

algorithmic approach

check whether unwinding exists, accept if unwinding found.

issues?

- what are “candidates” for unwindings?
- how many equivalence relations on a set with  $|S|$  elements?
- candidate given by proof:

$$s \sim_u t \text{ iff } \forall \alpha_1, \alpha_2 \text{ with } \text{purge}(\alpha_1) = \text{purge}(\alpha_2) : \text{obs}_u(s \cdot \alpha_1) = \text{obs}_u(t \cdot \alpha_2)$$

- difficult to construct algorithmically!



## lemma

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

Input:  $(S, A, \text{step}, D, \text{dom})$

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

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

## IP-Security

Motivation and Definition

Automatic Verification

## TA-Security

Motivation and Definition

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

## IP-Security

Motivation and Definition

Automatic Verification

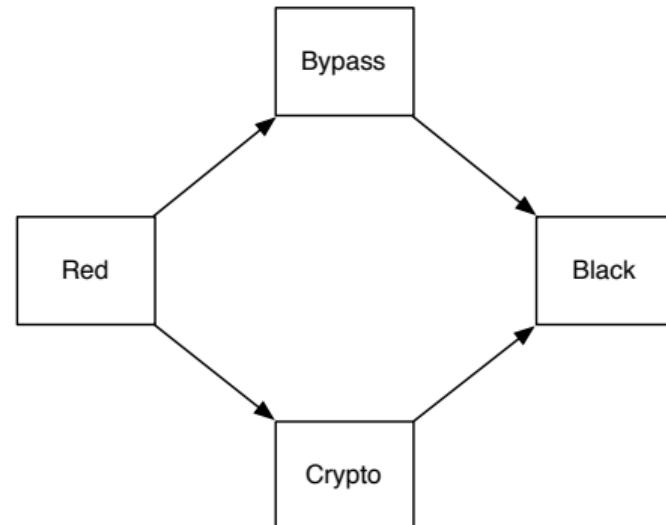
## TA-Security

Motivation and Definition

# Intransitive Noninterference

## P-Security

- reasonable definition of security
- assumes that policies are transitive
- intransitive policies occur in more complex scenarios



# Intransitive Noninterference

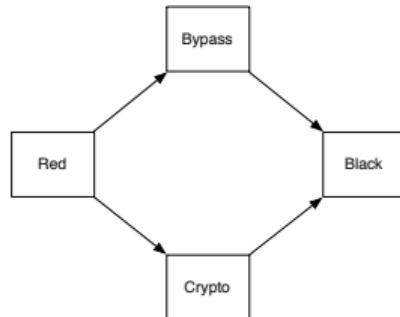
## issue

- information may flow from **Red** to **Black**, but must pass **Crypto** or **Bypass**
- all-or-nothing approach of P-security does not suffice

## goals for definition

- **Red**'s actions may have impact on **Black**'s view
- but **Black** may **only** learn of these actions “via **Bypass** or **Crypto**”
- question whether **Black** may learn of action depends on what happens *after* action

## intransitive policy



## downgrading

- indirect interference, introduced in [HY87]
- standard example: trusted “downgrader”  $D$ : declassifier, encryption device, ... *small enough to be formally verified*
- **intransitive** policies:

$$H \longrightarrow D \longrightarrow L$$

- $H$ 's actions “transmitted” to  $L$  by actions of  $D$
- $L$  must not learn about  $H$ 's actions directly

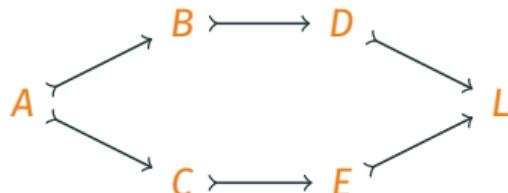
## intransitive noninterference

meaningful semantics for intransitive policies

# Intransitive Noninterference

## question

- action sequence:  $a\alpha$
- may  $L$  “learn” that  $a$  was performed?



## downgrading

transmission of actions by sequence of actions

### With each action

Agent performing action “transmits” knowledge about previous events

## step-by-step downgrading

- sequence  $abece$ : who may “know” that  $a$  occurred?
- knowledge “spreads” in each step:  $a b e c e$

# Intransitive Noninterference: IP-Security

## overview

- adaptation of P-security to intransitive case, defined in [HY87]
- replaces `purge` with `ipurge`: keeping track of “allowed interferences”

## definition (sources)

- $\text{sources}(\alpha, u)$ : agents who may interfere with  $u$  in sequence  $\alpha$
- $\text{sources}: A^* \times D \rightarrow \mathcal{P}(D)$ 
  - $\text{sources}(\epsilon, u) = \{u\}$
  - $\text{sources}(a\alpha, u)$  for  $a \in A, \alpha \in A^*$ : two cases
    1. there is  $v \in \text{sources}(\alpha, u)$  with  $\text{dom}(a) \rightarrow v$ , then
$$\text{sources}(a\alpha, u) = \text{sources}(\alpha, u) \cup \{\text{dom}(a)\}.$$
    2. otherwise:  $\text{sources}(a\alpha, u) = \text{sources}(\alpha, u).$

# Intransitive Noninterference: IP-Security

## definition (ipurge)

$\text{ipurge}: A^* \times D \rightarrow A^*$  (also:  $\text{ipurge}_u$ ) defined inductively

- $\text{ipurge}(\epsilon, u) = \epsilon$
- for  $a \in A, \alpha \in A^*$ :

$$\text{ipurge}(a\alpha, u) = \begin{cases} a\text{ipurge}(\alpha, u), & \text{if } \text{dom}(a) \in \text{sources}(a\alpha, u), \\ \text{ipurge}(\alpha, u), & \text{otherwise} \end{cases}$$

## definition (IP-security)

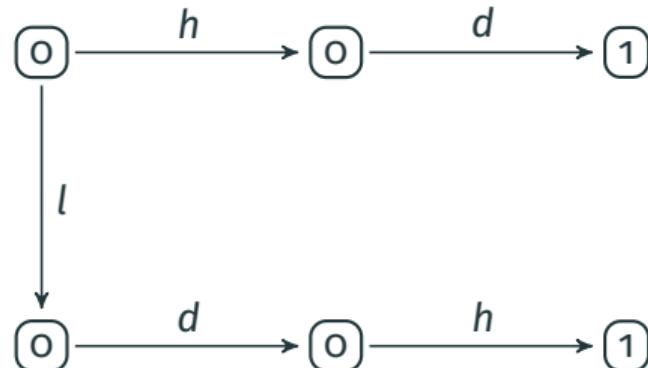
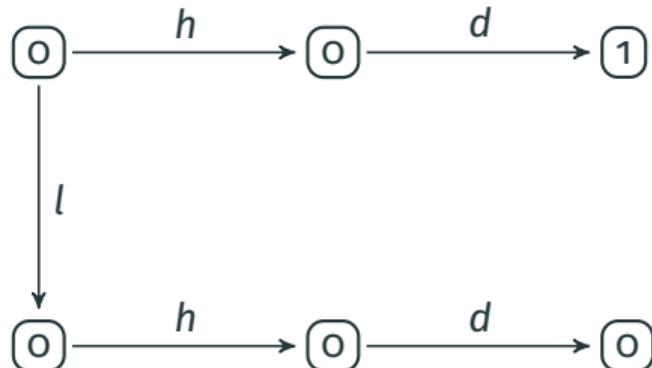
System  $(S, s_0, A, \text{step}, D, O, \text{obs}, \text{dom})$  is IP-secure with respect to a policy  $\rightarrow$ , if for all  $u \in D, s \in S, \alpha_1, \alpha_2 \in A^*$ :

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

## Exercise

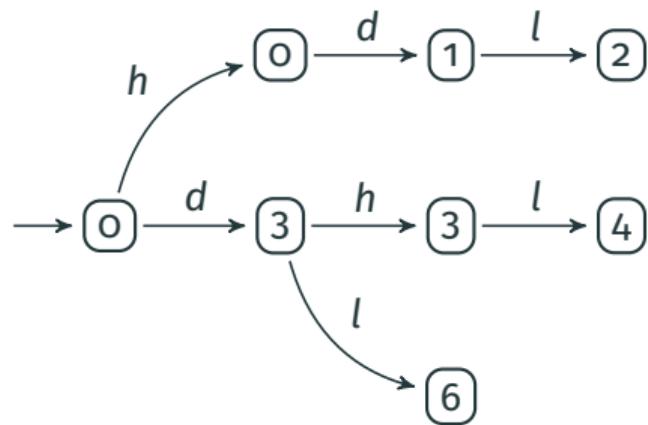
### Task (IP-Security examples)

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  $H \rightarrow D \rightarrow L$ . Additionally, assume that  $H$  and  $D$  make the same observation in each state of the system.





system



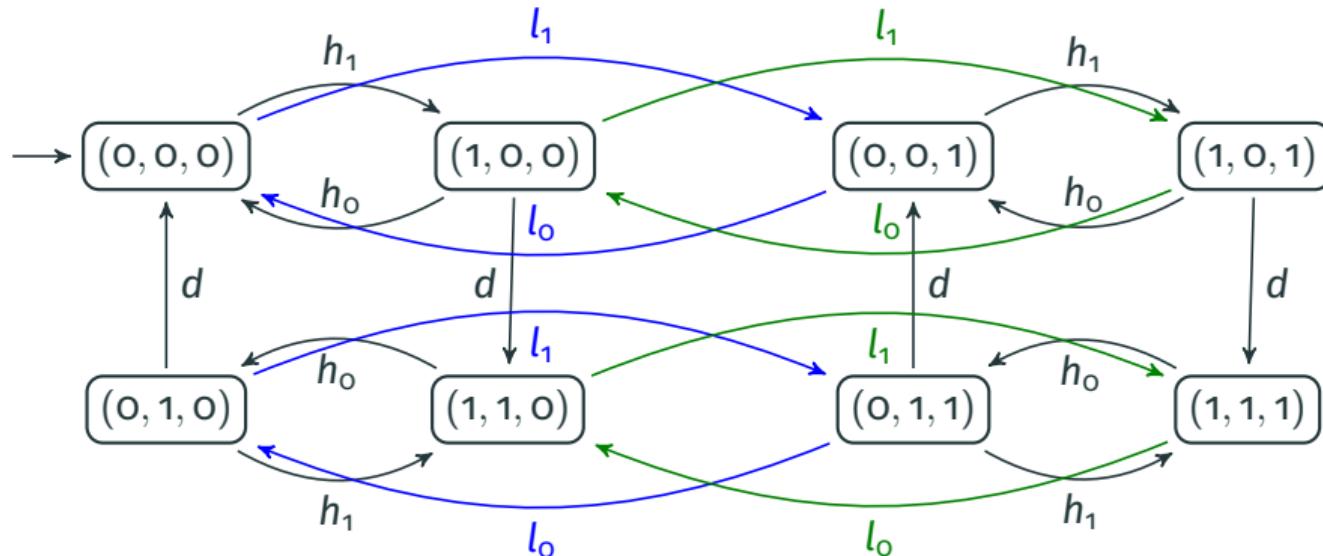
specification

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

analysis

system secure?

- intuitively?
- formally?



system

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



## 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.  $\rightarrow$ , then also IP-secure wrt.  $\rightarrow$ .

## converse?



## fact

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

## proof

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

completes proof (see exercise for transitive case).

## 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  $\rightarrow$  a policy).

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

## 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  $\rightarrow$  be a transitive policy. Show that the following are equivalent:

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

## Exercise

### Task (P-security and non-transitive policies)

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

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

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

### IP-Security

Motivation and Definition

Automatic Verification

### TA-Security

Motivation and Definition

# Unwindings for IP- (and TA-) security?

## observation

- IP (and TA) security are more “complex” than P-security
- for deciding security: must keep track of “who-knows-what”
- simple unwinding as in P-security not expected

## verification

- IP-security (and TA-security) can still be decided in polynomial time
- key: unwinding conditions “between several agents”

## reference

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

# Unwindings for IP-security

## definition

An **IP-unwinding** for a system  $(S, s_0, A, \text{step}, D, O, \text{obs}, \text{dom})$  and a policy  $\rightarrow$  is a family of equivalence relations  $(\sim_u^v)_{u,v \in D}$  on  $S$  such that

## intuition

- $u$ : observer ( $L$ )
- $v$ : potentially secret actions ( $H$ )

$\text{OC}^{IP}$  if  $s \sim_u^v t$ , then  $\text{obs}_u(s) = \text{obs}_u(t)$

$\text{SC}^{IP}$  if  $s \sim_u^v t$  and  $v \not\rightarrow \text{dom}(a)$  then  $s \cdot a \sim_u^v t \cdot a$

$\text{LR}^{IP}$  if  $v \not\rightarrow u$  and  $a \in A$  with  $\text{dom}(a) = v$  then  $s \sim_u^v s \cdot a$

## theorem [Egg+13]

A system  $M$  is IP-secure wrt.  $\rightarrow$  if and only if there is an IP-unwinding for  $M$  and  $\rightarrow$ .

## corollary

IP-security can be verified in polynomial time.

## proof

- $M$  IP-secure wrt  $\rightarrow$  iff all  $\sim_u^v$  satisfy output consistency, where:  
 $\sim_u^v$  smallest equivalence relation satisfying  $SC^{IP}$  and  $LR^{IP}$  with respect to  $\rightarrow$ .
- algorithm: immediately from unwinding, analogous to P-security

# Overview

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

## IP-Security

Motivation and Definition

Automatic Verification

## TA-Security

Motivation and Definition

## Part II: Information Flow

Examples

Introduction and Motivation

P-Security

Motivation and Definition

Automatic Verification

IP-Security

Motivation and Definition

Automatic Verification

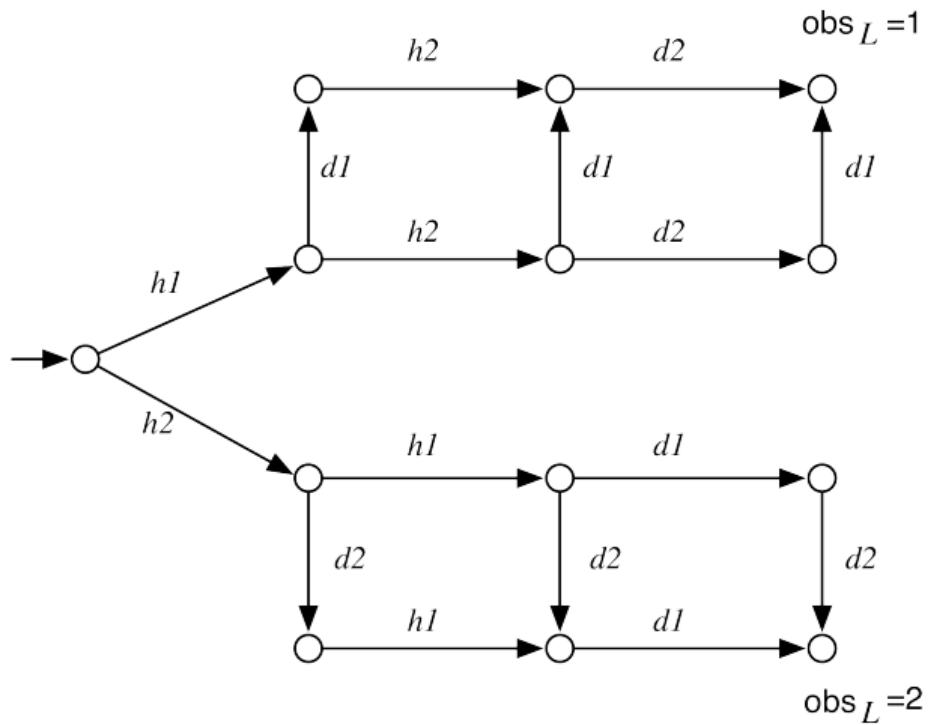
TA-Security

Motivation and Definition

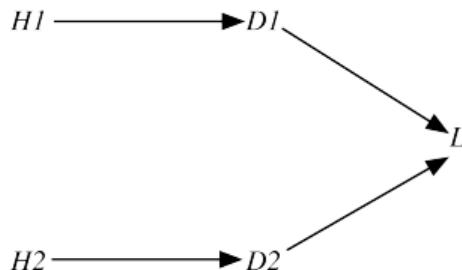
# IP-Security: Is it Enough?



system [Meyo7]



policy



analysis

system secure?

- intuitively?
- formally?

## observation

- P- and IP-security only model *which* actions an agent may “learn”
- not treated: information about order of actions

## fixing IP security

- modify definition to add order-information
- are we then sure we captured everything?

## overview

- ta-function: transmission of actions
- defines *maximal information*  $\text{ta}_u(\alpha)$  that agent  $u$  may have about run  $\alpha$
- TA-Security requirement: if  $\text{ta}_u(\alpha) = \text{ta}_u(\beta)$ , then  $\text{obs}_u(s \cdot \alpha) = \text{obs}_u(s \cdot \beta)$

## reference (definition and basic properties of TA-Security)

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

## References i

-  **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>.
-  Martin Abadi and Véronique Cortier. "Deciding knowledge in security protocols under equational theories". In: **Theoretical Computer Science** 367.1-2 (2006), pp. 2–32.
-  Bruno Blanchet. "Using Horn Clauses for Analyzing Security Protocols". In: **Cryptology and Information Security Series** 5 (2011), pp. 86–111.
-  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.
-  Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: **FOCS**. 2001, pp. 136–145.

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



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.

-  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/procceeding>.
-  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.
-  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.

-  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>.
-  Joseph A. Goguen and José Meseguer. “Security Policies and Security Models”. In: [IEEE Symposium on Security and Privacy](#). 1982, pp. 11–20.
-  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>.
-  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.

-  Richard J. Lipton and Kenneth W. Regan. **Timing Leaks Everything**. 2018. URL: <https://rjlipton.wordpress.com/2018/01/12/timing-leaks-everything>.
-  Robert Künemann, 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>.
-  Steve Kremer and Robert Künemann. “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>.
-  Detlef Kähler, Ralf Küsters, and Tomasz Truderung. “Infinite State AMC-Model Checking for Cryptographic Protocols”. In: **LICS**. IEEE Computer Society, 2007, pp. 181–192.

-  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.
-  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.
-  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>.

-  Ralf Küsters. "Simulation-Based Security with Inexhaustible Interactive Turing Machines". In: **CSFW**. IEEE Computer Society, 2006, pp. 309–320. ISBN: 0-7695-2615-2.
-  Ralf Küsters and Thomas Wilke. **Moderne Kryptographie - Eine Einführung**. Vieweg + Teubner, 2011. ISBN: 978-3-519-00509-4.
-  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](https://doi.org/10.1007/978-3-319-57048-8%5C_6).
-  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.

-  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.
-  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.
-  Jonathan K. Millen. "A Necessarily Parallel Attack". In: **In Workshop on Formal Methods and Security Protocols**. 1999.

-  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](http://dx.doi.org/10.1016/0890-5401(92)90008-4).
-  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.
-  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>.

-  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>.
-  Paul Rösler, Christian Mainka, and Jörg Schwenk. "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema". In: **2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018**. IEEE, 2018, pp. 415–429.
-  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.
-  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/>.

-  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.
-  T. J. Schaefer. "The complexity of satisfiability problems". In: **Proceedings 10th Symposium on Theory of Computing**. ACM Press, 1978, pp. 216–226.
-  Davide Sangiorgi and David Walker. **The Pi-Calculus - a theory of mobile processes**. Cambridge University Press, 2001. ISBN: 978-0-521-78177-0.
-  Mathy Vanhoef and Frank Piessens. **Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2**. 2017.
-  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>.

- 
- 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>.

## Sessions Overview

---

# Sessions Overview

## links to individual sessions

### # date

- 1 November 3, 2020
- 2 November 10, 2020
- 3 November 17, 2020
- 4 November 24, 2020
- 5 December 1, 2020
- 6 December 8, 2020
- 7 December 15, 2020

### # date

- 8 January 5, 2021
- 9 January 12, 2021
- 10 January 19, 2021
- 11 January 26, 2021
- 12 February 2, 2021
- 13 February 9, 2021