

02239 Data Security

Christian Damsgaard Jensen

Office: 322/225

Email: cdje@dtu.dk

Sebastian A. Mödersheim

Office: 321/018

Email: samo@dtu.dk

DTU Compute

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} =$$
$$\epsilon = \{2.7182818284590452353602874713526624977572470936999595749749466113934497633592468328234724281902169139949850801698436508563732816096112543352393307141890434993878964462656946165353239881303596623829086967745748272483111666872538033543291505872362902433789614685785424989152092096282925409171536436789259036001274874977299963347944126568127424509617361556354283993929893823980099104276190291891267390410656256117999342996882699842493521752934340518770784754550316401020253100747211019653142471569223832928350514879121688157460535940812848814566084561208936545930199373973145295772789152592521372748144666774350008398729937774177$$
$$\infty = \text{red line}$$
$$\chi^2 = \text{orange arrow}$$
$$\Sigma \gg = \text{blue bracket}$$
$$! = \text{red dot}$$

Course Objectives

*The objective of this course is to provide a **holistic introduction** to the **basic concepts** of computer security, which will provide the necessary basis for **ongoing scholarly studies** and **starting professional activities** in the area of computer security*

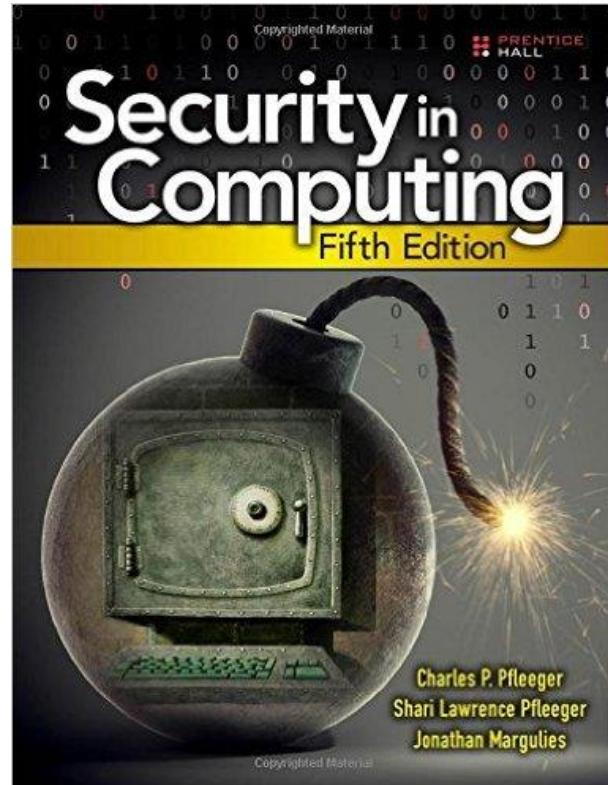
- No chain is stronger than its weakest link so all aspects that impact on security must be examined
- Our intention is to cover topics broadly, but not in great detail
 - Coverage should be sufficient to allow ongoing scholarly studies in computer security (possible courses listed on the next slide)
 - Coverage should be sufficient to allow a motivated engineer to learn more about computer security from self-study and on the job learning
- A few important topics are not covered at all
 - Some will be covered in later courses for those who continue

Course Overview

- Lectures (B303A/A042)
- Labs (B308/databars: 001,017,101,109,117,127)
 - Labs must be documented by a short report
 - *4 mandatory assignments*
- Examination
 - Evaluation of reports from 4 mandatory assignments
 - Separate report on chosen topic from the course
 - *Developing a multiple choice question*
 - Final exam (multiple choice)
 - *10% of questions on the final exam will be selected among the student contributed questions, small "obfuscations" are possible*

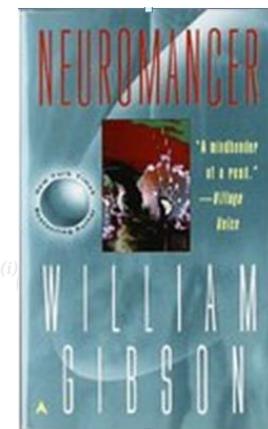
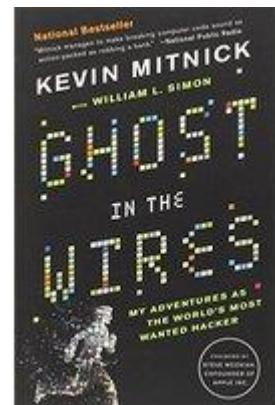
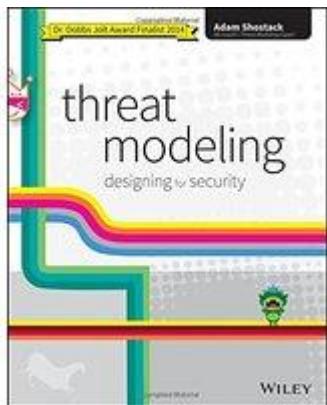
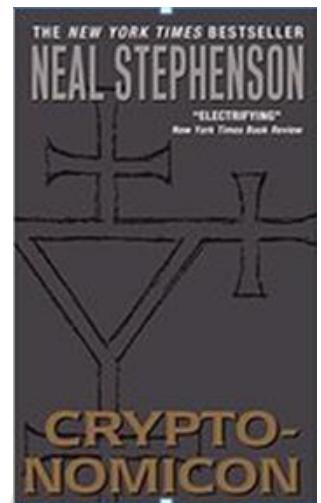
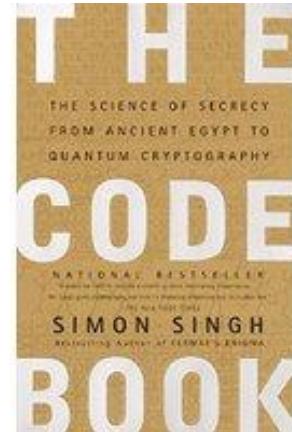
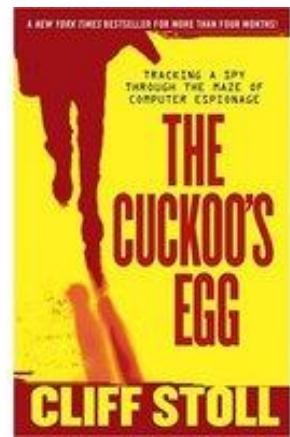
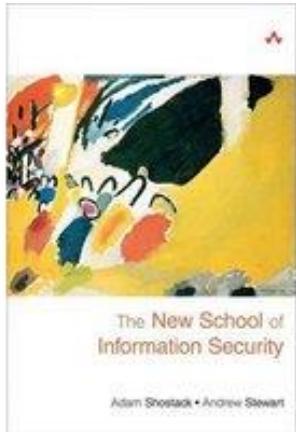
Textbook

- Pfleeger & Pfleeger: Security in Computing (5th Edition)



$$\sqrt{17} \Theta^{\int_a^b} + \Omega \delta e^{i\pi} = \\ \int_{\infty}^{\infty} \frac{\partial}{\partial t} \left[\frac{x^2}{\sum \Sigma!} \right] \{2.7182818284, \dots\}$$

Extra Curricular Security Reading



Course Schedule

Date	Topic
1 Sep	Introduction
8 Sep	Protocol Security I
15 Sep	Cryptography I
22 Sep	Protocol Security II
29 Sep	Cryptography II
6 Oct	Security in Networks
13 Oct	Authentication and Identity Management
20 Oct	Autumn Holiday
27 Oct	Protection/Access Control
3 Nov	Privacy and Privacy Enhancing Technologies
10 Nov	Security Management
17 Nov	Software Security I
24 Nov	Software Security II
1 Dec	Legal Issues

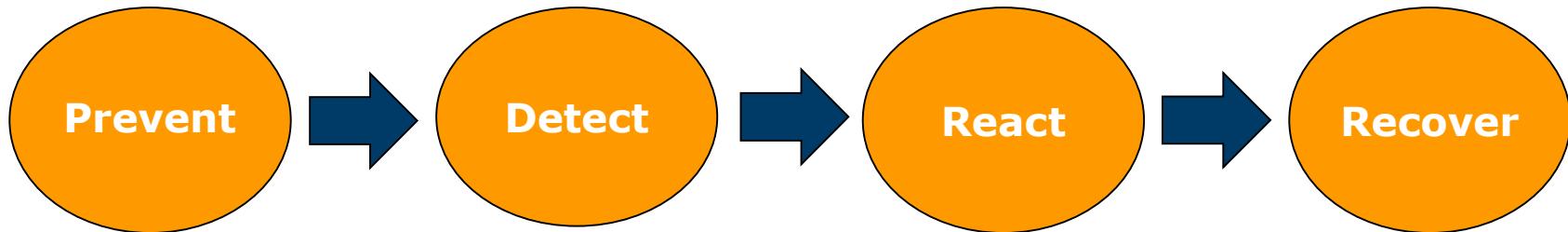
Security courses at DTU Compute

- Three research groups that focus on security at DTU Compute
 - Embedded Systems Engineering
 - *Investigates security in embedded systems incl. Wireless Sensor Networks and Internet of Things*
 - Formal Methods
 - *Focuses on modelling and analysis of information systems using language-based techniques and tools*
 - Cyber Security
 - *Focuses on models, policies and (cryptographic) mechanisms to develop and support secure computing systems*
- Security courses at DTU Compute
 - 02239 Data Security (fall)
 - 02232 Applied Cryptography (fall)
 - 02255 Practical Cryptology (fall)
 - 02234 Current Topics in Systems Security(fall)
 - 02238 Biometric Systems (June)
 - 02242 Program Analysis (fall)
 - 02233 Network Security (spring)
 - 02244 Language Based Security (spring)
 - 02191 Computer Security Forensics (January)
 - 02192 Computer Security Incident Response (spring)
 - 02193 Ethical Hacking (june)

Computer Security Study Line

- One the following courses is mandatory
 - 41633 Innovation and Product Development
 - 42435 Knowledge based Entrepreneurship
 - 42490 Technology, Economics, Management and Organization (TEMO)
- Selective courses (30 ECTS must be selected from this list)
 - 02220 Distributed Systems
 - 02232 Applied Cryptography
 - 02233 Network Security
 - 02234 Current Topics in System Security
 - 02238 Biometric Systems
 - 02239 Data Security
 - 02242 Program Analysis
 - 02244 Logic for Security
 - 02255 Modern Cryptology
 - 02291 System Integration

Specialist Program in Cybersecurity



Data Security
Applied Cryptology
Current Topics ...
Network Security
Biometric Systems
Program analysis
Language Based ...
System Integration

Ethical Hacking

Incident Response

Security Forensics

Join DTUHax

- Meets most Wednesdays 17-19 in the Hackerlab (322/233)



Event at Børsen (in Danish)

Strategisk cybersikkerhed i 2021

- Invitation to an event on strategic challenges in cybersecurity
 - Arranged by Dansk Erhverv and Women in Technology
 - Friday 3 September, 13-16:30
 - Børssalen
- You can see the program here:

<https://www.danskerhverv.dk/kurser-og-events/strategisk-cybersikkerhed-i-2021/>

- DTU demonstrate a few examples of hands-on hacking, so we have received a code for free registration: **Womenintech2021**

Registration deadline is today 1 September

An Overview of Computer Security



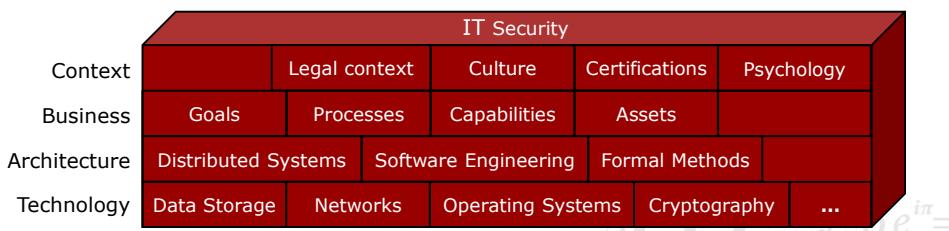
DTU Compute

Department of Applied Mathematics and Computer Science

$$\begin{aligned} \Theta^{\sqrt{17}} &+ \Omega^f \delta e^{i\pi} = \\ \infty^{\infty} &= \{2.7182818284 \\ \Sigma! &\gg \end{aligned}$$

Elements of Cybersecurity

- Cybersecurity must consider aspects from many domains of human activity, in addition to the theory and technologies normally attributed to the domain



$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$

... this makes it challenging, exciting and rewarding to work with!

The Basic Components

- Primary Security Goals (CIA-properties)
 - Confidentiality
 - Integrity
 - Availability
- Other goals frequently listed
 - Accountability
 - Actions can be traced back to a single entity
 - People can be made responsible for their actions
 - Principle known as non-repudiation in cryptography
 - Privacy (e.g. privacy families defined by Common Criteria)
 - Pseudonymity, unlinkability, anonymity, unobservability
 - There is an inherent conflict between accountability and privacy
 - Authenticity
 - Requests or information are authentic and authenticated
 - Resources (both hardware and software) are genuine

Confidentiality

- Preventing unauthorised observation of information or resources (keeping secrets secret)
 - War-plans, business strategies, client confidentiality (priest/lawyers), ...
- Particularly important in military information security
 - Security models, policies and mechanisms developed to enforce the need-to-know principle
- Confidentiality can be ensured with cryptography
 - A cryptographic key is used to scramble (encrypt) data so that unauthorised entities cannot read it
 - Authorised entities have access to a cryptographic key so that they can restore (decrypt) data to its original form
- Access control mechanisms protect data from unauthorised access
- Confidentiality may extend to protect knowledge about the existence of information or resources

Integrity

- Preventing unauthorised modification of information or resources
 - Data integrity pertains to the content of the information
 - Origin integrity pertains to the source of the information
 - *Origin integrity implies authentication of the source of the information, which is part of authenticity*
- Two classes of integrity mechanisms:
 - Prevention mechanisms
 - *Prevents data from being modified in unauthorised ways*
Prevents the bank's janitor from modifying my bank account, but does not prevent the bank manager from moving all my money to his own account
 - Detection mechanisms
 - *Detects unauthorised modification of data after the fact*
Prevents neither of the scenarios above, but allows both to be detected and corrected
- Integrity is often more important than confidentiality in commercial information systems

Availability

- Availability means that the systems information and resources are available to authorised users when they need them
- Attacks against availability is known as *Denial-of-Service* (DoS)
 - Many spectacular DoS attacks reported in the press
- Availability is devilishly difficult and most security research has focused on confidentiality and integrity
 - It is easy to ensure confidentiality and integrity, simply unplug the computer and store it in a bank vault
- Difficulties in ensuring availability include:
 - Difficult to distinguish between high load and DoS (./-phenomena)
 - Influenced by factors outside the security model
 - *A backhoe may be used to cut power or communication supplies*

Risks

- Security is concerned with management of risk
 - Eliminating or reducing harm to assets
- Material Harm
 - Theft of property (e.g. computers, peripherals, ...) or money
 - Harm to people or property (e.g. health, vandalism, ...)
- Immaterial Harm
 - Theft of Intellectual Property (incl. copyright violations)
 - Harm to Intellectual Property (e.g. disclosure of trade secrets)
 - Harm to reputation (e.g., website defacement, bad mouthing, ...)
- Risk Management
 - Risk Identification
 - Risk Analysis/Assessment
 - Risk Treatment
 - Monitoring/Review

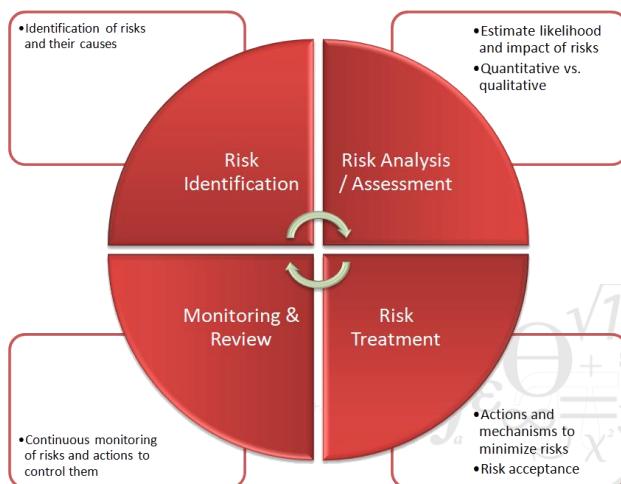


7

DTU Compute Technical University of Denmark

02239 – Data Security

Risk Management Cycle



8

DTU Compute Technical University of Denmark

02239 – Data Security

Threats

- A threat is a potential violation of security
 - Often a four step process
 - threat → vulnerability → opportunity → attack (exploit)
- Four major classes of threats:
 - Disclosure (unauthorised access to information)
 - Deception (acceptance of false data)
 - Disruption (interruption or prevention of correct operation)
 - Usurpation (unauthorised control of (part of) the system)
- Five ways to deal with the effects of exploits:
 - Prevention (remove all vulnerabilities)
 - Deterrence (making exploits difficult – *but not impossible*)
 - Deflection (make other targets relatively more attractive)
 - Detection (as they happen or after the fact – *forensics*)
 - Recovery (restore the system to a usable state)

Vulnerabilities

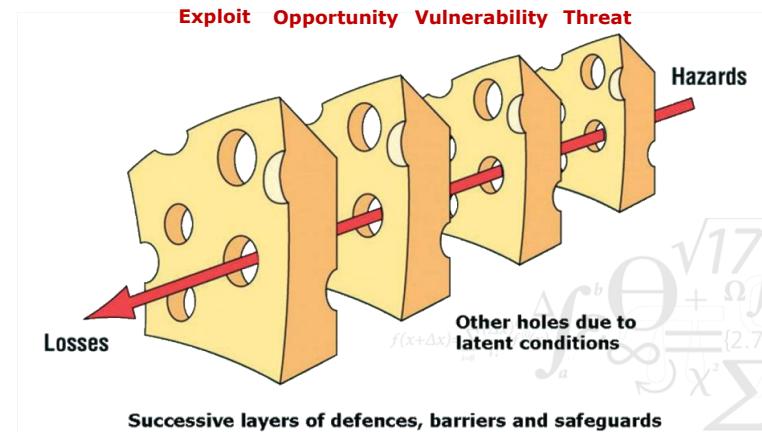
- Weaknesses in the Security Architecture
 - Weak assumptions
 - *Security requirements not specified or poorly understood*
 - Weak architecture
 - *Security requirements not properly identified*
 - *Security architecture does not cover all security requirements*
 - *Security Architecture not up to date (outdated requirements)*
 - Weak components
 - *Poor specification of components of the security architecture*
 - *Poor implementation of components of the security architecture*
 - *Components do not compose securely*
- Weak operation
 - Poor recruitment processes
 - Poor security awareness



Threats, Vulnerabilities and Attacks

Putting it All Together

- The "Swiss Cheese" Model

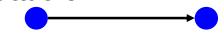


Pause



Network Attacks

Active attacks



Normal communication



Deletion



Modification

Passive attacks



Eavesdropping



Fabrication



Traffic analysis

Possible Attackers

- Insiders ($>50\%$)
 - Disgruntled employees
 - Guests, consultants, contract workers ...
- Crackers (*hackers*)
 - Technically knowledgeable programmers
- Script-Kiddies (*cracker wannabes*)
 - Tools provided by others
- Spies (*industrial and military*)
 - Technical knowledge, technical means, many resources
- Criminals (*thieves, organized crime*)
 - Technical knowledge, technical means, many resources
- Hacktivists and Terrorists
 - Technical knowledge and means, disproportionate allocation of resources
- We need to consider: *means (method), motives and opportunity*

Means of attackers

- Insiders
 - Knowledge of system configuration, network topologies, processes,...
 - Only computing resources provided by organisation
- Crackers (*hackers*)
 - Able to adapt tools to configuration of target
 - Able to write new tools/exploits
 - Few computing resources (apart from bot-nets)
- Script-Kiddies (*cracker wannabes*)
 - Can only use tools provided by others (already known attacks)
- Spies (*industrial and military*)
 - Technical knowledge, rich computing resources, other resources
- Criminals (*thieves, organized crime*)
 - Technical knowledge, technical means, many resources
- Terrorists
 - Probably between spies and script-kiddies, but nothing is really known

Motivation for Attackers

- Curiosity about how the system works
 - The challenge of hacking the system
 - "Ethical hacking" (expose vulnerabilities and warn owners)
 - *SIEM cannot tell difference between white-hat and black-hat hackers, so defenders must always react*
- Fame
 - Recognition for their achievements
- Financial Gains
 - Fraud, theft
 - Industrial Espionage
- Ideology
 - Hactivism: disrupt but do not cause serious damage
 - Cyberterrorism: disrupt/destroy important services

Opportunities for Attacks

- Insiders
 - Daily access to the network, relatively easy to launch attacks
- Crackers
 - Difficult to access internal network
 - Have to crack the firewall first
- Script-Kiddies
 - Can only exploit holes in the firewall (and possibly wireless access)
- Spies
 - Combination of insiders and crackers
 - 3 most important means of gathering information are:
 - *Beatings, Bribery and Blackmail (3 Bs)* -- Robert Morris Sr.
- Criminals
 - Ability to infiltrate, 3Bs, otherwise difficult
- Terrorists
 - Similar to criminals

Policy and Mechanisms

- It is always important to distinguish between the policy and the mechanism that enforces the policy
 - **Policy:** statement of what is, and what is not, allowed
 - **Mechanism:** method, tool or process for enforcing a security policy
- Security Policies may be defined in different ways
 - Different levels of detail from very general (users must not copy other users' files) to very specific (user A must not copy user B's files)
 - Different levels of accuracy from general statements in English to precise mathematical formalisms
 - *Formal statements are more precise when they are formulated correctly*
- When multiple organizations collaborate the composed entity often has a security policy based on the individual security policies.
 - This raises the problem of policy composition which is inherently difficult

Goals of Security Mechanisms

- Security mechanisms are put in place to *prevent* attacks, *detect* attacks and *recover* from attacks
- Prevention
 - Cryptography and access control are often used to prevent attacks
- Detection
 - Intrusion detection systems are often used to detect attacks during or after the attack
- Recovery
 - React to the attack
 - *Common reactions are to stop the attack in progress or allow it to continue with extensive logging (to allow the attacker to be traced)*
 - Repair the damage caused by the attack
 - *Identify the vulnerability, identify appropriate prevention mechanism (e.g., patch the system), determine damage caused by the attack, repair damage caused by the attack (e.g., roll-back of data bases to before the attack)*

Assumptions and Trust

- Security policies are always based on some assumptions about the behaviour of components and entities in the system
- Two assumptions are generally made:
 - Security policy unambiguously partitions the system state into *secure* and *nonsecure* states
 - Security mechanism will guarantee that a system in the *secure* states will never become *nonsecure*
 - If either assumption fails the system is insecure
- A security mechanism can be characterised as:
 - Secure: it does not allow the system to enter *nonsecure* states
 - Precise: it allows the system to enter all *secure* states
 - Broad: it allows the system to enter states that are *nonsecure*
 - In practise, most security mechanisms are broad

Trust Assumptions

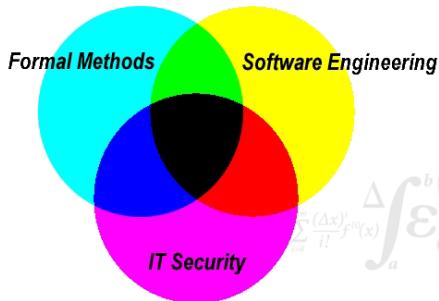
- Trusting that mechanisms work requires several assumptions
 - Each mechanism is designed to implement one or more elements of the security policy
 - The union of security mechanisms implements all aspects of the security policy
 - The mechanisms are implemented correctly
 - The mechanisms are installed and administered correctly
- So what is required to trust encrypted data from the network
 - Encryption algorithm must be strong (design)
 - Encryption algorithm must be implemented correctly (implementation)
 - Encryption software must be installed correctly (operation)
 - Cryptographic key must be secret (administration)

Assurance

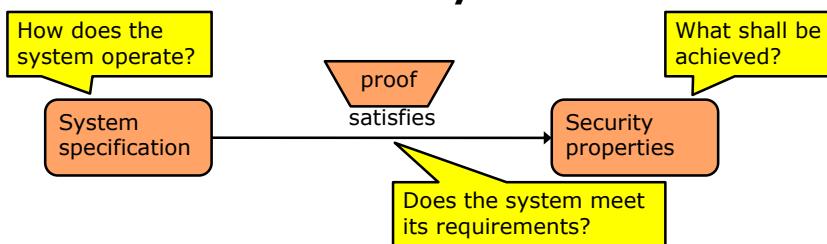
- Assurance attempts to quantify some of the assumptions about trust in the security mechanism
- Assurance requires a specification of the behaviour of the system
 - A system is said to *satisfy* a specification if the specification correctly states how the system will function
- Assurance considers all aspects of software development
 - *Specification* of the system must be correct and unambiguous
 - *Design* of the system translates the specification into components that will implement them
 - *Implementation* creates a system that satisfies the design
 - A program is correct if its implementation performs as specified
 - *Testing* verifies *a posteriori* if the implementation is correct
 - NB! testing cannot prove correctness, only incorrectness
- Stronger assurance requires formal proofs of correctness

Formal Methods for Security

- Systems can be understood as mathematical objects.
- Formal methods based on mathematics and logic should be used to model, analyze, and construct them.
- Doing so can substantially improve the security



Formal Methods for Security



- Even the mere attempt to formalize the security properties/goals of a system in a mathematically precise way can be revealing!
- FM group focuses on automatic methods to find either a proof or a counter example – given a specification of the system and the security properties
- Tools based on model-checking, static analysis, abstract interpretation...
- Many attacks have been detected – and fixed -- using such tools:
- H.530, Google-Apps SSO, Kerberos PKInit

Operational Issues

- Security policies and mechanisms must be effective
 - **Defender:** the cost of an effective attack must be higher than the design, implementation and operation of the security infrastructure
 - **Attacker:** the cost of an effective attack must be lower than the benefits gained through the attack
- Risk analysis determines what attacks are plausible and determines the cost of these attacks
 - Credit card companies accept small amounts of fraud, because it is cheaper than implementing all mechanisms to ensure "perfect" security
- Cost benefit analysis establish whether the benefits of a particular countermeasure exceeds the cost of implementing and operating it
- Laws and customs in the local environment plays an important role
 - Different laws on data protection are defined in different jurisdictions
 - Policies and mechanisms that are legal, but unacceptable, in the local environment should be avoided

Organisational Issues

- Implementing security in a large organisation is difficult and technology is often the easy bit (although we often get it wrong)
 - Benefits of security are not directly visible *on the bottom line*
 - Some security mechanisms may actually be costly
 - *Protocol overhead slows down communications*
 - *Requiring passwords to be entered frequently tends to lower productivity*
- Responsibility for computer security is not always obvious
 - IT department has responsibility for servers, hardware and software
 - Individual users are responsible for choosing good passwords (and keeping them secret)
 - If an attack exploits a weak password then who is to blame?
- Social Engineering (e.g. phishing) is surprisingly effective
- Security incidents are not always attacks
 - An employee who reads and sends private emails during work hours is generally breaching policy, but the breach is normally accepted

Human Issues

- Social Engineering is surprisingly effective
 - Main approach used by Kevin Mitnick (FBI Most Wanted for many years)

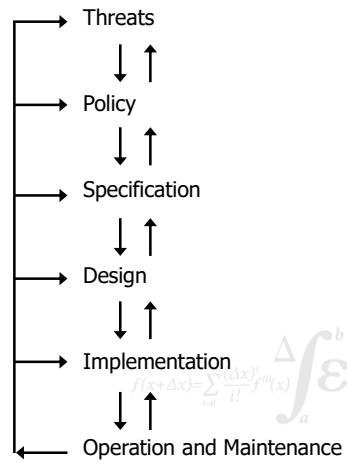


Security Usability

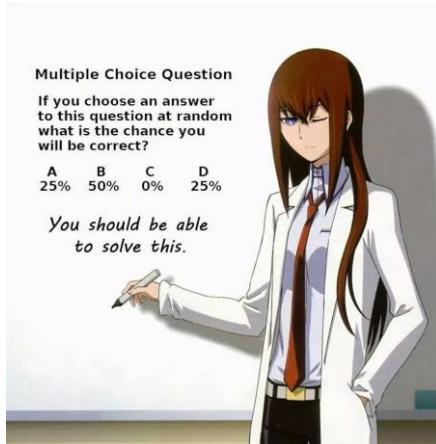
- Security mechanisms must be comprehensible and acceptable



Security Life Cycle



Writing Good Multiple Choice Questions



Multiple Choice Question

If you choose an answer to this question at random what is the chance you will be correct?

A 25% B 50% C 0% D 25%

You should be able to solve this.

DTU Compute

Department of Applied Mathematics and Computer Science

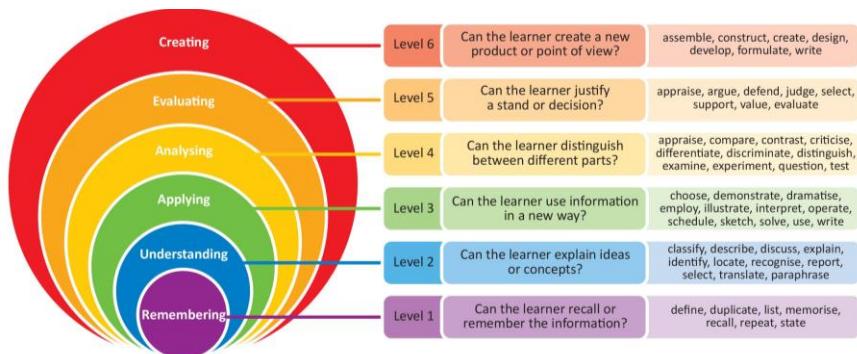
$$\Delta \int_a^b \varepsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = \\ \infty \approx \{2.7182818284 \\ \sum x^2 \gg , !$$

Multiple Choice Question Assignment

- Develop a multiple choice question for the exam in 02239
- Document the development of the question in a short report
 - Topic examined in the question
 - *theory behind the question*
 - *learning objectives being tested*
 - *cognitive level addressed by the question*
 - Design of the question
 - *define the correct answer*
 - *make sure that the one correct answer is unambiguous*
 - Design of the wrong answers (distractors)
 - *plausibility of distractors*
 - *ensuring that distractors are not trick questions*
- Report should be no longer than 3-5 pages

Bloom's Taxonomy of Knowledge

- Classification of educational learning objectives into levels of complexity and specificity



Multiple Choice Questions

- Multiple choice question has 3 components
 - The question (stem)
 - The correct answer (key)
 - The incorrect answers (distractors)

Parts of a Multiple-choice Question

Who sent the first internet email using the "@" symbol? > Stem

- a. Al Gore
- b. Douglas Engelbart
- c. Ray Tomlinson
- d. Vint Cerf

Answer

"Distractors"

Procedural Rules

- Use either the best answer or the correct answer format
 - Best answer format refers to a list of options that can all be correct in the sense that each has an advantage, but one of them is the best
 - Correct answer format refers to one and only one right answer
 - Mark all matching refers to a list of options with several correct answers
- Allow time for editing and other types of item revisions
- Use good grammar, punctuation, and spelling consistently
- Minimize the time required to read each item
- Avoid trick questions
- Use the active voice
- Have your questions peer-reviewed.
- Avoid giving unintended cues – such as making the correct answer longer in length than the distractors

Content-related Rules

- Base each item on an educational or instructional objective of the course, not trivial information
- Test for important or significant information
- Focus on a single problem or idea for each test item.
- Use the author's examples as a basis for developing your items
- Avoid overly specific knowledge when developing items
- Avoid textbook, verbatim phrasing when developing the items
- Avoid items based on opinions
- Use multiple-choice to measure higher level thinking
- Be sensitive to cultural and gender issues
- Use case-based questions that use a common text to which a set of questions refers

Stem Construction Rules

- State the stem in either question form or completion form
- Ensure that the directions in the stem are clear, and that wording lets the examinee know exactly what is being asked
- Avoid window dressing (excessive verbiage) in the stem
- Word the stem positively; avoid negative phrasing such as "not" or "except." If this cannot be avoided, the negative words should always be highlighted by underlining or capitalization:
Which of the following is NOT an example
- Include the central idea and most of the phrasing in the stem.
- Avoid giving clues such as linking the stem to the answer
(.... Is an example of *an*: test-wise students will know the correct answer should start with a vowel)

General Option Development Rules

- Place options in logical or numerical order
- Keep options independent; options should not be overlapping
- Keep all options homogeneous in content
- Keep the length of options fairly consistent.
- Phrase options positively, not negatively
- Avoid distractors that can clue test-wise examinees; for example, absurd options, formal prompts, or semantic (overly specific or overly general) clues
- Avoid giving clues through the use of faulty grammatical construction.
- Avoid specific determinates, such as *never* and *always*

Distractor Development Rules

- Use plausible distractors
- Avoid technically phrased distractors
- Use familiar yet incorrect phrases as distractors
- Use true statements that do not correctly answer the item
- Avoid the use of humour when developing options



9 DTU Compute Technical University of Denmark

02239 Data Security

Ideas for Good Multiple Choice Questions

- Present practical or real-world situations
- Present a diagram of system and ask for application, analysis or evaluation
- Present actual quotations taken from newspapers or other published sources and ask for the interpretation or evaluation of these quotations
- Use pictorial materials that require students to apply principles and concepts
- Use charts, tables or figures that require interpretation

10 DTU Compute Technical University of Denmark

02239 Data Security

Scenario-Based Problem Solving Item Set

- Good for testing higher level cognitive skills
- Present a scenario and ask questions that:
 - require understanding of the theory
 - require application of theory and techniques to specific scenario
 - may require calculations on scratch paper to answer



02239 Data Security Introduction to Protocol Security

Sebastian Mödersheim



September 8, 2021



```

$ file chall
chall: ELF 32-bit LSB executable, Intel 80386,, statically linked, stripped
$ readelf -s chall
readelf: Error: Unable to read in 0x28 bytes of section headers
readelf: Error: Unable to read in 0x488 bytes of section headers
readelf: Error: Unable to read in 0x100 bytes of program headers
$ hexdump -C chall | head -2
00000000  7f 45 4c 46 01 01 01 13  37 13 37 13 37 13 37 00  |.ELF....7.7.7.7.| 
00000010  02 00 03 00 01 13 37 00  90 84 04 08 34 13 37 00  |.....7.....4.7.| 
$ hexdump -C /bin/ls | head -2
00000000  7f 45 4c 46 01 01 00  00 00 00 00 00 00 00 00 00  |.ELF.....| 
00000010  02 00 03 00 01 00 00  b4 c1 04 08 34 00 00 00 00  |.....4...| 
$ cp chall wip
$ sed -i 's/\x13\x37\x00\x00/g' wip
$ readelf -s wip

```

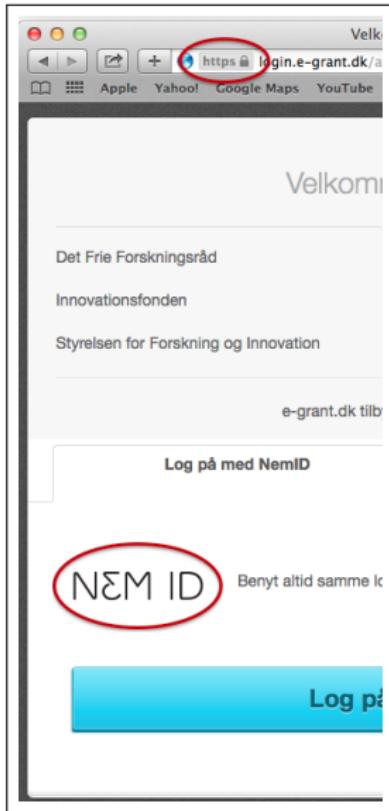
Symbol table '.dynsym' contains 12 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	printf@GLIBC_2.0 (2)
2:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	free@GLIBC_2.0 (2)
3:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	fgets@GLIBC_2.0 (2)
4:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	malloc@GLIBC_2.0 (2)
5:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	puts@GLIBC_2.0 (2)
6:	00000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
7:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	exit@GLIBC_2.0 (2)
8:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	strlen@GLIBC_2.0 (2)
9:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.0 (2)
10:	0804889c	4	OBJECT	GLOBAL	DEFAULT	16	_IO_stdin_used
11:	08049aa0	4	OBJECT	GLOBAL	DEFAULT	26	stdin@GLIBC_2.0 (2)

\$./wip

Ready to become 1337?

Security Protocols



- TLS/SSL
- NemID
- IPSec,
IKE/IKEv2
- Kerberos
- WEP/WPA
- Single Sign On
- Mobile IP
- SIP, SOAP,
geopriv.



Today's Program

① Introduction

② Needham-Schroeder

③ Development of a Key-Exchange Protocol in AnB

④ Diffie-Hellman

Cryptographic Building Blocks (I)

- Symmetric Cryptography:
 - ★ two (or more) agents share a secret key K
 - ★ $\{M\}_K$ denotes symmetric encryption of message M with key K
 - ★ one can decrypt $\{M\}_K$ only when knowing K

Cryptographic Building Blocks (I)

- Symmetric Cryptography:
 - ★ two (or more) agents share a secret key K
 - ★ $\{M\}_K$ denotes symmetric encryption of message M with key K
 - ★ one can decrypt $\{M\}_K$ only when knowing K
- Asymmetric Encryption (Public-key Encryption):
 - ★ every agent A has a key-pair $(K, \text{inv}(K))$ consisting of
 - ▶ the public key K that everybody knows
 - ▶ the private key $\text{inv}(K)$ that only A knows
 - ★ $\{M\}_K$ denotes asymmetric encryption of M with public key K .
 - ★ one can decrypt $\{M\}_K$ only when knowing $\text{inv}(K)$

Cryptographic Building Blocks (I)

- Symmetric Cryptography:
 - ★ two (or more) agents share a secret key K
 - ★ $\{M\}_K$ denotes symmetric encryption of message M with key K
 - ★ one can decrypt $\{M\}_K$ only when knowing K
- Asymmetric Encryption (Public-key Encryption):
 - ★ every agent A has a key-pair $(K, \text{inv}(K))$ consisting of
 - ▶ the public key K that everybody knows
 - ▶ the private key $\text{inv}(K)$ that only A knows
 - ★ $\{M\}_K$ denotes asymmetric encryption of M with public key K .
 - ★ one can decrypt $\{M\}_K$ only when knowing $\text{inv}(K)$
- Digital Signatures
 - ★ Signing is “encryption” with a private key
 - ★ Signature checking is “decryption” with a public key
 - ★ Thus, if $(K, \text{inv}(K))$ is the key pair of A then
 - ▶ $\{M\}_{\text{inv}(K)}$ can only be produced by A
 - ▶ but everybody can read M and check that it comes from A .

Cryptographic Building Blocks (II)

- Nonces (From [Number once](#)): random numbers that is used only once for a [challenge-response](#)

Cryptographic Building Blocks (II)

- Nonces (From [Number once](#)): random numbers that is used only once for a [challenge-response](#)
- Hashes
 - ★ $h(M)$ denotes the [cryptographic hash](#) of message M .
 - ★ Hard to [invert](#): given $h(M)$, find M .
 - ★ Hard to [find collisions](#): find M and M' with $h(M) = h(M')$.
 - ★ Variant: [Message Authentication Code](#) (MAC, keyed hash)
 $h(K, M)$ additionally has a symmetric key K .

Cryptographic Building Blocks (II)

- Nonces (From [Number once](#)): random numbers that is used only once for a [challenge-response](#)
- Hashes
 - ★ $h(M)$ denotes the [cryptographic hash](#) of message M .
 - ★ Hard to [invert](#): given $h(M)$, find M .
 - ★ Hard to [find collisions](#): find M and M' with $h(M) = h(M')$.
 - ★ Variant: [Message Authentication Code](#) (MAC, keyed hash)
 $h(K, M)$ additionally has a symmetric key K .
- Timestamps

Cryptographic Building Blocks (II)

- Nonces (From Number once): random numbers that is used only once for a challenge-response
 - Hashes
 - ★ $h(M)$ denotes the cryptographic hash of message M .
 - ★ Hard to invert: given $h(M)$, find M .
 - ★ Hard to find collisions: find M and M' with $h(M) = h(M')$.
 - ★ Variant: Message Authentication Code (MAC, keyed hash)
 $h(K, M)$ additionally has a symmetric key K .
 - Timestamps
 - Concatenation, i.e., a sequence of messages
 - ★ written as M_1, M_2, M_3 for simplicity
 - ★ reality: quite complex encodings and source of mistakes
- see also: Mödersheim & Katsoris. *A sound abstraction of the parsing problem*, CSF 2014.

Alice and Bob notation

(aka Message Sequence Charts aka Narrations)

A typical protocol description combines prose, data type specifications, various kinds of diagrams, ad hoc notations, and message sequences like

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$
2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$
3. $A \rightarrow B : \{NB\}_{pk(B)}$

They often include informal statements concerning the properties of the protocol and why they should hold.

There are formal languages based on this notation, e.g. [AnB](#).

Protocols (cont.)

What does a message $A \rightarrow B : M$ actually mean?

We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material.

We also assume that each principal has a secure environment in which to compute such as is provided by a personal computer...

Needham and Schroeder

We will be more precise later..

Today's Program

① Introduction

② Needham-Schroeder

③ Development of a Key-Exchange Protocol in AnB

④ Diffie-Hellman

An authentication protocol

The Needham-Schroeder Public Key protocol (NSPK, 1978):

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$
2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$
3. $A \rightarrow B : \{NB\}_{pk(B)}$

How the protocol is executed

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$
2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$
3. $A \rightarrow B : \{NB\}_{pk(B)}$

Role A:

1. Generate nonce NA , compose $\{NA, A\}_{pk(B)}$ and send to B .
2. Receive some message M .
Decrypt M it with $\text{inv}(pk(A))$.
Split the result into two nonces NA' and NB .
Check that $NA = NA'$
If any of this fails, reject M .
3. Compose $\{NB\}_{pk(B)}$ and send to B .

Problem with protocols

- Goal: mutual (entity) authentication.
- Correctness argument (informal):

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$ “This is Alice and I have chosen a nonce NA .”

2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$ “Here is your Nonce NA . Since I could read it, I must be Bob. I also have a challenge NB for you.”

3. $A \rightarrow B : \{NB\}_{pk(B)}$ “You sent me NB . Since only Alice can read this and I sent it back, I must be Alice.”

Problem with protocols

- Goal: mutual (entity) authentication.
- Correctness argument (informal):

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$ “This is Alice and I have chosen a nonce NA .”

2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$ “Here is your Nonce NA . Since I could read it, I must be Bob. I also have a challenge NB for you.”

3. $A \rightarrow B : \{NB\}_{pk(B)}$ “You sent me NB . Since only Alice can read this and I sent it back, I must be Alice.”

Protocols are typically small and convincing...

Problem with protocols

- Goal: mutual (entity) authentication.
- Correctness argument (informal):

1. $A \rightarrow B : \{NA, A\}_{pk(B)}$ “This is Alice and I have chosen a nonce NA .”

2. $B \rightarrow A : \{NA, NB\}_{pk(A)}$ “Here is your Nonce NA . Since I could read it, I must be Bob. I also have a challenge NB for you.”

3. $A \rightarrow B : \{NB\}_{pk(B)}$ “You sent me NB . Since only Alice can read this and I sent it back, I must be Alice.”

Protocols are typically small and convincing... **and wrong!**

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B : & \{NA, A\}_{pk(B)} \\ B \rightarrow A : & \{NA, NB\}_{pk(A)} \\ A \rightarrow B : & \{NB\}_{pk(B)} \end{aligned}$$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B : & \{NA, A\}_{pk(B)} \\ B \rightarrow A : & \{NA, NB\}_{pk(A)} \\ A \rightarrow B : & \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

$$1. \quad a \rightarrow i : \{na, a\}_{pk(i)}$$

$$1.' \quad i(a) \rightarrow b : \{na, a\}_{pk(b)}$$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

$$1. \quad a \rightarrow i : \{na, a\}_{pk(i)}$$

$$1.' \quad i(a) \rightarrow b : \{na, a\}_{pk(b)}$$

$$2.' \quad b \rightarrow i(a) : \{na, nb\}_{pk(a)}$$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

3.' $i(a) \rightarrow b : \{nb\}_{pk(b)}$

Man-in-the-middle attack

NSPK (1978)

$$\begin{aligned} A \rightarrow B : & \{NA, A\}_{pk(B)} \\ B \rightarrow A : & \{NA, NB\}_{pk(A)} \\ A \rightarrow B : & \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

3.' $i(a) \rightarrow b : \{nb\}_{pk(b)}$

What went wrong?

What went wrong?

NSPK (1978)

$$\begin{aligned} A \rightarrow B &: \{NA, A\}_{pk(B)} \\ B \rightarrow A &: \{NA, NB\}_{pk(A)} \\ A \rightarrow B &: \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

3.' $i(a) \rightarrow b : \{nb\}_{pk(b)}$

What went wrong?

NSPK (1978)

$$\begin{aligned} A \rightarrow B : & \{NA, A\}_{pk(B)} \\ B \rightarrow A : & \{NA, NB\}_{pk(A)} \\ A \rightarrow B : & \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

$$\begin{aligned} 1.' \quad & i(a) \rightarrow b : \{na, a\}_{pk(b)} \\ 2.' \quad & b \rightarrow i(a) : \{na, nb\}_{pk(a)} \end{aligned}$$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

3.' $i(a) \rightarrow b : \{nb\}_{pk(b)}$

Nothing in second message indicates who created it.

What went wrong?

NSPK (1978)

$$\begin{aligned} A \rightarrow B : & \{NA, A\}_{pk(B)} \\ B \rightarrow A : & \{NA, NB\}_{pk(A)} \\ A \rightarrow B : & \{NB\}_{pk(B)} \end{aligned}$$

Attack (Lowe 1996):

1. $a \rightarrow i : \{na, a\}_{pk(i)}$

1.' $i(a) \rightarrow b : \{na, a\}_{pk(b)}$

2.' $b \rightarrow i(a) : \{na, nb\}_{pk(a)}$

2. $i \rightarrow a : \{na, nb\}_{pk(a)}$

3. $a \rightarrow i : \{nb\}_{pk(i)}$

3.' $i(a) \rightarrow b : \{nb\}_{pk(b)}$

Nothing in second message indicates who created it.

The protocol **wrongly assumes** that the second message can only be created by the person to whom A had sent NA in the first step.

What went wrong?

- Problem in step 2.

$$B \rightarrow A : \{NA, NB\}_{pk(A)}$$

Agent B should also give his name: $\{NA, NB, \textcolor{green}{B}\}_{pk(A)}$.

- Known as [Lowe's Fix](#).
- Is the improved version now correct?

Today's Program

- ① Introduction
- ② Needham-Schroeder
- ③ Development of a Key-Exchange Protocol in AnB
- ④ Diffie-Hellman

AnB and OFMC

- AnB: Formal language based on [Alice and Bob](#) notation for security protocols
- OFMC: Open-Source Fixedpoint Model-Checker
 - ★ Searching for attacks against an AnB protocol
 - ★ Verification for a bounded number of protocol sessions

- Demo/Tutorial for AnB.
- The AnB source codes of the protocol will be put on campusnet
- A similar development is found in the folder *AnB Tutorial* in the OFMC package.

Today's Program

- ① Introduction
- ② Needham-Schroeder
- ③ Development of a Key-Exchange Protocol in AnB
- ④ Diffie-Hellman

Cyclic Group

For a **large** prime p consider:

$$\mathbb{Z}_p^* = \{1, \dots, p-1\}$$

- multiplication **modulo p** ,
e.g. $5 * 3 \bmod 11 = 15 \bmod 11 = 4$.
- exponentiation also modulo p ,
e.g. $5^2 \bmod 11 = 25 \bmod 11 = 3$.

Cyclic Group

Consider exponentiations of the form $g^X \pmod{p}$ with $g, X \in \mathbb{Z}_p^\star$.

E.g. $p = 7$:

X	1	2	3	4	5	6
g=1	1	1	1	1	1	1
g=2	2	4	1	2	4	1
g=3	3	2	6	4	5	1
g=4	4	2	1	4	2	1
g=5	5	4	6	2	3	1
g=6	6	1	6	1	6	1

Cyclic Group

Consider exponentiations of the form $g^X \pmod{p}$ with $g, X \in \mathbb{Z}_p^*$.

E.g. $p = 7$:

X	1	2	3	4	5	6
g=1	1	1	1	1	1	1
g=2	2	4	1	2	4	1
g=3	3	2	6	4	5	1
g=4	4	2	1	4	2	1
g=5	5	4	6	2	3	1
g=6	6	1	6	1	6	1

- g is called a **generator** for \mathbb{Z}_p^* if $\mathbb{Z}_p^* = \{g^X \pmod{p} \mid X \in \mathbb{Z}_p^*\}$.

Cyclic Group

Consider exponentiations of the form $g^X \pmod{p}$ with $g, X \in \mathbb{Z}_p^*$.

E.g. $p = 7$:

X	1	2	3	4	5	6
g=1	1	1	1	1	1	1
g=2	2	4	1	2	4	1
g=3	3	2	6	4	5	1
g=4	4	2	1	4	2	1
g=5	5	4	6	2	3	1
g=6	6	1	6	1	6	1

- g is called a **generator** for \mathbb{Z}_p^* if $\mathbb{Z}_p^* = \{g^X \pmod{p} \mid X \in \mathbb{Z}_p^*\}$.
- In the example $p = 7$, $g = 3$ and $g = 5$ are generators.

Cyclic Group

Consider exponentiations of the form $g^X \pmod{p}$ with $g, X \in \mathbb{Z}_p^*$.

E.g. $p = 7$:

X	1	2	3	4	5	6
g=1	1	1	1	1	1	1
g=2	2	4	1	2	4	1
g=3	3	2	6	4	5	1
g=4	4	2	1	4	2	1
g=5	5	4	6	2	3	1
g=6	6	1	6	1	6	1

- g is called a **generator** for \mathbb{Z}_p^* if $\mathbb{Z}_p^* = \{g^X \pmod{p} \mid X \in \mathbb{Z}_p^*\}$.
- In the example $p = 7$, $g = 3$ and $g = 5$ are generators.
- Given g, p, X it is **easy** to compute $g^X \pmod{p}$.

Cyclic Group

Consider exponentiations of the form $g^X \pmod{p}$ with $g, X \in \mathbb{Z}_p^*$.

E.g. $p = 7$:

X	1	2	3	4	5	6
g=1	1	1	1	1	1	1
g=2	2	4	1	2	4	1
g=3	3	2	6	4	5	1
g=4	4	2	1	4	2	1
g=5	5	4	6	2	3	1
g=6	6	1	6	1	6	1

- g is called a **generator** for \mathbb{Z}_p^* if $\mathbb{Z}_p^* = \{g^X \pmod{p} \mid X \in \mathbb{Z}_p^*\}$.
- In the example $p = 7$, $g = 3$ and $g = 5$ are generators.
- Given g, p, X it is **easy** to compute $g^X \pmod{p}$.
- Given $g, p, g^X \pmod{p}$ it is **believed hard** to compute X .

Kivinen & Kojo

RFC 3526

5. 4096-bit MODP Group

Standards Track

MODP Diffie-Hellman groups for IKE

[Page 4]

May 2003

This group is assigned id 16.

This prime is: $2^{4096} - 2^{4032} - 1 + 2^{64} * \{ [2^{3966} p_1] + 240904 \}$

Its hexadecimal value is:

```

FFFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386FBF 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
15728E5A 8AAC42D AD33170D 04507A33 A85521AB DF1CBA64
ECFB8504 58DBEFOA 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
43DB5BFC EOFD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199
FFFFFFFFFF FFFFFFFF

```

The generator is: 2.

Unauthenticated Diffie-Hellman

Protocol : *Diffie-Hellman*

Types :

Agent A, B ;

Number g, X, Y, Msg ;

Function pk ;

Knowledge :

$A : A, B, g, \text{pk}, \text{inv}(\text{pk}(A))$;

$B : B, g, \text{pk}, \text{inv}(\text{pk}(B))$;

Actions :

$A \rightarrow B : \exp(g, X)$

$B \rightarrow A : \exp(g, Y)$

$A \rightarrow B : \{\|A, \text{Msg}\|\}_{\exp(\exp(g, X), Y)}$

Goals :

...

Relies on the algebraic property

$\exp(\exp(g, X), Y) \approx \exp(\exp(g, Y), X)$.

Diffie-Hellman: man-in-the-middle attack

- Diffie-Hellman (without authentication of the half-keys) can be attacked:

$$1. \quad a \rightarrow i(b) : \exp(g, x)$$

$$1.' \quad i(a) \rightarrow b : \exp(g, z)$$

$$2.' \quad b \rightarrow i(a) : \exp(g, y)$$

$$2. \quad i(b) \rightarrow a : \exp(g, z)$$

- a believes to share key $\exp(\exp(g, x), z)$ with b . b believes ...
- The intruder knows both keys
- Prevention: authenticate the half-keys, e.g. with digital signatures:

$$1. \quad A \rightarrow B : \{\exp(g, X)\}_{\text{inv}(\text{pk}(A))}$$

$$2. \quad B \rightarrow A : \{\exp(g, Y)\}_{\text{inv}(\text{pk}(B))}$$

- In general, using Diffie-Hellman for key-exchange is a good idea!

Bibliography

- Matt Bishop. *Computer Security (Art and Science)*. Pearson, 2003.
- Kaufman, Perlman, Speciner. *Network Security: Private Communication in a Public World*, Prentice Hall, 2002.
- Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.
- Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- John Clark and Jeremy Jacob: A survey of authentication protocol literature, 1997. <http://www.cs.york.ac.uk/~jac/>
- Martín Abadi and Roger Needham: Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):2-15, 1996.
- Sebastian Mödersheim and Luca Viganò: The Open-source Fixed-point Model Checker for Symbolic Analysis of Security Protocols. *Fosad 2007-2008-2009, LNCS 5705*, Springer, 2009.

02239 Data Security Security Protocols II

Sebastian Mödersheim



September 22, 2021

Outline

- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

Outline

- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

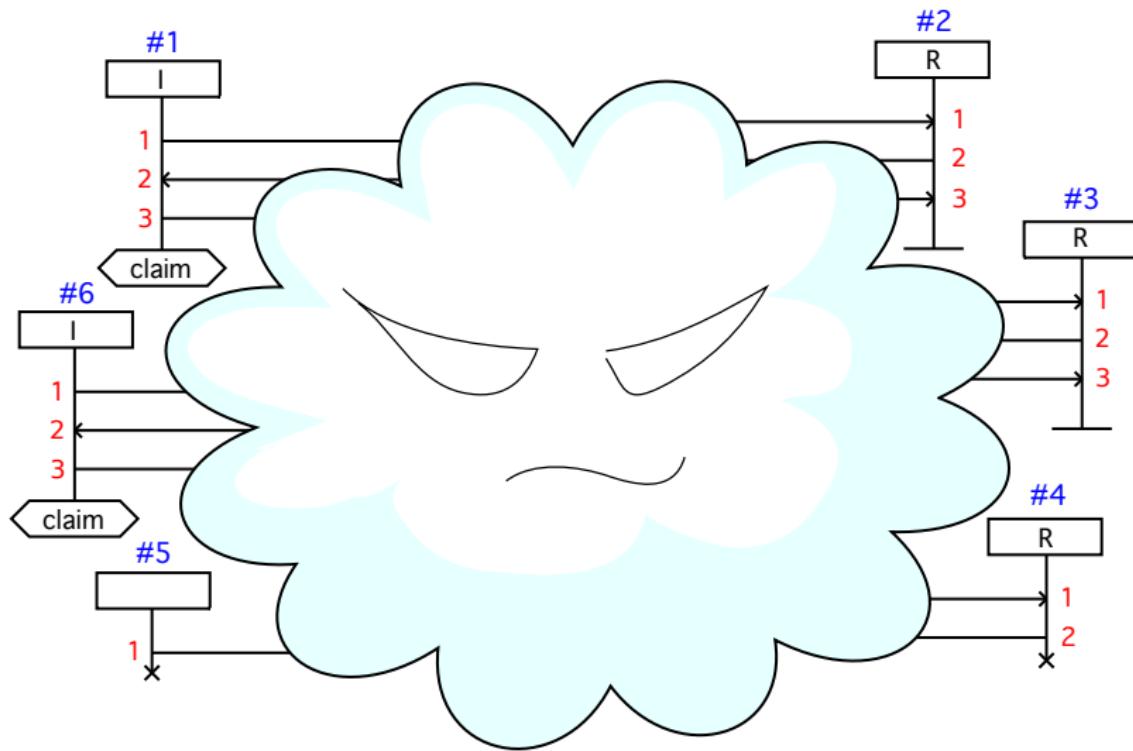
Examples of Intruder Models



- He knows the protocol but cannot break cryptography.
(Standard: **perfect encryption**.)
- He is **passive** but overhears all communications.
- He is **active** and can intercept and generate messages.
“Transfer 100 Kr to Dorrit” ↵ “Transfer 10000 Kr to Charlie”
Worst-case: the intruder controls the entire network
- He might even be one of the principals running the protocol!
We should expect this unless a role is explicitly assumed to be trusted/honest.

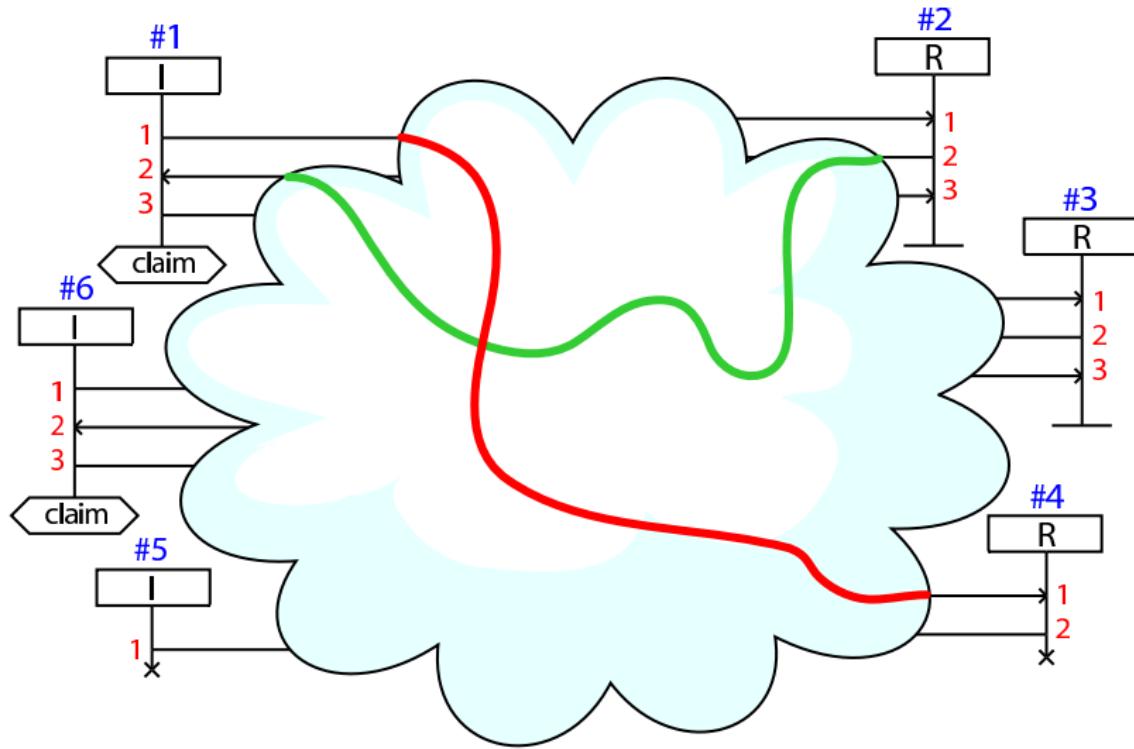
A friend's just an enemy in disguise. You can't trust nobody.
(Charles Dickens, *Oliver Twist*)

Deployment in a Hostile Environment

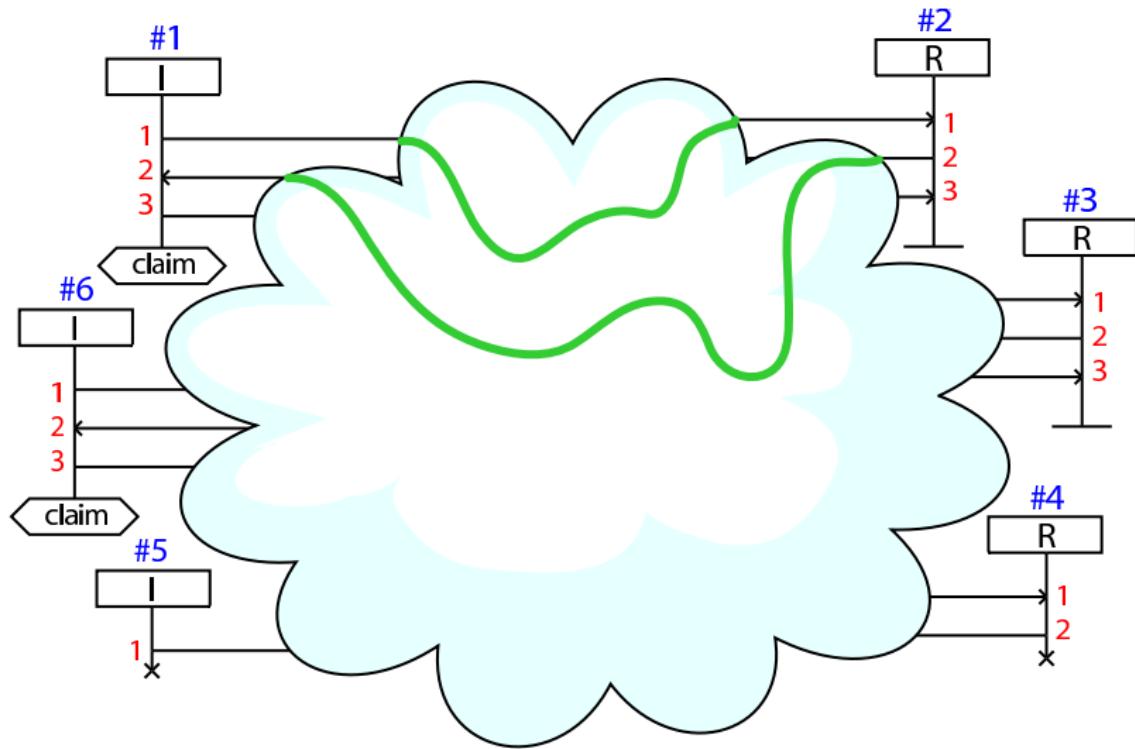


Cloud illustrations from Cas Cremers.

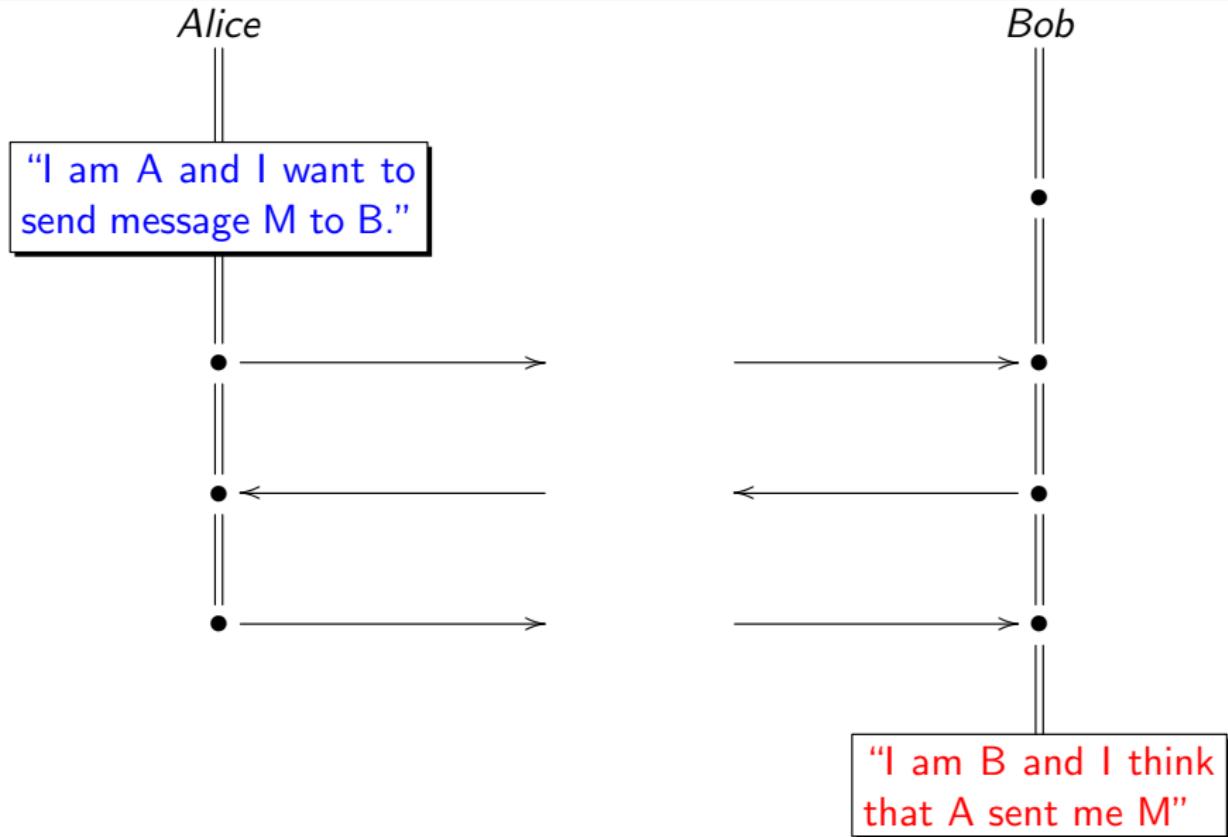
Failed (Weak) Authentication



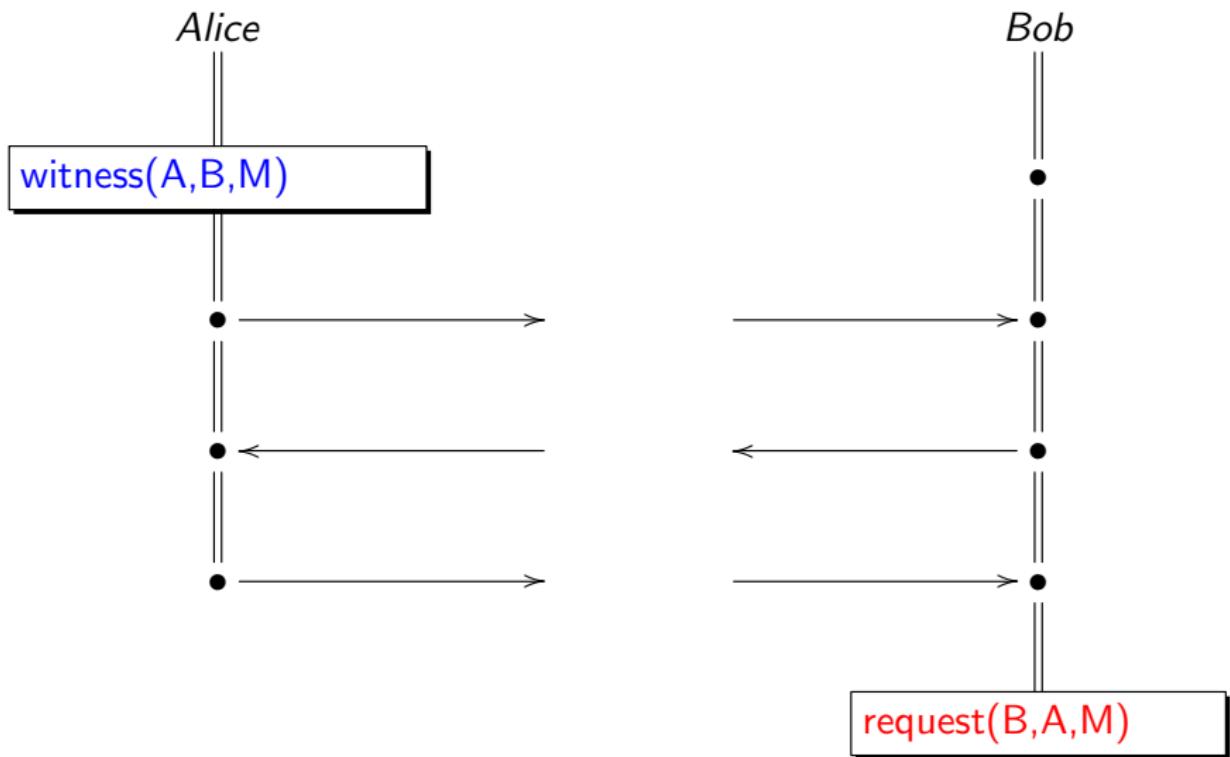
Successful (Weak) Authentication



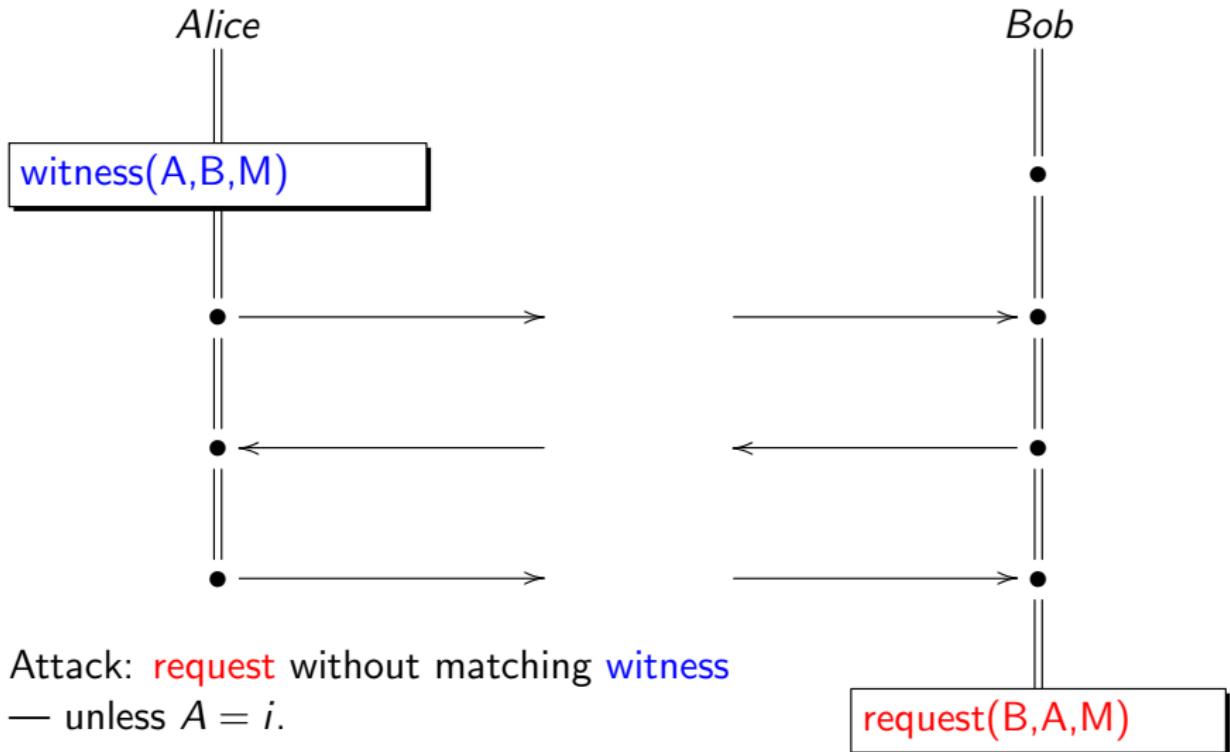
Formalization: (Weak) Authentication



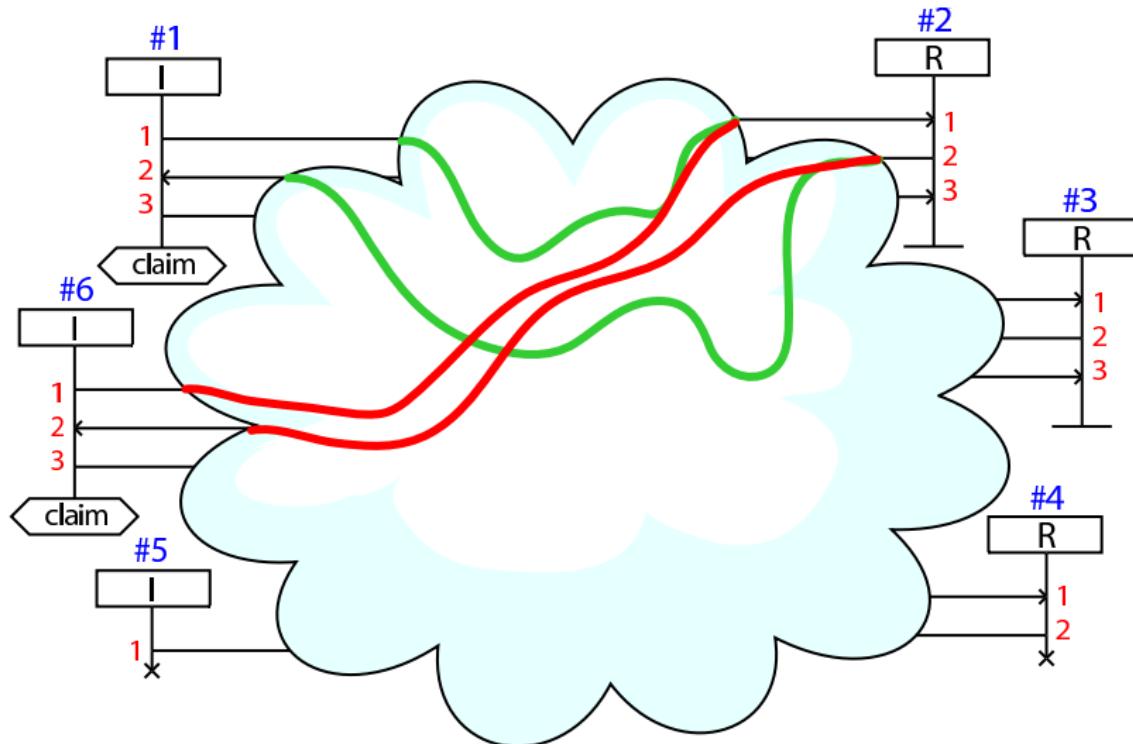
Formalization: (Weak) Authentication



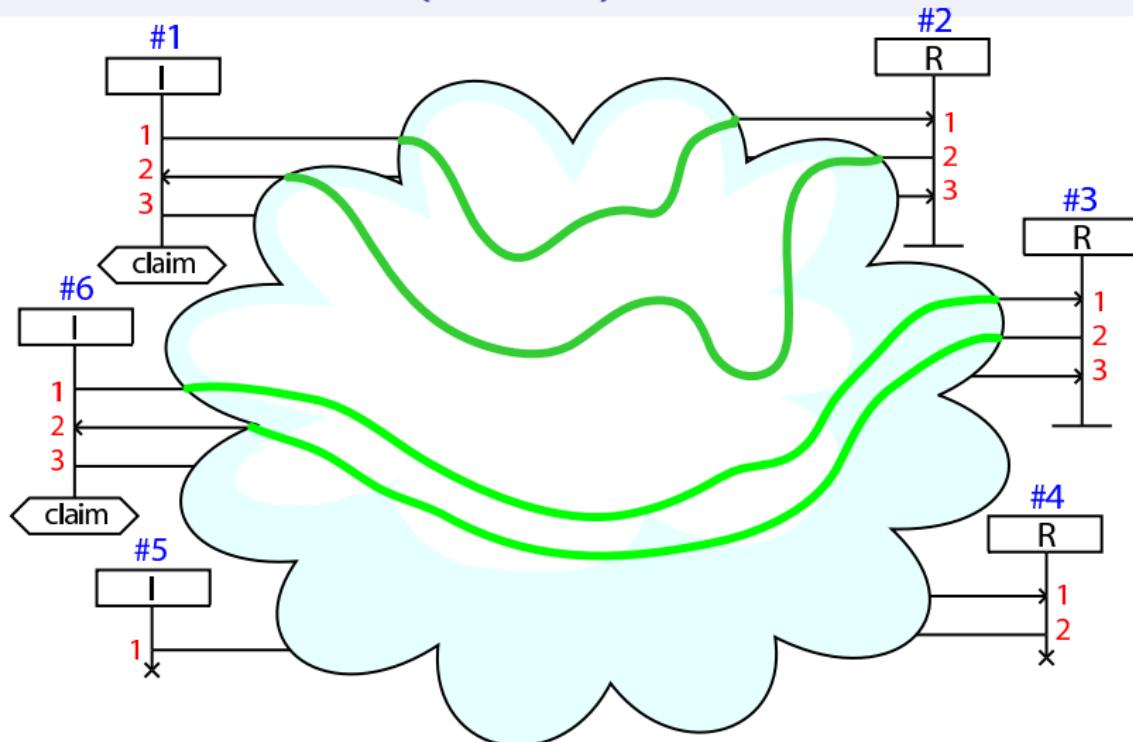
Formalization: (Weak) Authentication



Failed (Strong) Authentication



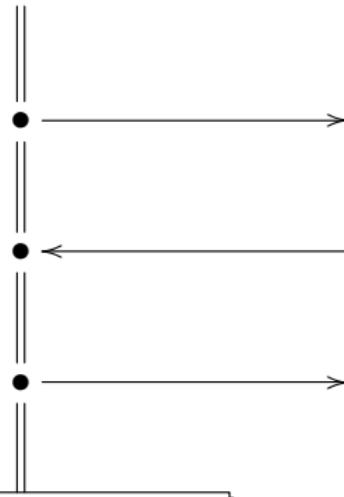
Successful (Strong) Authentication



Attack: more **requests** than **witnesses**.

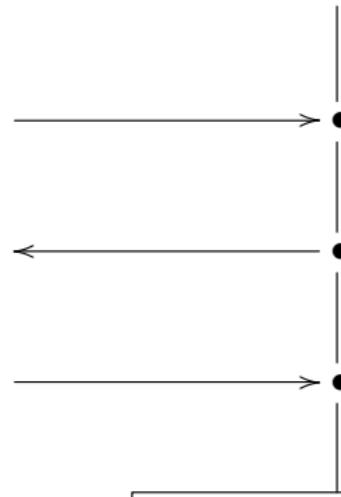
Secrecy: M secret between A,B,C

Alice



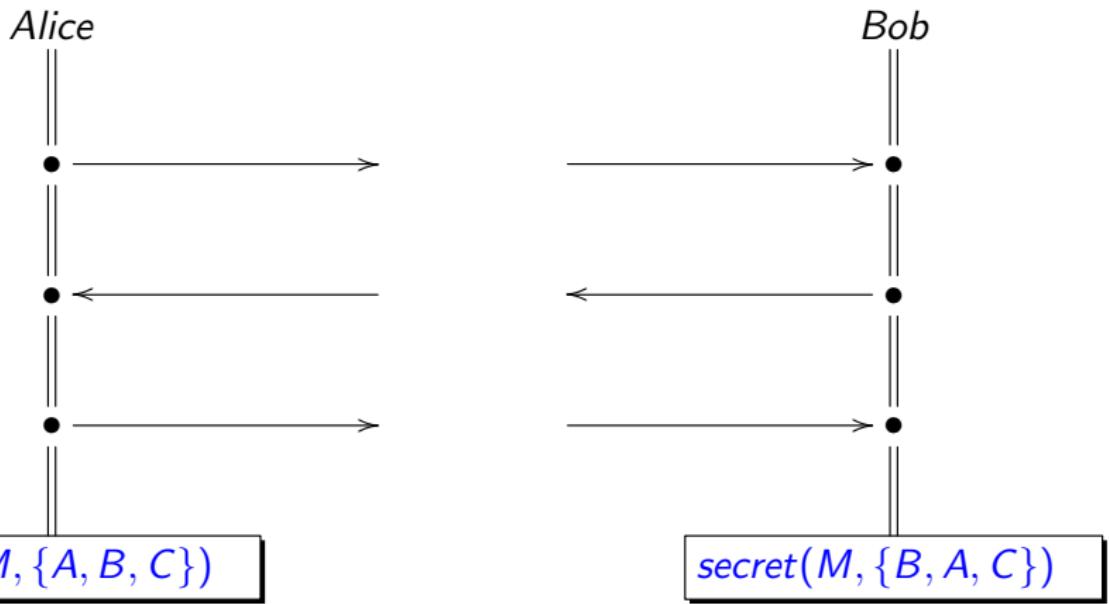
"I am A and I want M
to be secret with B,C."

Bob



"I am B and I want M
to be secret with A,C."

Secrecy: M secret between A,B,C



Attack: $\text{secret}(M, \text{Set})$, intruder knows M and is not a member of the Set.

Outline

- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

Motivation

Example (NSL)

$$A \rightarrow B : \{NA, A\}_{pk(B)}$$
$$B \rightarrow A : \{NA, NB, B\}_{pk(A)}$$
$$A \rightarrow B : \{NB\}_{pk(B)}$$

- Public-key cryptography is used here to ensure **confidential transmission** of messages.

Motivation

Example (NSL)

$$\begin{array}{ll} A \xrightarrow{\bullet} B : NA, A \\ B \xrightarrow{\bullet} A : NA, NB, B \\ A \xrightarrow{\bullet} B : NB \end{array}$$

- Public-key cryptography is used here to ensure **confidential transmission** of messages.
- Abstraction: use a **confidential channel**.

Channel Notation

The Diffie-Hellman assumes an **authentic exchange** of the half-keys:

$$\begin{array}{lll} A & \xrightarrow{\bullet} & B : \exp(g, X) \\ B & \xrightarrow{\bullet} & A : \exp(g, Y) \end{array}$$

- How this exchange is authenticated is not relevant for Diffie-Hellman!
- Many protocols use Diffie-Hellman, e.g. Station2Station, IKE/IKEv2/JFK, Kerberos, TLS, device-pairing. . . .
- Many different ways to authenticate the key-exchange:
Cryptographically Digital signatures, symmetric/asymmetric encryption, MACs.
Non-Cryptographically using a trusted third party, meeting face to face, using additional channels (SMS etc.)
- Using an authentic channel abstracts from the realization.

Channel Notation

- Channels can be both **assumptions** and **goals** of a protocol:

Example

$$\begin{array}{lll} A & \xrightarrow{\bullet} & B : \exp(g, X) \\ B & \xrightarrow{\bullet} & A : \exp(g, Y) \\ A & \xrightarrow{} & B : \{ \textit{Payload} \}_{\exp(\exp(g, X), Y)} \\ \hline A & \xrightarrow{\bullet \rightarrow \bullet} & B : \textit{Payload} \end{array}$$

“Diffie-Hellman creates a secure channel from authentic channels.”

- Actually, public-key cryptography could be defined in a broad sense as a mechanism to obtain secure channels from authentic channels.
- Very general way to see Diffie-Hellman.
- Good for system design and verification: reason about small components with a **well-defined interface**.

Outline

- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

Transport Layer Security

TLS consists of basically two phases:

- **Handshake**: A client (typically webbrowser) and a server establish a “secure channel”: a **pair of symmetric keys** for communicating
- **Transport**: the parties exchange messages over the secure channel (encrypting with the symmetric keys).

Actually, there are some additions like re-establishing a session which we do not discuss.

TLS 1.3 (simplified)

A->B: $A, \exp(g, X)$

B->A: $\exp(g, Y),$

let $k_1 = \text{clientK}(\exp(\exp(g, X), Y))$

let $k_2 = \text{serverK}(\exp(\exp(g, X), Y))$

{| $\{B, \text{pk}(B)\}^{-1}(\text{pk}(s))$ |} $k_2,$

{| $\{h(\exp(g, X), \exp(g, Y))\}^{-1}(\text{pk}(B))$ |} k_2

A->B: {| $h(\exp(g, X), \exp(g, Y))$ |} $k_1,$

{| data, DATA_A |} k_1

B->A: {| data, DATA_B |} k_2

where

- h , clientK , serverK are one-way functions
- $\{B, \text{pk}(B)\}^{-1}(\text{pk}(s))$ is a key certificate issued for B by a trusted third party s .
- data is just a tag to distinguish transport messages.
- DATA_A and DATA_B represent payload messages transmitted on the channel.

TLS 1.3 (simplified)

A->B: $A, \exp(g, X)$

B->A: $\exp(g, Y),$

let $k1 = \text{clientK}(\exp(\exp(g, X), Y))$

let $k2 = \text{serverK}(\exp(\exp(g, X), Y))$

{| $\{B, \text{pk}(B)\}^{-1}(\text{inv}(\text{pk}(s)))$ |} $k2,$

{| $\{h(\exp(g, X), \exp(g, Y))\}^{-1}(\text{inv}(\text{pk}(B)))$ |} $k2$

A->B: $\{| h(\exp(g, X), \exp(g, Y)) | \} k1,$

{| $\text{data}, \text{DATA_A}$ |} $k1$

B->A: $\{| \text{data}, \text{DATA_B} | \} k2$

- Note: only B has a certificate, but not A.
 - ★ TLS can also be deployed when both sides have a certificate.
 - ★ Typically A is a user/webbrowser that does not have a certificate.

TLS 1.3 (simplified)

A->B: $A, \exp(g, X)$

B->A: $\exp(g, Y),$

let $k1 = \text{clientK}(\exp(\exp(g, X), Y))$

let $k2 = \text{serverK}(\exp(\exp(g, X), Y))$

{| $\{B, \text{pk}(B)\}^{-1}(\text{inv}(\text{pk}(s)))$ |} $k2,$

{| $\{h(\exp(g, X), \exp(g, Y))\}^{-1}(\text{inv}(\text{pk}(B)))$ |} $k2$

A->B: $\{| h(\exp(g, X), \exp(g, Y)) | \} k1,$

{| $\text{data}, \text{DATA_A}$ |} $k1$

B->A: $\{| \text{data}, \text{DATA_B} | \} k2$

- Note: only B has a certificate, but not A.
 - ★ TLS can also be deployed when both sides have a certificate.
 - ★ Typically A is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?

TLS 1.3 (simplified)

A->B: $A, \exp(g, X)$

B->A: $\exp(g, Y),$

let $k1 = \text{clientK}(\exp(\exp(g, X), Y))$

let $k2 = \text{serverK}(\exp(\exp(g, X), Y))$

{| $\{B, \text{pk}(B)\}^{-1}(\text{inv}(\text{pk}(s)))$ |} $k2,$

{| $\{h(\exp(g, X), \exp(g, Y))\}^{-1}(\text{inv}(\text{pk}(B)))$ |} $k2$

A->B: $\{| h(\exp(g, X), \exp(g, Y)) | \} k1,$

{| $\text{data}, \text{DATA_A}$ |} $k1$

B->A: $\{| \text{data}, \text{DATA_B} | \} k2$

- Note: only B has a certificate, but not A.
 - ★ TLS can also be deployed when both sides have a certificate.
 - ★ Typically A is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?
- The intruder can impersonate the client A towards B.

TLS 1.3 (simplified)

A->B: $A, \exp(g, X)$

B->A: $\exp(g, Y),$

```
let k1 = clientK(exp(exp(g,X),Y))
```

```
let k2 = serverK(exp(exp(g,X),Y))
```

```
{| {B,pk(B)}inv(pk(s)) |}k2,
```

```
{| {h(exp(g,X),exp(g,Y))}inv(pk(B)) |}k2
```

A->B: $\{| h(\exp(g,X),\exp(g,Y)) | \}k1,$

```
{| data,DATA_A |}k1
```

B->A: $\{| data,DATA_B | \}k2$

- Note: only B has a certificate, but not A.
 - ★ TLS can also be deployed when both sides have a certificate.
 - ★ Typically A is a user/webbrowser that does not have a certificate.
- What kind of channel do we get without the client certificate?
- The intruder can impersonate the client A towards B.
- But B has a secure channel with whoever created the secret X!
 - ★ A can be sure who B is.
 - ★ B cannot be sure who A is.
 - ★ the intruder cannot enter the connection between honest A and B.

Secure Pseudonymous Channels

- Consider the $\exp(g, X)$ of agent A as a **pseudonym** of A .
- The link between A and $\exp(g, X)$ could be achieved by a certificate, but it is not available here.
- The pseudonym $\exp(g, X)$ **cannot be stolen/hijacked** because “ownership” is the knowledge of x .
- This kind of channel is good enough for many applications such as transmitting credit card data or a login.
- We write often for this kind of channel:

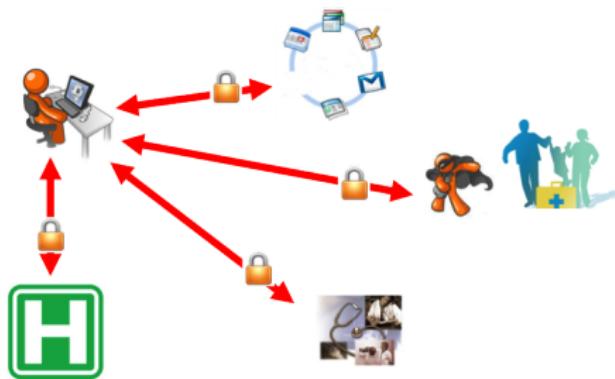
$$[A] \bullet\rightarrow\bullet B : M$$

Outline

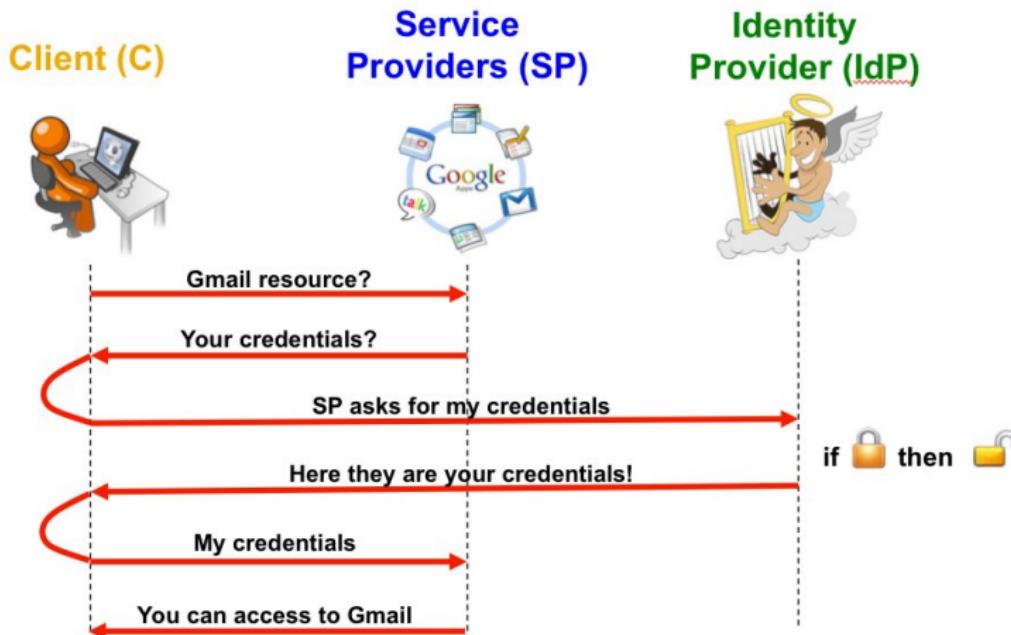
- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

Single Sign-On (SSO)

- Username/passwords for different websites/accounts:
 - ★ Hassle and security problem
- Avoid this by **identity federation**:
 - ★ user has trusted **identity provider (idp)**
 - ★ user signs up only **once** at idp
 - ★ idp **vouches** for them to any website (that trusts idp)



SSO with web-browser



Google's SSO

Knowledge: C: C,idp,SP,pk(idp);
idp: C,idp,pk(idp),inv(pk(idp));
SP: idp,SP,pk(idp)

Actions:

[C] *->* SP : C,SP,URI
SP *->* [C] : C,idp,SP,URI

C *->* idp : C,idp,SP,URI
idp *->* C : {C,idp}inv(pk(idp)),URI

[C] *->* SP : {C,idp}inv(pk(idp)),URI
SP *->* [C] : Data

Goals:

SP authenticates C on URI
C authenticates SP on Data
Data secret between SP,C

Attack on Google's SSO

The attack found by OFMC in nice notation:

1. [a] *->* i: a.i.URI(1)

1.' [i] *->* b: a.b.x306

2.' b *->* [i]: a.idp.b.ID(2).x306

2. i *->* [a]: a.idp.i.x505.URI(1)

3. a *->* idp: a.idp.i.x505.URI(1)

4. idp *->* a: {a.idp}_inv(pk(idp)).URI(1)

5. [a] *->* i: {a.idp}_inv(pk(idp)).URI(1)

5.' [i] *->* b: {a.idp}_inv(pk(idp)).x306

6.' b *->* [i]: Data(6)

The Problem

The authentication assertion from the ipd:

$$\{ID, C, ipd, SP\}_{\text{inv}(\text{pk}(ipd))}$$

- Google had omitted **some parts** that were suggested but not required by the standard.
- This allows a dishonest *SP* to re-use the authentication assertion and log in to other sites as *C*.

Again, this is a problem of a dishonest participant!

Outline

- ① Assumptions and Goals
- ② Channels
- ③ TLS
- ④ Single Sign-On
- ⑤ Password Guessing Attacks

Guessing Attacks

Example: a Variant of Microsoft-ChapV2

Protocol: PW

Types: Agent A, B;

Number NB;

Function pw, h;

Knowledge:

A: A, B, pw(A, B), h;

B: A, B, pw(A, B), h;

Actions:

B->A: NB

A->B: h(pw(A, B), NB)

Goals:

B authenticates A on NB

What if pw(A, B) has low entropy?

Low Entropy Passwords

- Low entropy passwords
 - ★ e.g. natural language words
 - ★ short

Low Entropy Passwords

- Low entropy passwords
 - ★ e.g. natural language words
 - ★ short
- The intruder compiles a **dictionary D** of passwords
 - ★ Relatively small space of passwords (a few million)
 - ★ Contains many weak passwords

Low Entropy Passwords

- Low entropy passwords
 - ★ e.g. natural language words
 - ★ short
- The intruder compiles a **dictionary D** of passwords
 - ★ Relatively small space of passwords (a few million)
 - ★ Contains many weak passwords
- Use D for a **brute-force** attack on an observed message:

Observed message	$nb, h(\text{pw}(a, b), nb)$
Construct	$nb, h(\text{guess}, nb)$ for every $\text{guess} \in D$

Low Entropy Passwords

- Low entropy passwords
 - ★ e.g. natural language words
 - ★ short
- The intruder compiles a **dictionary D** of passwords
 - ★ Relatively small space of passwords (a few million)
 - ★ Contains many weak passwords
- Use D for a **brute-force** attack on an observed message:

Observed message	$nb, h(\text{pw}(a, b), nb)$
Construct	$nb, h(\text{guess}, nb)$ for every $\text{guess} \in D$
- When $\text{pw}(a, b) = \text{guess}$ for some guess, the constructed and the observed message are identical, and the intruder has found out $\text{pw}(a, b)$.

Are weak passwords of any use?

Knowledge:

A: A ,B ,pw(A ,B) ,pk(B);

B: A ,B ,pw(A ,B) ,pk(B) ,inv(pk(B));

Actions :

A->B: { A ,pw(A ,B) ,K }pk(B)

B->A: { | NB | }K

Goals :

B authenticates A on K

A authenticates B on NB

K secret between A ,B

- This is similar to what happens in TLS-based login protocols.

Are weak passwords of any use?

Knowledge:

A: A, B, $\text{pw}(A, B)$, $\text{pk}(B)$;

B: A, B, $\text{pw}(A, B)$, $\text{pk}(B)$, $\text{inv}(\text{pk}(B))$;

Actions:

A \rightarrow B: { A, $\text{pw}(A, B)$, K } $\text{pk}(B)$

B \rightarrow A: { | NB | }K

Goals:

B authenticates A on K

A authenticates B on NB

K secret between A, B

- This is similar to what happens in TLS-based login protocols.
- Guessing not possible – despite low-entropy $\text{pw}(A, B)$
 - ★ The intruder has to create {A, guess , K} $\text{pk}(B)$ for every guess.
 - ▶ That requires guessing K, too!

Are weak passwords of any use?

Knowledge:

A: A, B, $\text{pw}(A, B)$, $\text{pk}(B)$;

B: A, B, $\text{pw}(A, B)$, $\text{pk}(B)$, $\text{inv}(\text{pk}(B))$;

Actions:

A \rightarrow B: { A, $\text{pw}(A, B)$, K } $\text{pk}(B)$

B \rightarrow A: { | NB | }K

Goals:

B authenticates A on K

A authenticates B on NB

K secret between A, B

- This is similar to what happens in TLS-based login protocols.
- Guessing not possible – despite low-entropy $\text{pw}(A, B)$
 - ★ The intruder has to create {A, *guess*, K} $_{\text{pk}(B)}$ for every guess.
 - ▶ That requires guessing K, too!
- This is also why any reasonable implementation of asymmetric encryption contains randomization!
 - ★ {*guessable message*} $_{\text{pk}(B)}$

Guessing Attacks in OFMC

B → A : NB

A → B : $h(pw(A, B), NB)$

Goals:

B authenticates A on NB

$pw(A, B)$ guessable secret between A, B

Gives attack:

GOAL:

guesswhat

...

ATTACK TRACE:

i → (x20, 1) : x205

(x20, 1) → i : $h(pw(x20, x25), x205)$

i → (i, 17) : $h(guessPW, x205)$

i → (i, 17) : $h(guessPW, x205)$

Guessing Attacks in OFMC

Implementation in OFMC:

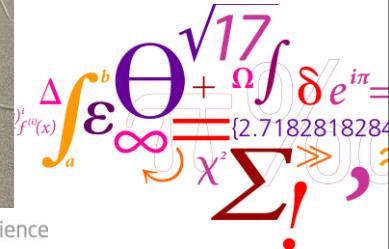
- Say $pw(A, B)$ is specified as a guessable secret between A and B .
- Let $guessPW$ be a constant known to the intruder.
- Check every message sent by an honest agent if it contains $pw(A, B)$.
 - ★ Example: outgoing message $h(pw(A, B), NA)$
 - ★ Then declare $h(guessPW, NA)$ as a secret between A and B .
 - ★ If the intruder is able to violate this secrecy goal, then he can make a guessing attack.
- Additionally, if the password is used for symmetric encryption, then guessing the key is sufficient:
 - ★ Example: $\{m\}_{h(pw(A,B),NA)}$
 - ★ Then also $h(guessPW, NA)$ is a secret between A and B .

Cryptography I



DTU Compute

Department of Applied Mathematics and Computer Science

$$\Delta \int_a^b \varepsilon \Theta + \Omega \int \delta e^{i\pi} = \{2.71828182845904523536028747135266249 \dots\}$$


What Cryptography can do

- Cryptography is just one of the tools in the security tool box
 - But a very versatile and powerful tool
- Cryptography can help solve important problems:
 - Keeping secrets
 - *Messages on the network*
 - *Data stored on disks, memory cards, USB sticks, ...*
 - Ensuring integrity
 - *Messages on the network (Message Authentication Codes)*
 - *Data stored on disk, ...*
 - Authentication
 - *User authentication*
 - Protecting shared secrets during communication
 - *Message authentication*
 - Sender authentication
 - Message authentication

Keeping Secrets

Steganography, Codes and Ciphers

- Steganography

- Hide the existence of a secret message
- Inconspicuous message (invisible ink, micro dots)
- If the message is found, then the secret is revealed



- Codes

- Replace symbols in the message with "codes"
- Transformation must be agreed in advance
- Inconspicuous if code is well defined



- Ciphers

- Hide the meaning of the secret message
- Scrambling according to an agreed algorithm
- Conspicuous message (obviously a secret)



3

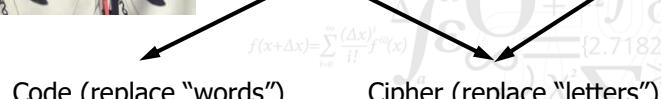
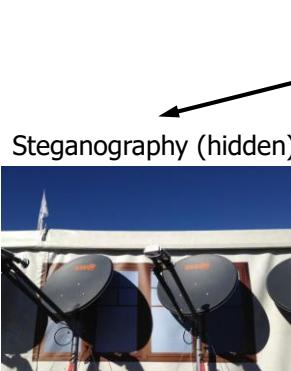
DTU Compute Technical University of Denmark

Master of Cybersecurity

28/8/2021

Cryptography

Fundamental principles



4

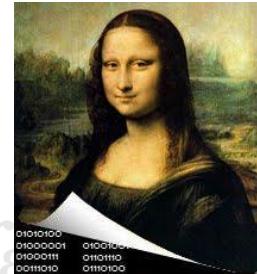
DTU Compute Technical University of Denmark

Master of Cybersecurity

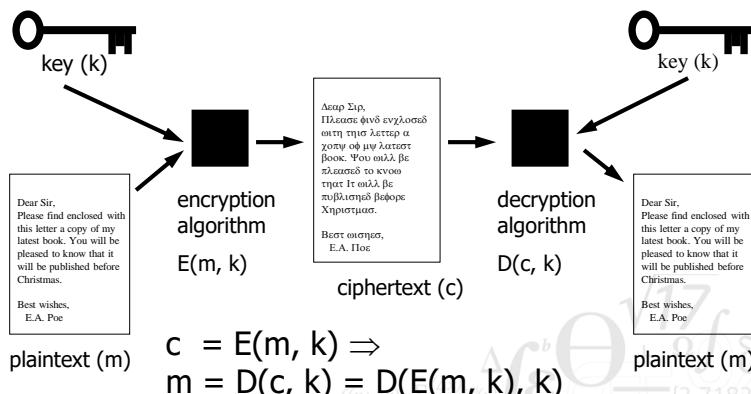
28/8/2021

Steganography

- Classic example from antiquity
- Xerxes' invasion of Greece 480 BC
 - Persian build-up observed by Greek citizen Demaratos
 - Sent a warning to Athens on wax-plates
 - *Scraped of the wax*
 - *Engraved the warning in the wood*
 - *Covered the wood with new wax, so the plates looked unused*
 - *Warning safely reached Athens*
 - Xerxes' navy was defeated at Salamis
- Current methods for steganography now often uses images or sound
 - Message encoded in least significant bit of pixel or sample
 - Cover message (e.g. image) can be uploaded to public forum



Cipher = Algorithm + Key



No cipher should rely on the secrecy of the algorithm!

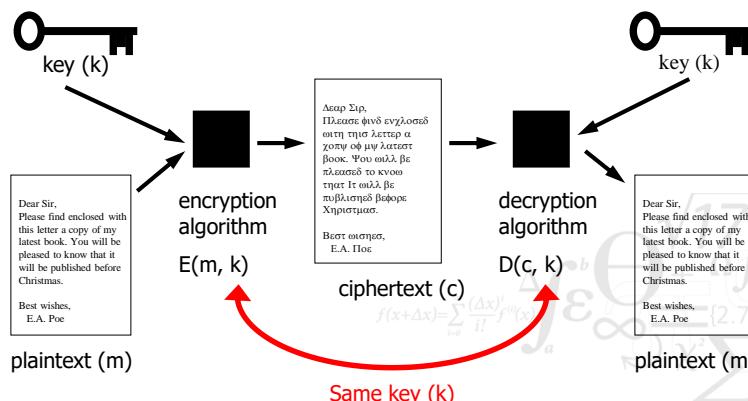
A. Kerckoffs, "La Cryptographie Militaire", 1883

Basic Building Blocks of Cryptography

- Symmetric ciphers (1 key is used)
 - Same key is used for encryption and decryption
- Asymmetric ciphers (2 keys are used)
 - One key is used for encryption
 - Another key is used for decryption
 - *It is not computationally feasible to derive one key from the other*
- Cryptographic Hash Functions (no keys are used)
 - Scrambles input in a unique way
 - Generates fixed length output from variable length input
 - Scrambles input ("decryption" is infeasible)
- Digital signatures
 - One key signs data (private key)
 - Another key is used to verify signature (public key)
- Advanced algorithms, protocols and constructs not covered here

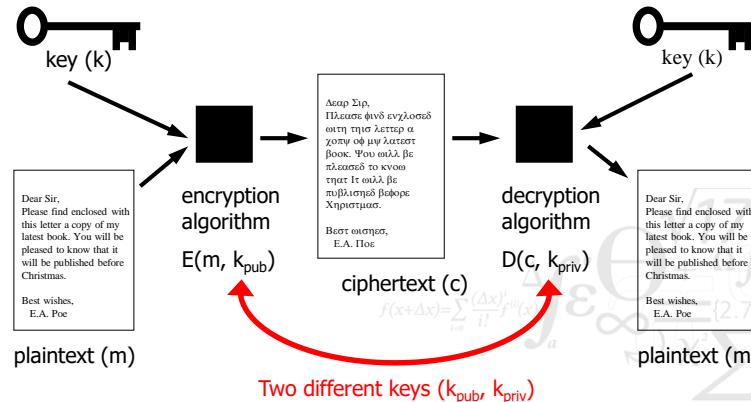
Symmetric Cryptography

- Decryption-key is identical to Encryption-key (or easily derived)



Asymmetric Cryptography

- Decryption-key cannot be derived from Encryption-key



9

DTU Compute Technical University of Denmark

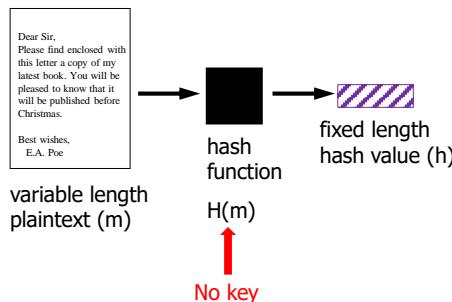
Master of Cybersecurity 28/8/2021

Cryptographic Hash Functions

- Cryptographic hash functions must also satisfy:

- Given M , computation of h is easy
- Given h , it is intractable to compute M such that $H(M) = h$
- It is intractable to find M and M' such that $H(M') = H(M)$

- Hash value h is often called a “fingerprint” or “digest” of M



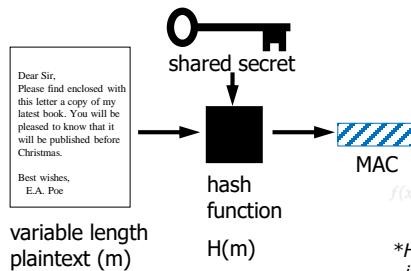
10

DTU Compute Technical University of Denmark

Master of Cybersecurity 28/8/2021

Message Authentication Codes (MAC)

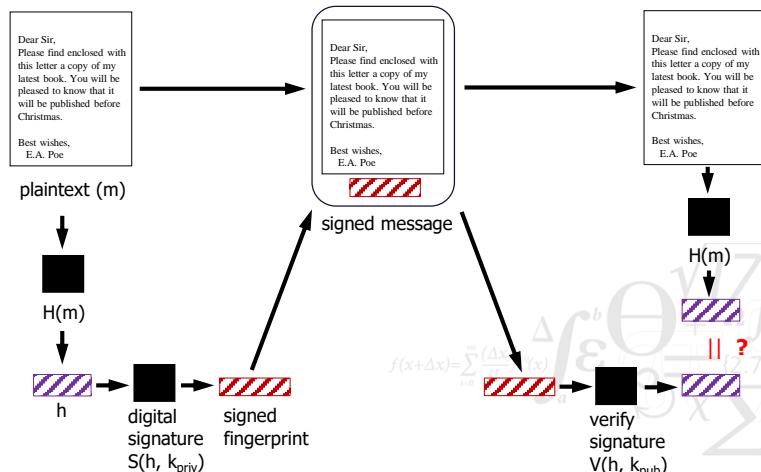
- Message Authentication Codes protect integrity of messages
- Hash-based MAC uses a cryptographic hash function and a shared secret
 - Shared secret (K) is prepended to the message (M) before applying H
 - $\text{MAC}(K, M) = H(K \parallel M \parallel K)^*$



*HMAC is a frequently used MAC function; it is defined in RFC 2104

Digital Signatures

- Operation is expensive, so we normally sign the fingerprint



Security Properties

- Symmetric cryptography
 - Confidentiality of messages
 - Both parties know the shared key (may be more than two)
- Asymmetric cryptography
 - Confidentiality of messages
 - Only one party knows the secret key (only party who can decrypt M)
- Hash functions
 - Hash value corresponds to given M (with very high probability)
- Message Authentication Codes
 - Integrity of message (hash function)
 - Authenticity of message (shared secret)
- Digital signatures
 - Integrity and Authenticity (as with MAC)
 - Non-repudiation of message (assuming security of private-key)

Protection Goals

- Before considering a cryptographic solution, get clear about its protection goals:
 - Confidentiality
 - Integrity
 - Authenticity
 - Non-Repudiation
- People often say: "Our system provides secure communication" or "we need a secure communication channel"
What does this really mean?
- Cryptography is expensive, so we should only use the building blocks that we need

Characteristics of Good Ciphers

- Defined by Claude Shannon (1949)
 - The father of Information Theory



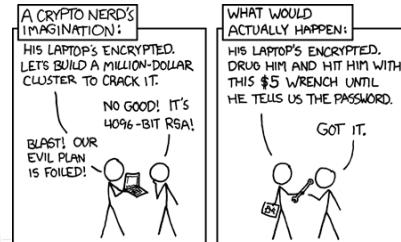
1. The amount of secrecy needed should determine the amount of labour appropriate for the encryption and decryption
2. The set of keys and the enciphering algorithm should be free from complexity
3. The implementation of the process should be as simple as possible
4. Errors in ciphering should not propagate and cause corruption of further information in the message
5. The size of the enciphered text should be no larger than the text of the original message

Security of Cryptographic Solutions

- There are 3 classes of attacks on crypto-systems
 - Attacking the cipher (algorithm and mode)
 - Attacking the key (key space, key generation, key management)
 - Attacking the cryptographic protocol
- Most crypto-systems are "computationally secure"
 - Security often measured in number of operations required by the best known attack (this is known as the "workload" of an attack)
 - *Strong algorithm, brute force attack* \Rightarrow workload = key-length/2

Cryptanalysis attacking ciphers

- Cryptanalysis attempts to recover plaintext without access to key, but with full knowledge of algorithm
- There are four general types of cryptanalytic attacks:
 - Ciphertext-only attack
 - Known-plaintext attack
 - Chosen-plaintext attack
 - Adaptive chosen-plaintext attack
- Other attacks are equally effective
 - Rubber-hose cryptanalysis aka. purchase-key attack



$f(x+\Delta x) =$
<https://xkcd.com/538>

Cryptanalysis attacking keys

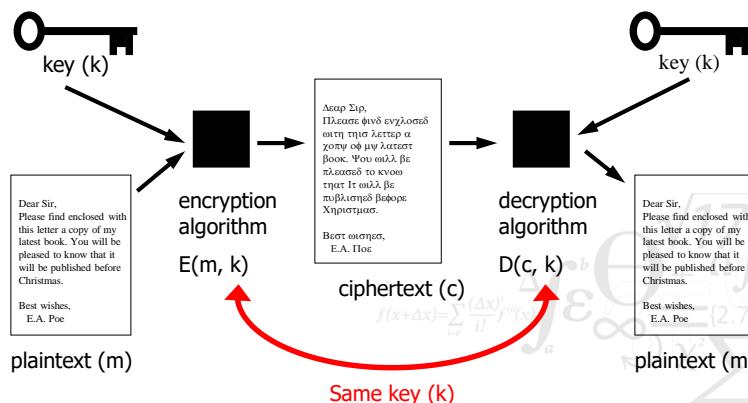
- Key space attacks
 - Exhaustive key search (brute force) attacks are possible with short keys
 - DES has fixed 56 bit keys, which are cracked very quickly
 - AES uses 128, 192 or 256 bit keys (algorithm allows longer keys)
 - Asymmetric algorithm keys are an order of magnitude longer
 - Elliptic Curve crypto. achieves equivalent security with shorter keys
- Key generation attacks
 - 128 bits = 16 random bytes are hard to remember
 - Key derivation functions (KDF) take a password and generates a key
 - Guessing the password \Rightarrow knowing the key
- Key management
 - Keys must be stored, shared, distributed, ...
 - All these steps may be attacked

Coffee Break



Symmetric Cryptography

- Decryption-key is identical to Encryption-key (or easily derived)



One Time Pads

- A One Time Pad consists of a large non-repeating sequence of *truly random* characters
- Encryption xor each letter in the plaintext with the corresponding letter from the one time pad

$$C = P \oplus K$$

- Decryption xor each letter in the ciphertext with the corresponding letter from the one time pad

$$P = C \oplus K$$

- One time pads produce perfectly secure encryption

- Known as information-theoretic security, cryptosystem cannot be broken by attacker with unlimited resources

- *Derived from the work by Claude Shannon*

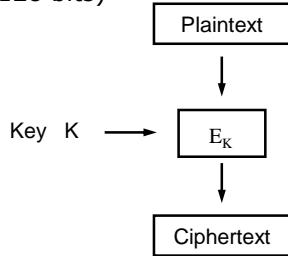


Symmetric Cryptography

- Two main classes of symmetric cryptography:
 - Block Ciphers
 - Streams Ciphers
- Well known Block Ciphers include:
 - DES (badly broken, but found in older textbooks)
 - *Keys are too short to protect against brute force*
 - Triple DES (3DES, still in use, but will be retired by 2023)
 - $C = E_{K3}(D_{K2}(E_{K1}(P)))$
 - AES (current standard adopted by NIST)
- Well known Stream Ciphers include:
 - A5/1 (used in GSM networks, essentially broken)
 - RC4 (difficult to use securely)
 - Block ciphers can be used to construct stream ciphers
 - *This is often the better option*

Block Cipher Algorithms

- Block ciphers operate on blocks of plaintext and ciphertext (usually 32, 64 or 128 bits)



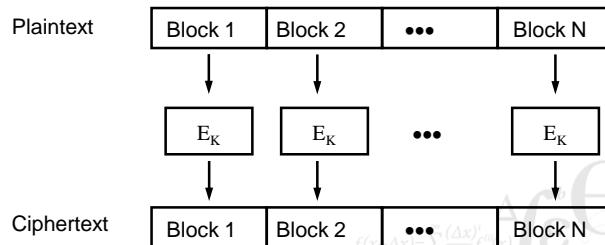
- Algorithm describes how to transform one block of a certain size
 - What happens when plaintext is shorter than block size of algorithm?
 - What happens when plaintext is longer than block size of algorithm?
- Introduces encryption schemes/modes

Algorithms and Schemes/Modes

- Cryptographic algorithms
 - Describe transformation of plaintext/ciphertext to ciphertext/plaintext
 - Do not describe how cryptography is used in systems
- Algorithms operate on fixed sized data
 - Blocks, stream units (bits/bytes)
- What happens when message size does not fit algorithm size?
 - Message is padded to fit algorithm size
- Cryptographic schemes/modes define:
 - How to encrypt plaintexts of arbitrary lengths
 - How to use initialisation vectors to make each encryption unique

Encrypting Long Plaintexts

- Divide the plaintext into blocks of the defined block size
 - Add padding to the end if necessary



Electronic Codebook Mode

- ECB is the simplest application of a symmetric block cipher
 - plaintext blocks are separately encrypted into ciphertext blocks
 - *Figure on the previous slide*
 - the same plaintext block is always encrypted into the same ciphertext block (with the same key)
 - *Substitution of one "letter" for another "letter" => cipher*
 - Size of AES alphabet is 2^{128}
- Properties of ECB encryption
 - blocks can be en-/decrypted in random order
 - *ECB used in encrypted file systems means that the "seek" operation works*
 - Does require block alignment, but disk blocks are typically multiples of 32, 64 or 128 bits (e.g., 512B, 1024B or 4096B)
 - blocks can be en-/decrypted in parallel
 - *Useful in high speed network communication*

Problems with ECB

- Structure in the plaintext is reflected in the ciphertext



Tux



ECB encoding



Non-ECB encoding

- Messages often have stereotyped beginnings (Dear Foo) and endings (Best regards, Bar)
- Cryptanalyst who learn the encryption of a plaintext block can decrypt the corresponding ciphertext block in all messages encrypted with the same key
- ECB is also vulnerable to *block replay attacks*

Block Replay Attack

- Mallory has access to the network
- He deposits money several times in the Receiving bank (and looks for duplicates)
- He deduces the sending banks encoding of his name, account no. and the amount
- He can now modify other people's money transfers

Block number

1	2	3	4	5	6	7	8	9	10	11
Time-stamp	Sending bank	Receiving bank	Account holder's name			Account number	Amount			

Field

NB! You don't need the key to modify the message
Encryption does not protect integrity

Cipher Block Chaining Mode

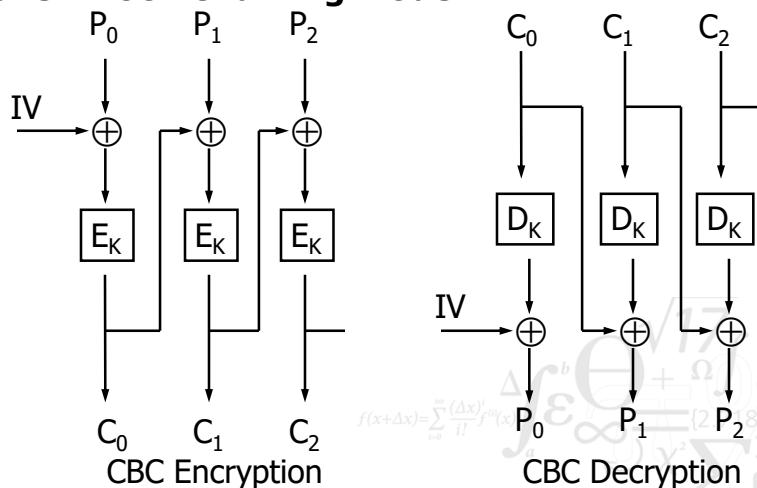
- Introduce a feedback mechanism
 - include the previous ciphertext block in the encryption of the current plaintext block

$$C_i = E_k(P_i \oplus C_{i-1})$$

$$P_i = C_{i-1} \oplus D_k(C_i)$$

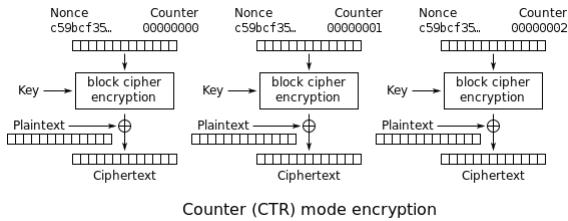
- An initialization vector (IV) is used to start the process (the IV need not be secret, but must not be reused)
 - Nonce, sequence number, date, ...
 - IV may be public, i.e., known to the attacker

Cipher Block Chaining Mode

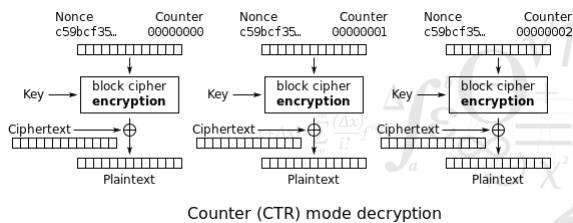


Counter Mode

- Encryption



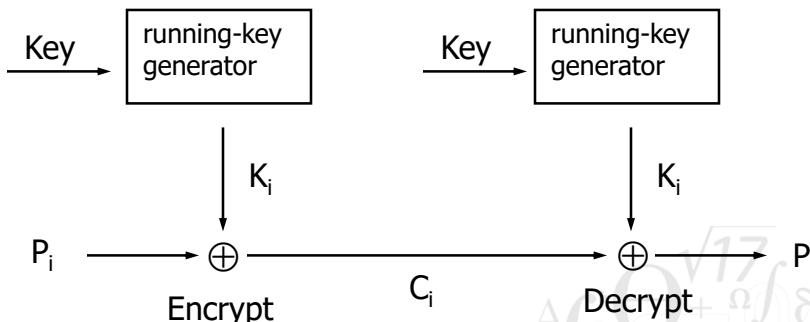
- Decryption



Stream Ciphers Algorithms

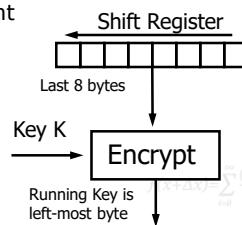
- Stream ciphers work on 1 bit/byte at the time
- A running-key generator generates a pseudo-random string of bits used for encryption
 - same running-key every time means that cryptanalysis is trivial
 - completely random running-key is equivalent to a one time pad (complete security, but not possible)
 - the key is used to seed the running-key generator
- Stream ciphers do not require an entire block to work
 - Good for character based applications

Stream Ciphers II



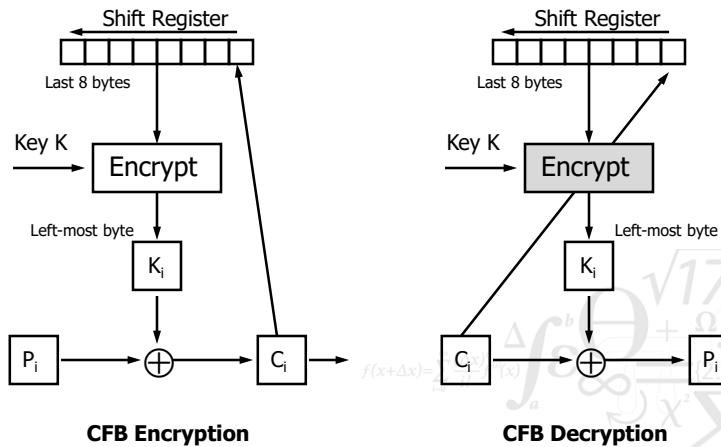
Creating Stream Ciphers from Block Ciphers

- Block ciphers can be used to implement stream ciphers
- Example: a 64bit block cipher is used to implement a byte stream cipher
 - A "shift register" (64bit) is encrypted and the leftmost byte is XORed with the plaintext byte
 - the shift register is shifted 1 byte to the left and the ciphertext byte is added to the right

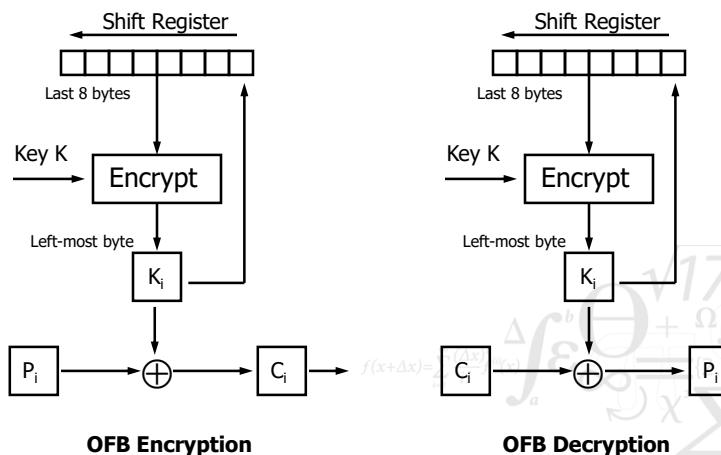


- The IV must be unique, e.g., a serial-number

Cipher Feedback Mode

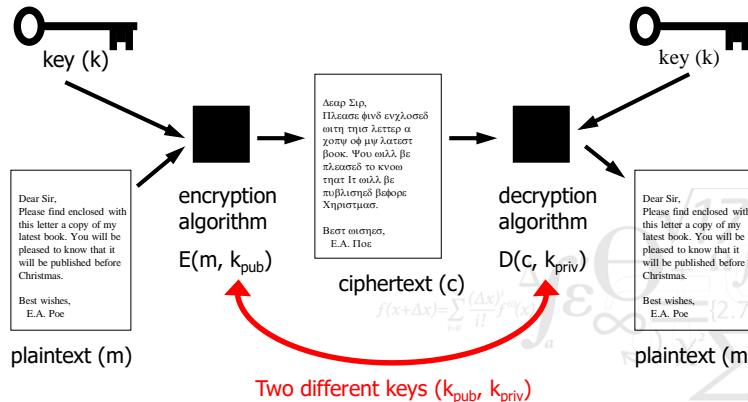


Output Feedback Mode



Asymmetric Cryptography

- Decryption-key cannot be derived from Encryption-key



Limitations

- Compared to symmetric cryptography, asymmetric cryptography is inefficient (slow, long keys)
 - In practice it is not used to encrypt large amounts of data
 - *Building block for key exchange*
 - *Building block for other protocols*

Sample Asymmetric Encryption Algorithms

- Asymmetric encryption is based on computationally hard problems
 - Problems that we cannot solve efficiently on computers
- Important groups of asymmetric encryption algorithms
 - RSA (prime factorization of large numbers)
 - Diffie-Hellman (modular arithmetic)
 - Diffie-Hellman (elliptic curves)
- Asymmetric encryption schemes define how algorithms are used in practice, i.e., arbitrary length messages, padding, etc.
- Particularly important: Semantic Security
Encryption scheme has to randomize the message, since otherwise, the following attacks become possible
 - Guess the message's content (guess M that corresponds to given C)
 - Use (known) public key to check whether the guess is correct

RSA

- Most popular public-key cryptosystem
- 1977: Rivest, Shamir, Adleman (RSA)
- Both encryption and authentication
- Survived years of attacks (cryptanalysis)
 - Probably secure
- Part of many official standards worldwide
 - Interoperability with existing code base



RSA overview

- Based on the difficulty of factorization
- Pick two random large primes: p and q
- Calculate $n = pq$
- Choose random encryption key e
 - e and $(p-1)(q-1)$ must be relatively prime
- Compute decryption key d (extended Euclidian algorithm), such that
$$ed \equiv 1 \pmod{(p-1)(q-1)}$$
- Public key = (e, n) Private key = d

Using RSA

- Encryption of a plaintext M
 - divide M into numerical blocks $m_i < n$
 - $c_i = m_i^e \pmod{n}$
- Decryption of ciphertext block c_i
$$m_i = c_i^d \pmod{n}$$

because

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^1 \pmod{n}$$

ElGamal

- Based on the difficulty of calculating discrete logarithms in a finite field

$$y = g^x \bmod p$$

- p is prime
- g and x are random numbers less than p

- Public key = (y, g, p)
- Private key = x

$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
$$\int_a^b \Theta + \Omega \int \delta e^{i\pi} = [2.718281828]$$
$$\Sigma \gg !$$

ElGamal Encryption

Encryption of message M

- Select random k relatively prime to $p - 1$
 $a = g^k \bmod p$
 $b = y^k M \bmod p$
- The pair (a, b) is the ciphertext

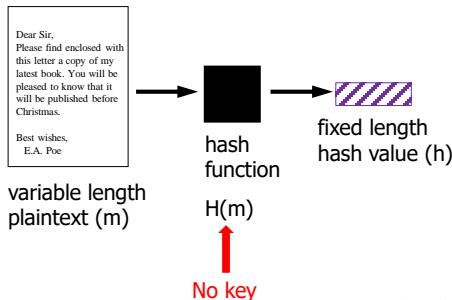
Decryption of a and b

$$M = b/a^x \bmod p$$

because

$$a^x \equiv g^{kx} \pmod{p} \wedge b/a^x \equiv y^k M/a^x \equiv g^{kx} M/g^{kx} \equiv M \quad (\text{all is mod } p)$$

Cryptographic Hash Functions



- Cryptographic hash functions must also satisfy:

1. Given M , computation of h is easy
2. Given h , it is intractable to compute M such that $H(M) = h$
3. It is intractable to find M and M' such that $H(M') = H(M)$

- Hash value h is often called a “fingerprint” or “digest” of M

Hash Functions

- The “work horses” of cryptography
- Main uses include:
 - Condensing long strings into short strings (collision resistant hash function)
 - Making an irreversible transformation without a key (one-way function)
- Prominent Examples:
 - MD4, MD5, SHA-0 (badly broken)
 - SHA-1 (still found in standards, but practically broken since 2020)
 - SHA-2 (current standard)
 - SHA-3 (newest standard – since 2015)

Collision Resistance

- Intractable to find random M and M' such that

$$H(M) = H(M')$$

- Collisions invalidates the use of hash functions in digital signatures
 - Alice creates two contracts M and M' , where M is fair, but M' is favourable to the Alice
 - Bob signs M : $\text{Sign}(H(M), \text{Bob}_{\text{Priv}})$
 - Since $H(M) = H(M')$ Alice can present M' as if it was signed by Bob

Birthday Paradox

- Standard statistics problem

How many people do you need in a room to have better than 50% chance of two persons with the same birthday?

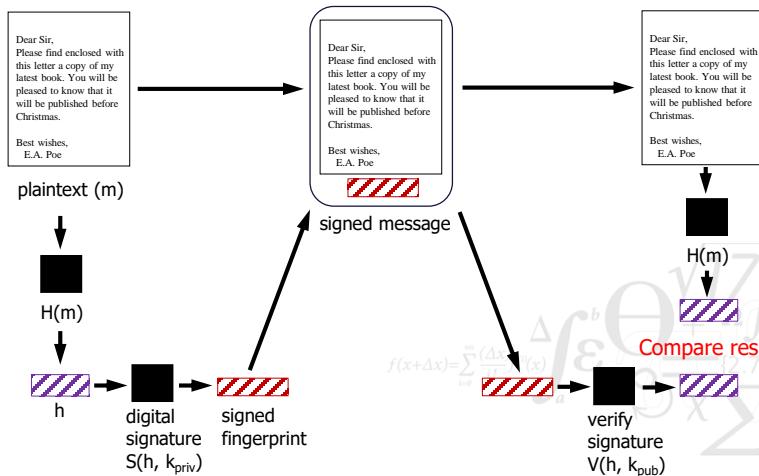
- someone with your birthday 253
- any two with the same birthday 23
- Any two born of the same day of the month 10

- m bit hash function

- 2^m random hashes needed to find a particular h
- $2^{m/2}$ random hashes needed to find two messages with the same hash value

Digital Signatures

- Operation is expensive, so we normally sign the fingerprint



Security Level

- If the best known attack is equivalent to running the cryptographic algorithm 2^n times, then we have a security level of n bit
- Typical 80 bit (too low) 128 bit (decent), 256 bit (high)
- Often, the security level is equal to the key length
- Important exceptions:
 - Hash functions (hash size $\geq 2 \times$ security level)
 - Asymmetric cryptography (e.g. RSA: 1024/2048/4096 bit)

Summary on Building Blocks

- Please remember the following
 - Clarify your security goals
 - Don't design your own cryptographic primitives and protocols
 - Always rely on well known (and analysed) standards
 - Make sure that you understand the primitives and protocols you use:
 - Security goals
 - Security level
 - How to apply them
 - Known problems and pitfalls

$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
$$\Delta \int_a^b \Theta^{17} + \Omega \int \delta e^{i\pi} =$$
$$\mathcal{E}_{\infty} = [2.718281828]$$
$$\sum x^2 \gg !,$$



Cryptography II



DTU Compute

Department of Applied Mathematics and Computer Science

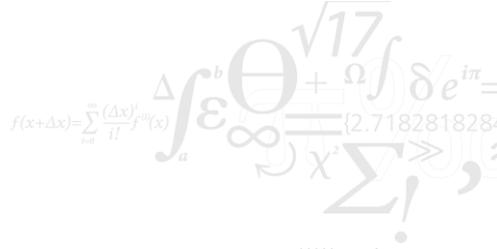
$$\Delta \int_a^b \varepsilon^{\sqrt{17}} \Theta + \Omega \int \delta e^{i\pi} = f^{(0)}(x) \infty = \{2.71828182845904523536028747135266249 \dots\}$$

Encrypted Communication

- A protocol defines a sequence of steps involving several parties
- Characteristics of a protocol
 - Everyone knows the protocol in advance
 - Everyone agrees to follow the protocol
 - The protocol must be unambiguous
 - The protocol must be complete
- Typical “actors” in description of cryptographic protocols
 - Alice and Bob (parties who wish to communicate securely)
 - *Charlie and Dave are often added for multiparty protocols*
 - Eve (a malicious agent who wishes to eavesdrop on communication)
 - Mallory (a malicious attacker of any kind)
 - Trent (a trusted arbitrator – Trusted Third Party (TTP))

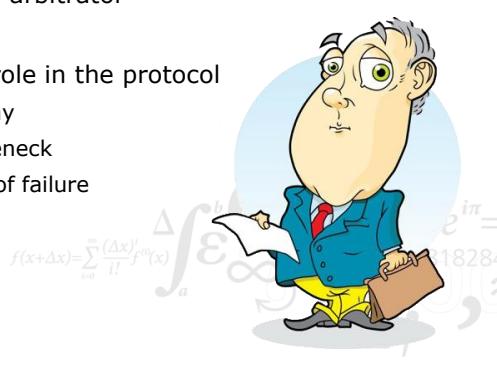
Cryptographic Protocols

- Cryptographic protocols
 - Describe how multiple parties employ cryptography together
 - *How each party uses the algorithms*
 - *How cryptographic keys are managed*
 - Key generation (not always covered in the protocol description)
 - Key distribution
- Three types of protocols
 - Arbitrated protocols
 - Adjudicated protocols
 - Self-enforcing protocols



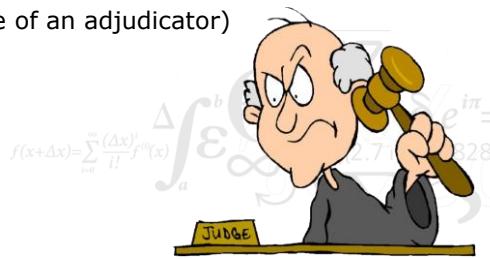
Arbitrated Protocols

- An arbitrator is an uninterested trusted third party (e.g., lawyers, notaries, banks)
- All parties must trust the same arbitrator
- The arbitrator plays an active role in the protocol
 - the arbitrator introduces a delay
 - the arbitrator becomes a bottleneck
 - the arbitrator is a single point of failure



Adjudicated Protocols

- The costs of arbitration can be reduced by subdividing the protocol in two
 - 1) A non-arbitrated protocol to do transactions
 - 2) An arbitrated protocol to resolve disputes
- The non-arbitrated protocol must provide non-disputable evidence of the transaction
- The adjudicator is only invoked in case of disputes (a judge in the legal system is a good example of an adjudicator)



5 DTU Compute Technical University of Denmark

02239 Data Security

Self-Enforcing Protocols

- The protocol itself guarantees fairness
- It must be designed so that there cannot be any disputes
- Attempts to cheat must be detected before damage is done and the protocol stopped
- Self-enforcing protocols are preferred whenever possible
- Do you know examples of self-enforcing protocols?

The Enforcer



6 DTU Compute Technical University of Denmark

02239 Data Security

Encrypted Communication in Practice

- Handshake (taking the initial steps)
 - Agree on cryptographic protocol, including:
 - Algorithms and Modes
 - Cryptographic session keys
 - Authentication of Communicating Parties
 - Single-side authentication (typically server)
 - Mutual authentication (both parties are authenticated)
- Communication (steady state operation)
 - Send and receive messages securely
 - Confidentiality protected through encryption
 - Integrity protected through MAC or digital signatures
- Connection termination
 - Delete session specific state (keys, cookies, ...)

Usage of Cryptographic Keys

- The following is a categorisation of cryptographic keys according to what they are used for:
 - **Data key:** Directly used for cryptographic purposes, e.g. encryption or authentication
 - **Key-encryption key:** Used to encrypt other keys, e.g. in key exchange or key storage
 - **Master key:** Used to generate other keys, e.g. in key derivation function (KDF).

Example: $\text{Session_Key} := \text{KDF}(\text{Master_Key}, \text{Session_Number})$

Cryptographic Key Management

- Key Generation
- Key Distribution
 - Key Exchange
 - *One party generates the key*
 - Requires key transport to all other parties
 - Key Agreement
 - *All parties influence the generation of the key*
 - Diffie-Hellman algorithm popular for two parties
- Perfect Forward Secrecy (PFS)
 - Ensures that a session key derived from a set of long-term keys cannot be compromised if one of the long-term keys is compromised in the future

Key Generation

- Cryptographic keys must be intractable to guess
 - All possible cryptographic keys should be equally likely
- Typical key generation methods include:
 - Password Based Key Derivation Function
 - *generates a cryptographic key from a readable string*
 - Random number generator (RNG)
 - *Statistical RNG, not cryptographically secure*
Never use this for Cryptographic purposes!
 - *Pseudo-random number generator (PRNG)*
Must be seeded correctly, so use with care
 - *True random number generator*
Uses measurements of physical processes to generate "real"
randomness - too expensive for most applications

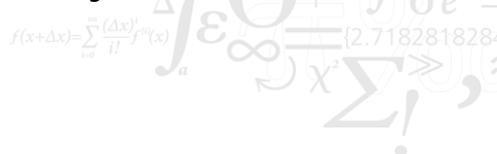
True random numbers are available online: <https://www.random.org>

Key Exchange Requirements

- In addition to being generated, the key must also be distributed to all legitimate parties
 - How to prevent others from seeing the key?
 - How to authenticate the legitimate parties (sender and receiver)?
 - How to distribute the key to the legitimate parties?
 - How to ensure that the key is fresh?
 - How to verify that the legitimate parties received the key?

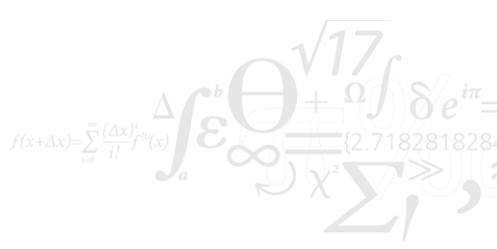
If done remotely: Use cryptography (many different solutions)

Sometimes easier: Personal key exchange

$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$


Key Exchange Techniques

- For key exchange we have the following options:
 - Generated by all parties working together (key agreement)
 - Generated by one party, then published in a public place Public Key Infrastructure (PKI)
 - Generated by one party, then sent to the other (key transport)
 - Generated by a trusted third party and sent to all parties TTP key transport

$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$


Key Expiration

- Keys can (in fact: should) expire sometime. Problems include:
 - How to keep track of key expiration?
 - How to inform all users of key expiration?
 - How to set up a new key?
 - What happens after expiration?
 - *Archive old key material? How?*
 - *Delete old key material? How? Remember to delete all copies!*

Key Compromise

- Worst case: Key has been compromised because
 1. An attacker has potentially had access to the key
 2. The corresponding cryptographic algorithm was broken
- What do we have to do?
 - Key must no longer be used in the future
 - *Key expiration (see above)*
 - All concerned parties have to be informed!
 - *Key Revocation*
 - Old data has to be protected
 - *Re-encryption? Re-signing?*
 - *Destruction of all copies of old data*

Symmetric-Key Cryptography Protocol

- The following steps are required to setup communication using SKC
 - 1) Alice and Bob agree on a cryptosystem
 - 2) Alice and Bob agree on a key
 - 3) Alice encrypts her plaintext with the key
 - 4) Alice sends the ciphertext to Bob
 - 5) Bob decrypts the ciphertext with the key

How can these steps be compromised?

$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$

Problems of SKC

- Keys must be distributed in advance
- Keys must be kept secret from non-participants in the protocol
- A compromised key reveals all information encrypted with that key
 - Keys can be compromised:
 - Weak keys (e.g. PRNG or KDF is weak) can be guessed
 - Man In The Middle (MITM)
 - Compromising hosts
 - Probability of a host is compromised is p , probability of their shared key is compromised is $2p$
- Both parties have the same key
 - A compromised key can be used to masquerade as either party
 - SKC cannot be used directly in adjudicated protocols
- Different keys are needed for every pair of communicating parties
 - n users require $n(n - 1)/2$ keys

Key-Exchange Protocols

- Assume a Key Distribution Center (KDC)
- Assume all participants share a key with KDC
- Shared-Key (symmetric) Protocols
 - Alice → KDC, request session key to talk with Bob
 - KDC generates Key, returns $K_A(K_{AB})$, $K_B(K_{AB})$ to Alice
 - Alice decrypts $K_A(K_{AB})$ and sends $K_B(K_{AB})$ to Bob
 - Bob decrypts $K_B(K_{AB})$, both now know K_{AB}
- Public-Key (asymmetric) Protocols
 - Alice gets Bob's public-key from KDC
 - Alice generates random session-key K and sends $K_{pub_B}(K)$ to Bob
 - Bob decrypts $K_{priv_B}(K)$, both now know K

Both protocols are vulnerable to "man in the middle" attacks

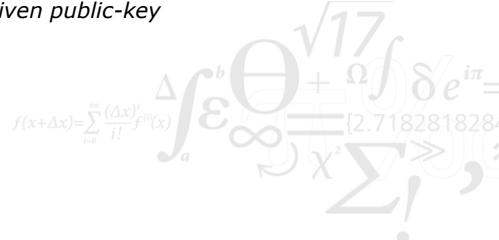
Public-Key Cryptography Protocol

- The following steps are required to setup communication with PKC
 - 1) Alice and Bob agree on cryptosystem
 - 2) Bob sends Alice his public key
 - 3) Alice encrypts plaintext with Bobs public key
 - 4) Alice sends ciphertext to Bob
 - 5) Bob decrypts ciphertext using his secret key

How can these steps be compromised?

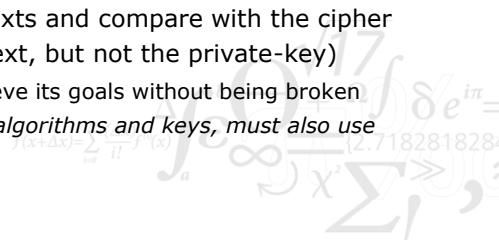
Public-Key Distribution

- Public-key distribution must address:
 - Authenticity (Certification) of public-keys
 - *Linking public-keys to named identities*
 - Distribution of public-keys
 - *Obtaining somebody else's public-key*
 - *Distribute own public-key*
 - Revocation of public-keys
 - *Revoking published keys*
 - *Determining validity of a given public-key*



Hybrid Cryptosystems

- Asymmetric cryptography is 3 orders of a magnitude slower than symmetric
- Asymmetric cryptography is vulnerable to chosen plaintext attacks
- If $C = E(P)$, where P is plaintext with entropy n , a cryptanalyst can encrypt all possible plaintexts and compare with the cipher text (thus he learns the plaintext, but not the private-key)
 - The cryptosystem fails to achieve its goals without being broken
 - *Not enough to use strong algorithms and keys, must also use them in a meaningful way*



Hybrid Cryptosystems II

- Asymmetric cryptography can be used to encrypt a randomly generated symmetric key (session key)
- Advantages
 - Randomly chosen symmetric key has entropy close to the key-size
 - The symmetric key is short (compared to the encryption key)
 - *Encrypting symmetric key is shorter than encrypting message*
 - The encrypted symmetric key reveals little about the asymmetric key

Hybrid Cryptosystems III

- The following steps are required to setup communication with hybrid cryptography
 - 1) Bob sends Alice his public key
 - 2) Alice generates a random session key, K
 - 3) Alice encrypts K with Bobs public key
 - 4) Alice sends the encrypted key to Bob
 - 5) Bob decrypts K with his secret key
 - 6) Alice and Bob exchanges messages encrypted with the session key

These steps still require authenticity of public-key and authentication of end points

Coffee Break



23 DTU Compute Technical University of Denmark

02239 Data Security

Public Key Distribution

- Public-key cryptography simplifies key distribution
 - secrecy of the encryption key is not required
- Authentication of Bob's public-key is required
 - In-Band (online)
 - *Public Key Infrastructures*
 - Key Distribution Centers (KDC)/Certificate Authorities (CA)
 - Out-of-Band
 - *Build into product*
 - *Published in Sunday Newspaper every week*

24 DTU Compute Technical University of Denmark

02239 Data Security

Certification Authorities

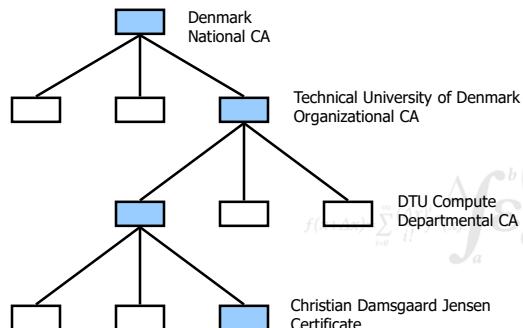
- A certification authority (CA) guarantees that the key belongs to the named principal
- A principal can be:
 - A user
 - An attribute of the user (e.g., her role in within an organisation)
 - An organisation (e.g., a company or another CA)
 - A pseudonym
 - A piece of hardware/software
- Some CAs only allow certain types of principals

Obtaining a Certificate from a CA

- Alice wishes to obtain a certificate from Charlie the CA
- 1. Alice generates a public-/private-key pair and signs the public-key and identification information with the private-key
 - *Proves that Alice knows the private-key*
 - *Protects public-key and identification information in transit to CA*
- 2. CA verifies Alice's signature on the public-key and her ID
 - *Verification may be done out-of-band*
 - Email/phone callback
 - Business/Credit bureau records, in-house records
- 3. CA signs Alice's public-key and ID with the CA private-key
 - *Creating a certificate that binds Alice's public-key and her ID*
- 4. Alice verifies the public-key, ID and CA's signature
 - *Proves that CA didn't substitute Alice's public-key*
 - *Protection of the certificate in transit from CA*
- 5. Alice and/or CA publishes the certificate

PKI

- Certificate Authorities organized in a hierarchy
 - Only top-level CA's certificate needs to be known by everyone
 - Intermediate CAs have certificates signed by "superiors"
 - Certificate verification is done by verifying certificates from the principal towards the top-level CA (aka. "Root CA")

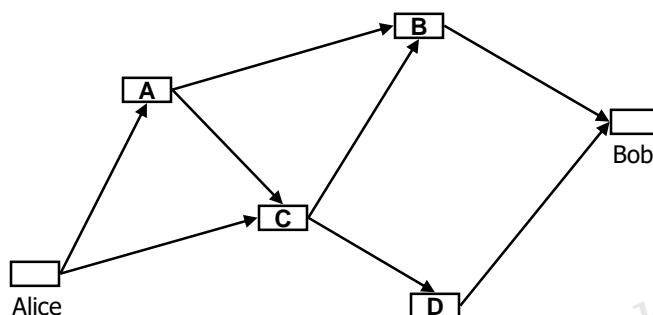


27

DTU Compute Technical University of Denmark

02239 Data Security

Alternatives Trust Hierarchies



- Bob knows B and D who know A and C who know Alice \Rightarrow Bob knows the key came from Alice
- Web of trust closer to human notions of trust
 - This is the method used in PGP (and GPG)

28

DTU Compute Technical University of Denmark

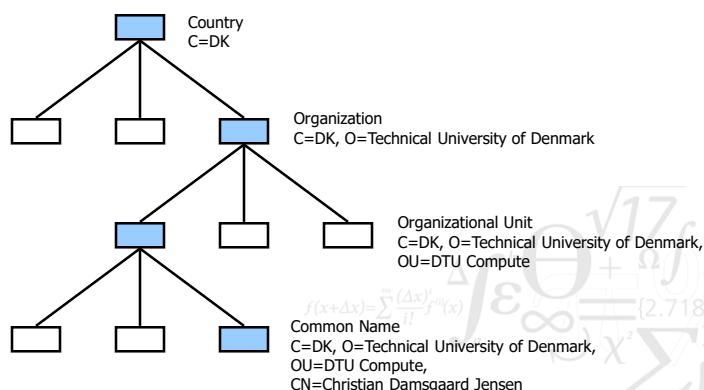
02239 Data Security

What's in a Name

- Alice uses a PKI to find Bob's certificate
 - Which PKI?
 - Which Bob?
- Names are context dependant
 - Bob is personally known by everyone in the village
 - *People have many names: Robert Johnson, Big Bob the sheriff, ...*
 - There are many people called Bob (or Robert) in the town
 - *People may not know that Robert Johnson = Big Bob*
 - The ASCII string "Bob" loses meaning in the big city
- Qualifiers can be added to names
 - Passport number
 - Central Person Register (CPR) number
- Are we looking for Bob123 or Bob421?

X.500 Naming

- X.500 defines Distinguished Names (DN) that uniquely names everything on earth



X.500 Distinguished Names

- Typical DN components
 - Country (C)
 - State or Province (SP)
 - Locality (L)
 - Organisation (O)
 - Organisational Unit (OU)
 - Common Name (CN)
- Problems with X.500 Distinguished Names
 - No rules for how the naming hierarchy should be organised
 - Hierarchical naming only works in clearly hierarchical contexts
 - *Governments, national telecoms providers, ...*
 - *Cannot accommodate nomadic people, stateless people, people with dual citizenship, ...*

What's in a Certificate

- A typical certificate contains
 - Public-key (e.g., 2048bit RSA key)
 - Identification (e.g., X.500 DN)
 - Validity
 - *Not valid before, not valid after*
 - Issuer identification
 - *Used to establish certification path back to root CA*
 - Issuer signature
- Extensions may qualify the certificate
 - Certificate can only be used for certain purposes
 - *Especially important for CA certificates*
 - Establish the domain of authority for the CA

Authority of a CA

- What is the root of authority for a CA?
 - TLS certificate binds a public-key to a business' web-server
 - CA has no authority to register businesses
 - CA has no authority to register domain names
- How do we get the CA root certificate?
 - Built into most browsers
 - Firefox comes with around 100 root-certificates pre-installed, incl.:
 - Deutsche Telekom AG
 - Digicert
 - Entrust, Inc.
 - TDC OCES CA
 - TÜRKTRUST Bilgi İletişim ve Bilişim
 - Verisign, Inc.



33 DTU Compute Technical University of Denmark

02239 Data Security

Trust in PKI

- Root CA can compromise the security of everyone
 - Root CA must be infallible
 - No single authority in the world is trusted by everyone
- Trust in root CA is diluted by the certification path
 - Problem exposed by Uli Maurer's model
 - 90% in CA gives 60% trust in certificate with 5 levels of CAs
- Adding attributes magnifies this problem
 - Credential containing permissions to access a particular resource lends authority from a root CA who knows neither principal nor resource
 - Further down the credential chain permissions become more specific, but the authority of the root CA more diluted



34 DTU Compute Technical University of Denmark

02239 Data Security

Certificate Revocation

- Certificates must be revoked whenever (*not if*) a private-key is compromised
- Revocation is measured by:
 - Speed of revocation: maximum delay from the compromise is discovered and the last use of the certificate?
 - Reliability of revocation: is it acceptable that someone sometimes may not have learned about the revocation?
 - Number of revocations: how many revocations can the system handle at the time?
- Revocation is either
 - Automatic, e.g., using short expiration times
 - Manual, e.g., using Certificate Revocation Lists (CRL)

Applications of Asymmetric Cryptography

- Asymmetric cryptography has many interesting applications
 - Blind signatures
 - Zero-Knowledge (ZK) protocols
 - Electronic voting systems
- We'll examine some of these in the following

Digital Signatures with RSA

- Some public-key crypto-systems allow either key to be used for encryption where the other key is used for decryption (e.g. RSA)
- The basic protocol is:
 - 1) Alice encrypts the message with her private key, this effectively signs the message
 $\text{sign}(m) = m^d \pmod{n} = m' \text{ (all mod } n\text{)}$
 - 2) Alice sends the encrypted message to Bob
 - 3) Bob decrypts the message with Alice's public key, this verifies Alice's signature
 $\text{verify}(m') = m'^e = (m^d)^e = m^{ed} = m \text{ (all mod } n\text{)}$
- Asymmetric encryption is slow
 - Normally sign a hash of the message instead of the message itself

Completely Blind Signatures

- Notaries don't always need to know what they sign
- Alice has a document (M) that she wishes Bob to sign using RSA
 - Bob should not learn the content of M
- 1) Alice picks a random value X and multiplies the document by X^e
(e is Bob's public-key and X is called a blinding factor)
$$M' = M \times X^e$$
- 2) Alice sends the blinded document to Bob
- 3) Bob signs the blinded document
$$M'' = M'^d = (M \times X^e)^d = M^d \times X^{ed} = M^d \times X$$
- 4) Alice divides out the blinding factor (X), leaving the original document signed by Bob (M^d)

This requires signature and multiplication to be commutative

Completely Blind Signatures II

- If Alice chooses a true random value, Bob cannot learn anything about the document
- There is no correlation between the two signed documents in step 3 and 4
- Even if Bob records all the blinded documents that he signs, he will be unable to correlate the two
- The signature is valid and can be verified in the normal way

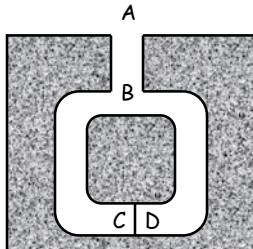
Blind Signatures

- Completely blind signatures are of limited use
- Blind signatures reassures the signer that he does not sign something bad, without revealing the precise content
 - Alice wish Bob to sign one of n "synonymous" documents
 - Alice choose n different random blinding factors
 - Alice blinds the documents and sends all n documents to Bob
 - Bob asks Alice for $n-1$ of the blinding factors
 - Bob verifies $n-1$ copies and signs the n^{th} copy
- In order to cheat Alice must guess which blinded copy will be signed
- This "cut and choose" principle is also used to generate "zero-knowledge" proofs
 - A proves to B that she knows secret s without revealing s

Zero Knowledge Proofs

the cave allegory

- How to prove that you know a secret without telling



How can Alice prove
that she knows the
secret password?

Alice goes into the cave, Bob moves to point B,
Bob calls what passage Alice should exit from,
Alice goes through the door (if necessary)
Alice exits from the correct passage

Zero Knowledge Proofs

basic protocol

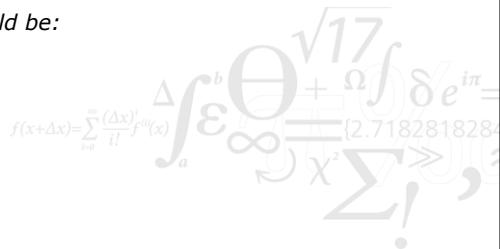
- Assume Alice knows the solution to a hard problem
 - 1) Alice uses a random number to transform her problem into another hard problem, isomorphic to the first problem, she is able to solve the new problem using the solution to the original problem
 - 2) Alice commits to the new problem
 - 3) Alice reveals the new problem to Bob
 - 4) Bob asks Alice to :
 - a) prove that the two problems are isomorphic
 - b) reveal the solution (from 2)
 - 5) Alice complies to the request
 - 6) Alice and Bob repeats the protocol until Bob is satisfied Alice knows the secret
- This is essentially the “cut and choose” principle often used to divide a cake between two siblings

Secure Elections

- Requirements for elections on the Internet
 - 1) Only authorized voters can vote
 - 2) No one can vote more than once
 - 3) No one can determine for whom someone else voted
 - 4) No one can duplicate anyone else's vote
 - 5) No one can change anyone else's vote
 - 6) Every voter can make sure that his vote is counted

an additional requirement could be:

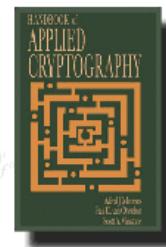
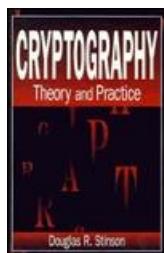
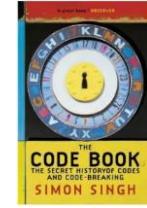
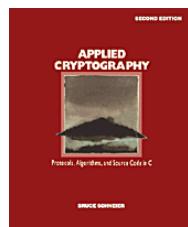
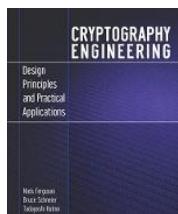
- 7) Everyone knows who voted

$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
A faint watermark in the background of the slide contains various mathematical symbols and equations, including $\sqrt{17}$, f , Θ , Ω , $\delta e^{i\pi}$, Σ , ∞ , x^2 , and exclamation marks.

Voting with Blind Signatures

- 1) The voter creates 10 sets of votes, each set contains one vote for every possible outcome (e.g., two votes for a referendum) and a large random number
- 2) The voter blinds all sets of votes to the polling station
- 3) The polling station verifies that the voter hasn't voted before, requests the blinding factor for 9 of the 10 sets, opens and verifies the 9 sets and signs the 10th set of votes, returning it to the voter
- 4) The voter unblinds the votes and selects the appropriate vote
- 5) The voter encrypts his unblinded vote with the public key of the voting station
- 6) The voter sends the encrypted vote to the polling station
- 7) The polling station decrypts the vote, checks both the signature and that the serial number has not been received before (i.e., that only one vote from the set has been received) and registers the vote, the serial is made public so the voter can know that his vote is counted

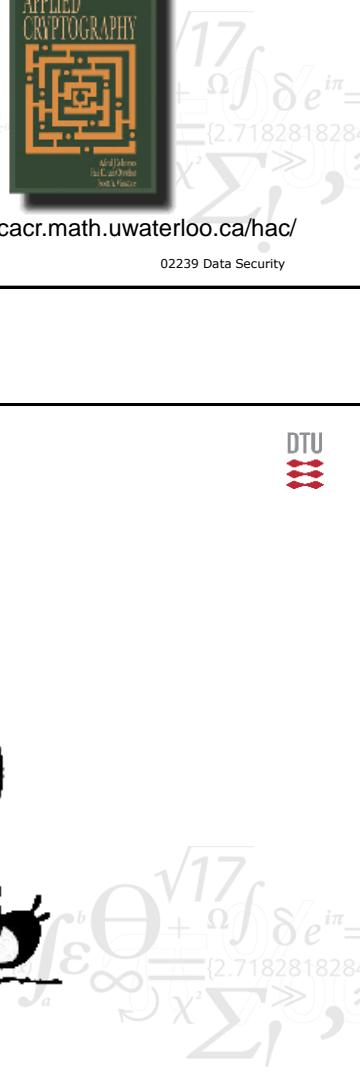
To learn more



<http://www.cacr.math.uwaterloo.ca/hac/>

45 DTU Compute Technical University of Denmark

02239 Data Security



46 DTU Compute Technical University of Denmark

02239 Data Security

Security in Networks

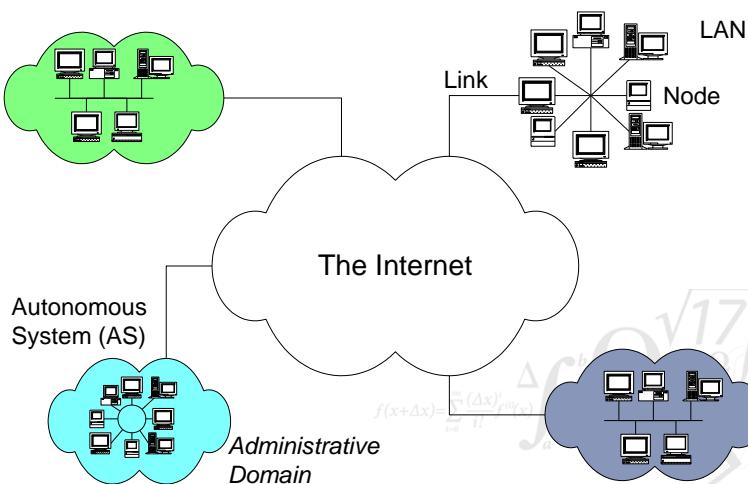


$$\begin{aligned} f(x+\Delta x) &= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x) \\ \int_a^b \mathcal{E}^{\Theta} \Omega^{\sqrt{17}} \delta e^{i\pi} &= \{2.7182818284 \\ \infty \cup x^2 \gg , \sum ! \end{aligned}$$

DTU Compute

Department of Applied Mathematics and Computer Science

Network Definitions



Network Characteristics

- Networks are defined by:
 - Boundary
Distinguishes nodes inside the network from nodes outside the network. It is theoretically possible to make a list of all components belonging to the network
 - Ownership (Autonomous System/Administrative Domain)
Nodes belonging to the same owner and managed by the same administrators. Ownership is sometimes difficult to determine, e.g., who owns the Internet?
 - Control (physical security)
Not all nodes belonging to the network are under control of the administrator, e.g., your own laptop connected to DTU's wireless network, Bring Your Own Device (BYOD) in many companies.

Types of Networks

- 2 types of networks:
 - Shared medium (Bus, Token Ring, Mesh, ...)
 - Point-to-point networks
- Classification based on diameter:

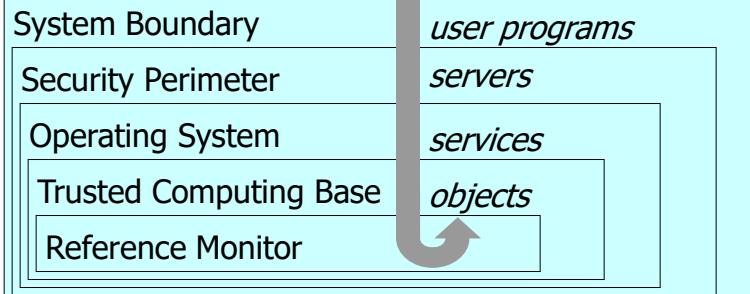
1 m	System
10 m	Room
100 m	Building
1 km	Campus
10 km	City
100 km	Country
1,000 km	Continent
10,000 km	Planet

Multi-processor
Device connection, Body Area Networks
LAN
Metropolitan Area Network (MAN)
WAN
The Internet

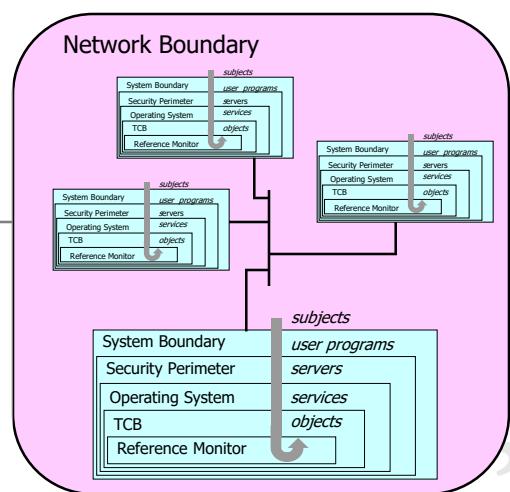
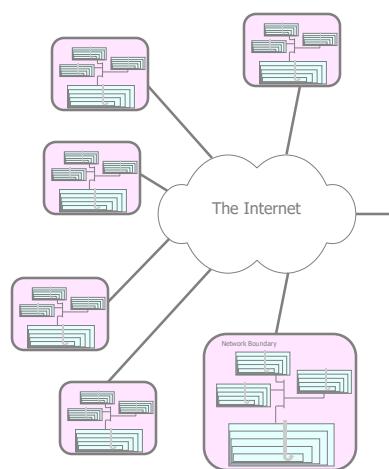
Stand Alone Systems

users, input devices, output devices,...

subjects



Networked Systems

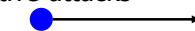


Network Vulnerabilities

- Programming mistakes
 - Public services must handle active and passive data from untrusted sources
 - *Buffer overflows, malicious applets, ...*
- Protocol Flaws
 - Internet was designed for a few hundred trusted computers
 - *This is evident in the lack of security in many protocols (SMTP)*
 - Protocols are often complex and specifications are ambiguous
- Misconfiguration of software and systems
 - Complexity of systems and focus on functional requirements (short deadlines) leads to misconfigured systems
- Unpatched software
 - Vendors normally issue patches when vulnerabilities are exposed, responsibility of customers to apply patches

Network Attacks

Active attacks



Normal communication



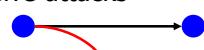
Deletion

Modification



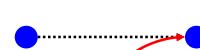
Modification

Passive attacks



Eaves dropping

Fabrication



Fabrication

Traffic analysis



Traffic analysis

Prevent – Detect – React (– Recover)

- Network attack prevention
 - Firewalls - *perimeter security*
 - Identity and Access Management (IAM) – *network access authorisation*
 - Segmentation - *compartmentalisation + security in depth*
 - Virtual Private Networks (VPN) – *secure communication*
- Network attack detection
 - Intrusion Detection Systems
 - Honey Pots
 - Security Information and Event Management (SIEM)
- Network attack reaction
 - Systems-/Network Operation Centres (SOC/NOC)
 - Contingency-/Disaster plans
 - *Procedures described in playbooks or runbooks*

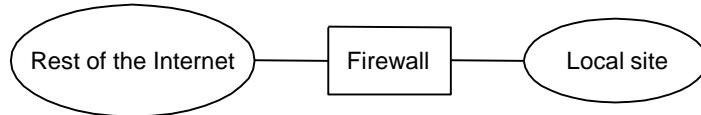
Perimeter Security

- Defending the Network Boundary
 - Keep unwanted outsiders away from valuable resources on the network
 - Don't want outsiders coming in (except for special web-services)
 - *Implies filtering of traffic*
 - Allow insiders to go out
 - *Network Address Translation (NAT) may suffice*
- Filtering traffic at the network boundary (corporate firewalls)
 - Network wide filtering
 - Configured/managed by system administrators
 - Transparent to individual nodes/users
- Filtering traffic at each individual node (personal firewalls)
 - Host-based policy (locally or globally defined)
 - May allow local policy override
 - May require intervention by local users

Firewalls

- A firewall filters information that is sent and received from outside the network
 - It is software that resides on the network's gateway
- A firewall can filter
 - data packets, examining the source & destination IP
 - ports and applications, in order to deny access
 - *May include time of day in decision*
 - Content (payload), such as macros, inappropriate web content, ...

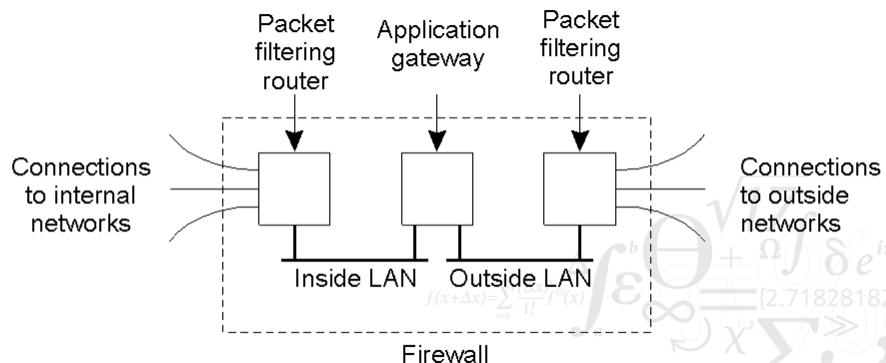
Filtering Firewalls



- Packet-Filter-Based Solution
 - example
 - (192.12.13.14, 1234, 128.7.6.5, 80)
 - (*, *, 128.7.6.5, 80)
 - default: forward or not forward?
 - how dynamic?

Firewalls

- A common implementation of a firewall



Impersonation Attacks falsely obtain valid authentication details

- Guess identity and authentication details (login, password)
 - Try popular logins (guest, admin) and passwords (secret, password)
- Obtain identity and authentication details by eavesdropping
 - `telnet` sends login/password over the wire in the clear
 - Bad encryption is sometimes used (MS LAN manager, WEP)
- Circumvent/disable authentication
 - Exploit buffer overflows
 - Social engineering ("I forgot my password, please reset to BANANA")
- Use a target that will not be authenticated
 - Remote authentication from "trusted hosts" is sometimes disabled
 - Public accounts (anonymous FTP, GUEST accounts)
- Use a target whose authentication details are well known
 - Standard accounts with default passwords
You can sometimes find these in the manuals



Spoofing falsify authentication details

- Masquerading
 - Falsify DNS entries, choose similar hostnames, ...
 - *www.whitehouse.com used to be a porn site*
 - *www.whitehouse.net is a spoof site against George W. Bush*
 - *www.whitehouse.gov is the official site*
- Session Hijacking
 - Intercepting a session initiated by another entity
 - *Redirect network communication to the attacker's host*
 - *Insert redirection on the victim's web-site*
- Man-in-the-Middle Attack
 - Alice wishes to talk to Bob
 - *Charlie pretends to be Alice to Bob and to be Bob to Alice*
 - *Charlie intercepts all messages between Alice and Bob and forwards, possibly modified versions, to the other entity*

Availability Threats Denial of Service (DoS)

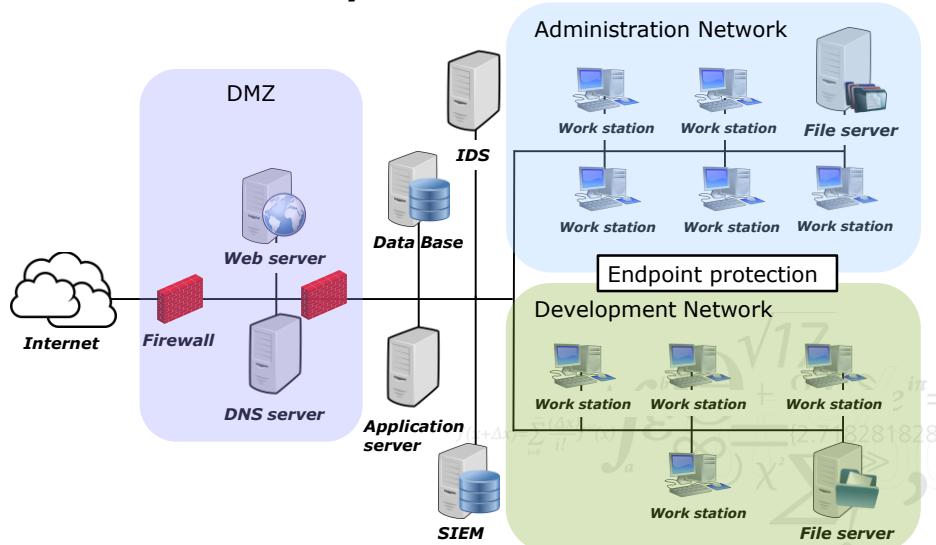
- Transmission failure
 - hardware/software failure of nodes or links
- Connection flooding
 - Exhaust bandwidth (sending too much traffic to the victim)
 - *This is what the Low Orbit Ion Cannon (LOIC) does*
 - Exhausting resources at the victim's site (SYN floods)
 - DNS attacks
 - *Redirecting traffic to the victim's site*
 - *Redirecting traffic away from the victim's site*
- Distributed Denial of Service (DDoS)
 - Break into many hosts (zombies) on the network (create a Botnet)
 - Instruct zombies to overload victim at a given time
 - Individual zombies may appear to be valid clients

Pause

17 DTU Compute Technical University of Denmark

02239 – Data Security

Network Security Architecture



18 DTU Compute Technical University of Denmark

02239 – Data Security

Communication Security

link encryption

- Encryption performed in “the Data Link Layer”
- Each link on the route is encrypted independently
- Advantages:
 - Transparent to hosts (OS and applications)
 - May protect packet meta data (headers)
 - Fast encryption hardware can be used
- Disadvantages:
 - Data decrypted/re-encrypted on all intermediate nodes
 - *Modification of standard protocols*
 - *Introduces a performance penalty*
 - *Data is exposed on all intermediate nodes (routers)*
 - Everything or nothing gets encrypted

Communication Security

end-to-end encryption

- Encryption performed in the “application layer”
- Encrypted at source and only decrypted at destination
- Advantages:
 - Data protected all the way from sender to receiver
 - Flexible choice of encryption algorithms
 - *Balance security and performance*
 - Some data can be sent unencrypted
 - No modification of the existing routing infrastructure
- Disadvantages:
 - User (programmer) needs to manage encryption explicitly
 - Applications and services may need to be modified
 - *Expensive and difficult to update encryption technology*

Virtual Private Networks (VPNs)

- VPNs use a public network (usually the Internet) to provide a secure connection between two private parts of a network or remote users on a network
- They can be used to connect a PC into a LAN (i.e., a telecommuter) or two connect two LANs (i.e., a branch office to a corporate headquarters)
- They are inexpensive to use
 - they eliminate the need for a costly private leased circuit
 - and they provide privacy over a public network

$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
$$\int_a^b \Theta^{\sqrt{17}} e^{\Omega f \delta e^{i\pi}} dx = [2.718281828]$$

Tunneling

- Tunneling is the technique used by the ends of a VPN to communicate
 - a packet send over a VPN is encapsulated in a secure protocol prior to being embedded into the IP protocol
- The receiving end (at the router) strips off the header and trailer information of the packet and the private network protocols decapsulates the packet and send the packet on
- Several private network protocols are available to encrypt the packet on a VPN



Tunneling

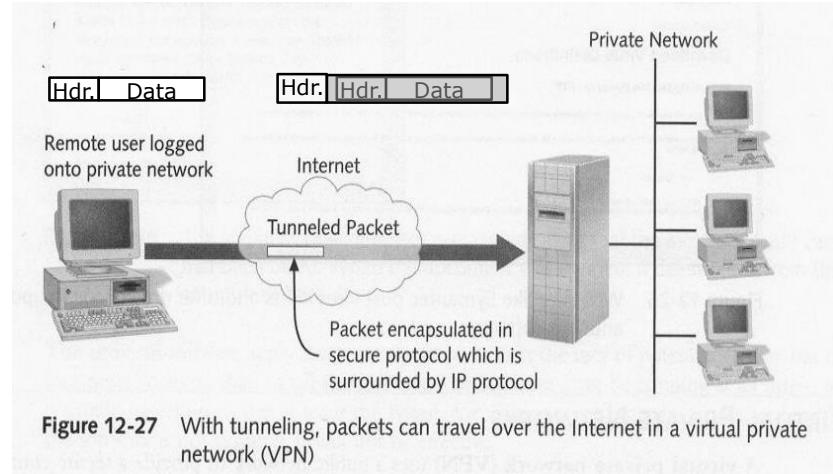


Figure 12-27 With tunneling, packets can travel over the Internet in a virtual private network (VPN)

Intrusion Detection definition

- *Intrusion detection is the process of identifying and responding to malicious activity targeted at computing and networking resources*
 - Process
 - *Interaction between people and tools, it takes time*
 - Identifying
 - *Before, during or after the intrusion*
 - Responding
 - *Collect evidence, limit damage (honey pots), shut-out*
 - Malicious activity
 - *Intentional attempts to do harm*
 - Computing and networking resources
 - *Logical intrusions as opposed to physical intrusions*

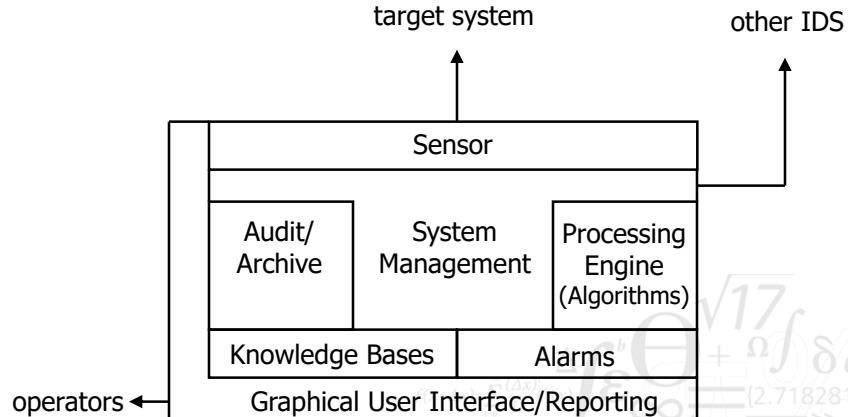
Two Types of IDS Systems

- Host Based Systems
 - Inspects locally available information
 - Application log-files
 - System log-files
 - Monitoring running activities
- Network Based Systems
 - Inspects traffic on the network
 - Signature Based Systems
 - Relies on established patterns ("signatures") in malicious network traffic
 - Similar to the signatures used in virus checkers
 - Anomaly Detection Systems
 - Establishes base line for normal communication
 - Detects abnormal traffic pattern
 - Employs techniques from machine learning, pattern matching, big data

IDS Infrastructure

- The IDS must be capable of collect and process enormous amounts of data
- Collect data from various sources
 - Network sniffers, login programs, logfiles, ...
- Capacity to examine all collected data
 - Look for intrusion patterns
 - Ignored data may contain evidence of an intrusion

IDS Components



Sensors

- Audit trail processing
 - Collect and check all logfiles (OS, servers, ...)
 - Most of the information is already collected
 - *Consolidate in Security Information and Event Management (SIEM)*
- On-the-fly processing
 - Look for “dirty words”, e.g., /etc/passwd
 - Complements audit trail processing
- Profiles of normal behaviour
 - Uses a priori information about system usage
 - Fine tuning of initial profile based on usage

Processing Engine

how to detect an intrusion?

- The processing engine is responsible for detecting intrusions
- Heuristics based intrusion detection
 - Repetition of a suspicious action
 - Mistyped command from an automated sequence
 - Known signatures (exploits)
 - Directional inconsistencies (inbound and outbound traffic)
 - Unexpected attributes of some service request or packet
 - Unexplained problems in a subsystem
- Anomaly detection
 - Statistics based
 - *Markov process, outlier detection, ...*
 - AI based
 - *Machine learning, Artificial Neural Networks, Deep Learning, ...*

Trapping Intruders

- The IDS may include a copy of the real system
 - Redirect intruders to this trap system
 - Must present a consistent view of the system in order to prevent detection
- The IDS may construct a honey pot
 - A trap system that looks attractive to intruder
 - *Based on his search for information*
 - *Example in Clifford Stoll: The Cuckoo's Egg*



Honeypots/Honeynets

- Machines that appear interesting to attackers
 - Offers no real data or services
 - Only “users” are attackers
 - Configured with sensors to detect and record usage
 - If used raise alarm
 - Provides clues to objectives and motives of attackers
- Purpose is to attract and retain attackers
 - Attract them away from production environment
 - Retain them while communication is analysed and tracked
 - Possibility of misinformation or misdirection
- Honeypots can be extended to networks
 - Check: <https://www.honeynet.org>



02239 – Data Security

31 DTU Compute Technical University of Denmark



Ethics of Intrusion Detection

- Intrusion detection requires monitoring of all network communication
 - Equivalent to Orwell’s Big Brother!
- Data collected for intrusion detection may be used (or abused) in another context
 - Establish activity of employees in a company



02239 – Data Security

32 DTU Compute Technical University of Denmark

Authentication



"On the Internet, nobody knows you're a dog."

DTU Compute

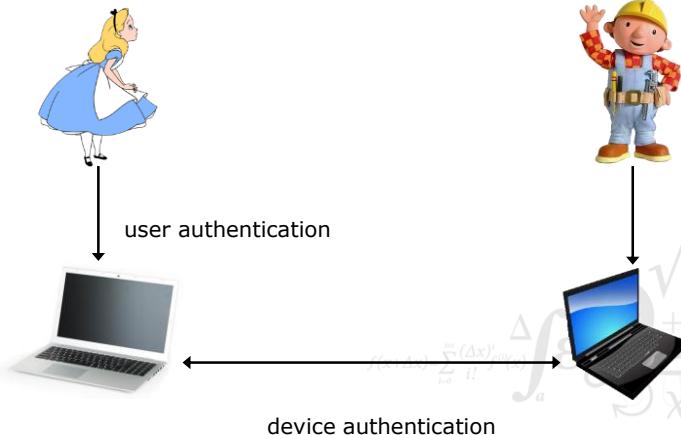
Department of Applied Mathematics and Computer Science

$$\int_a^b \varepsilon^{\sqrt{17}} \theta + \Omega \int \delta e^{i\pi} = \{2.7182818284 \dots\}$$

The purpose of authentication

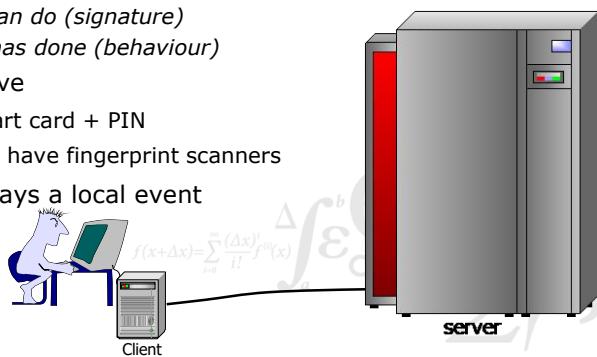
- Authentication is the process of verifying a claimed identity
- Allowing people to access the system
 - Ability to authenticate demonstrates authorization to access system
- Binding an identity to system entities
 - Authorizations are linked to identities (or roles that are assigned to ID)
- Associating a real world identity (transitively) with system events
 - Accountability facilitated through recording ID of requesting entities

Different Types of Authentication

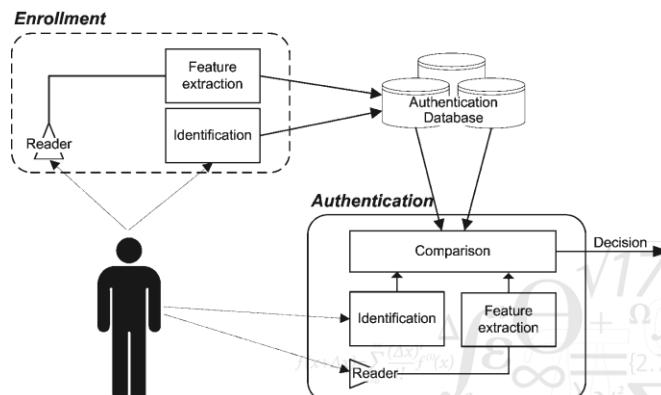


User Authentication

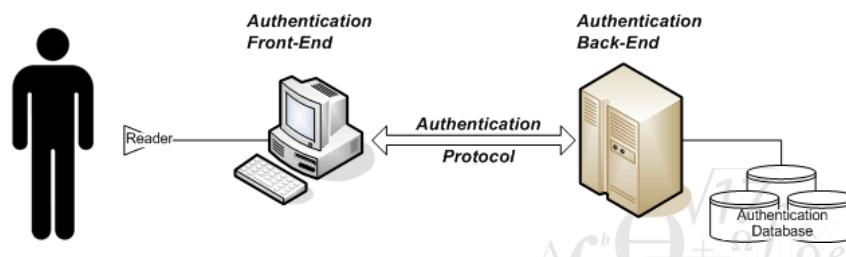
- Establish the identity of principals by means of:
 - something he knows (*shared secrets*, e.g., *password, PIN*)
 - something he possesses (*smart card, USB token, mobile phone*)
 - something he is (*fingerprint, face, voice, retina scan*)
 - *something he can do (signature)*
 - *Something he has done (behaviour)*
- Combinations of above
 - VISA cards has smart card + PIN
 - Many smart phones have fingerprint scanners
- Authentication is always a local event



Authentication Mechanism



General Model of Authentication



Where can this go wrong?

Password Authentication

- Most operating systems rely exclusively on passwords
 - login: username*
 - password: ******
- Password is checked by “login” program
 - Prompts for username
 - Prompts for password
 - Performs one-way function on password (hash)
 - Compares hashed password with password stored in password file
- Example – Classic Unix password file

```
Name : Password : UserID : GroupID : Gecos : HomeDirectory : Shell
- Example
bill:5fg63fhD3d5gh:157:5:Bill Smith:/home/bill:/bin/sh
- Password is used as key to encrypt a null string
  •  $E(pw+salt)+salt$  – password = 64 bit, salt = 12 bit
```

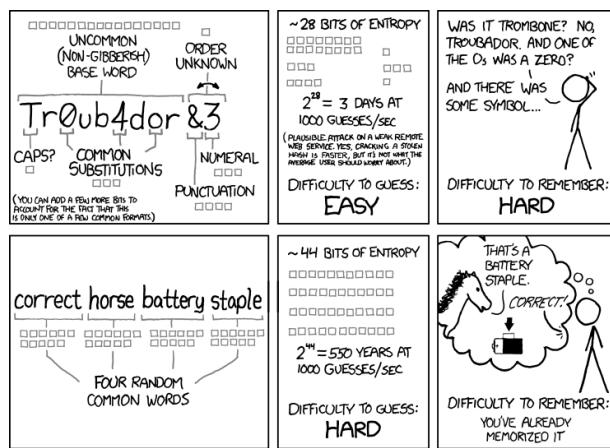
Passwords

- Single password is the most common method for authentication
 - Simple and “generally accepted”
 - *Everyone knows how they work*
 - Cheap to implement
 - *No additional hardware required*
- Password Security
 - Anyone who knows the password will be authenticated
 - Passwords must be difficult to guess
 - *Resistance to brute force attacks*
 - Passwords must be long (more than 12 characters)
 - Passwords must be complex (difficult to remember)
 - *Resistance to guessing-/dictionary attacks*
 - *Passwords should be unique (no reuse across websites)*
- Remembering many long complex passwords is hard for users

Passphrases

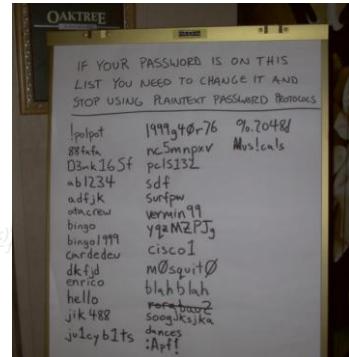
- Passphrases include several words
 - Technology is similar to password implementation
- Three major challenges
 - Usability vs. Security
 - Random words (similar to passwords but with a larger alphabet)
 - Natural language sentences (obey syntax and grammar)
 - Length improves security
 - Structure reduces entropy
 - Passphrase retention
 - Structure of passphrases makes them easier to remember
 - Common advice to construct and recall complex passwords
 - Passphrase Entry
 - Time consuming (typing more characters)
 - Error prone (getting all the characters exactly right)
- Improve usability by accepting passphrases with a few small errors

Password Strength



Attacks on Password Entry

- Passwords can be read over the shoulder (shoulder surfing)
 - Computers, ATM machine design, pay phones, ...
 - Try to hide what you are typing
- Passwords may be compromised every time they are used over a network (packet sniffers)
- Prevented by One Time Passwords (OTP)
- Password can be intercepted in transit between user and system
 - Trojan horse login screen
 - Trusted path between user and system is required
 - *ctrl-alt-del on Windows gives a genuine login screen*

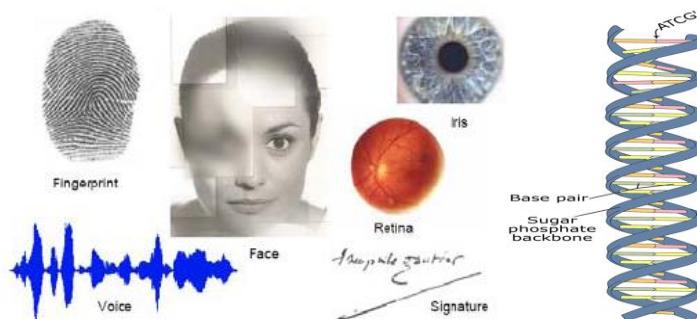


11 DTU Compute Technical University of Denmark

02239 – Data Security

Biometrics

Biometrics identify people by measuring some aspect of individual anatomy or physiology (hand geometry or fingerprint), some deeply ingrained skill, or other behavioural characteristic (handwritten signature), or something that is a combination of the two (voice)



12 DTU Compute Technical University of Denmark

02239 – Data Security

Biometric Authentication Systems

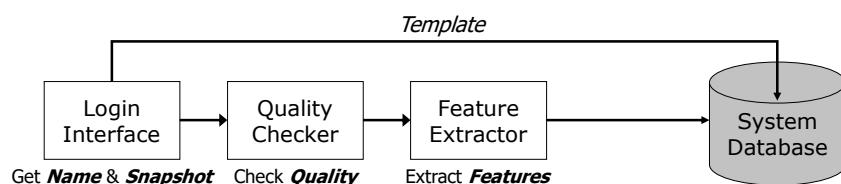
- Biometric systems have three types of operations
 - Enrollment (just like any other authentication system)
 - Verification (biometric authentication)
 - *match 1:1 one captured template to one stored template*
 - Identification
 - *match 1:N one captured template to N (or all) stored templates*



13 DTU Compute Technical University of Denmark

02239 – Data Security

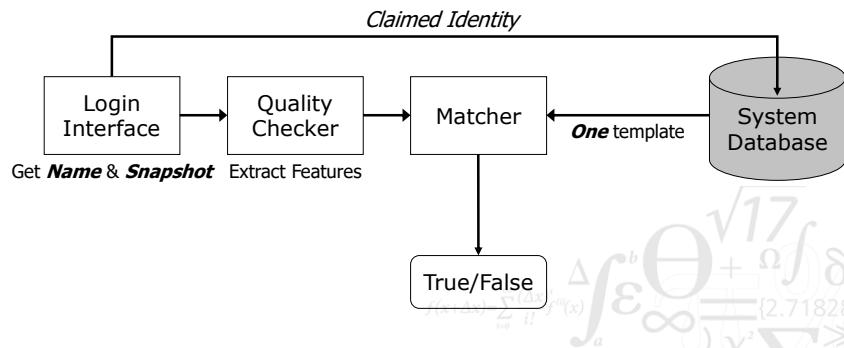
Enrollment in Biometric Systems



14 DTU Compute Technical University of Denmark

02239 – Data Security

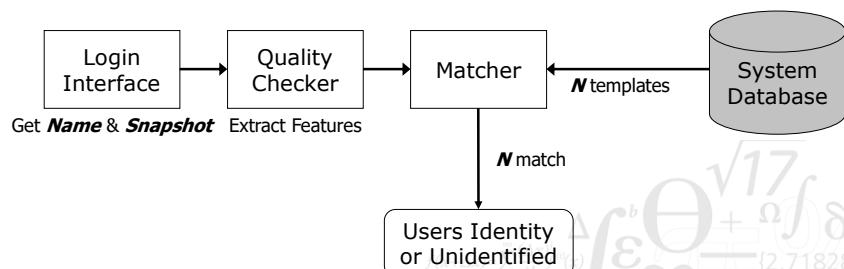
Verification in Biometric Systems



15 DTU Compute Technical University of Denmark

15 02239 – Data Security

Identification in Biometric Systems



16 DTU Compute Technical University of Denmark

16 02239 – Data Security

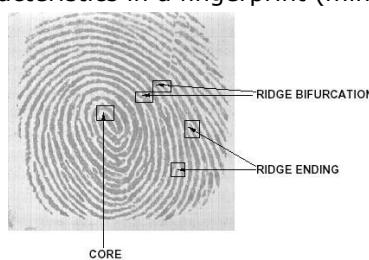
Handwritten Signatures

- Identifying handwriting is difficult, experts have an error rate of 6.5%, non experts have an error rate at 38%
- Problem with false accepts and rejects
 - false accepts result in fraud
 - false rejects result in insult (bad for business)
 - systems can be tuned to favour one over the other
- Optical systems are unreliable
- Signature tablets record shape, speed and dynamics of signature
 - more reliable

$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
$$\int_a^b \Theta + \Omega \int \delta e^{i\pi} = [2.718281828]$$
$$\Sigma !$$

Fingerprints

- Fingerprints have been used as signatures for several centuries
- They are currently used to identify criminals (affects acceptability)
- Measures unique characteristics in a fingerprint (minutiae)
 - Crossover
 - Core
 - Bifurcations
 - Ridge ending
 - Island
 - Delta
 - Pore
- Error rate can be affected by scars, wear, etc.
 - Very old and very young have weak fingerprints
- Many systems defeated by Gummy Fingers
 - Requires liveness detection



Cloning a Finger

[Matsumoto]

Making an Artificial Finger from a Residual Fingerprint

Materials

A photosensitive
coated Printed Circuit
Board (PCB)
"10K" by Sanhayato Co., Ltd.



Solid gelatin sheet
"GELATINE LEAF"
by MARUHA CORP



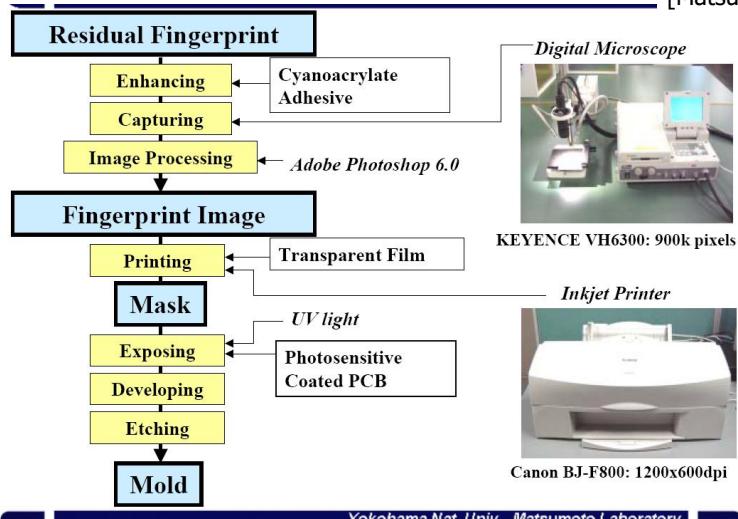
Yokohama Nat. Univ. Matsumoto Laboratory

19 DTU Compute Technical University of Denmark

02239 – Data Security

Cloning Process

[Matsumoto]

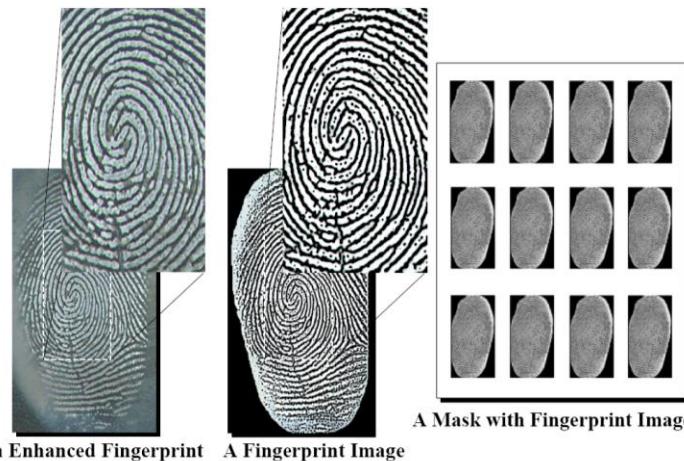


20 DTU Compute Technical University of Denmark

02239 – Data Security

Fingerprint Image

[Matsumoto]



An Enhanced Fingerprint A Fingerprint Image

A Mask with Fingerprint Images

Yokohama Nat. Univ. Matsumoto Laboratory

21

DTU Compute Technical University of Denmark

02239 – Data Security

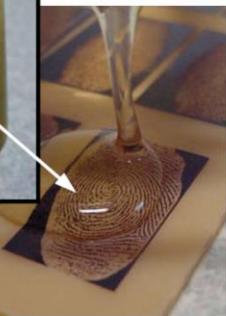
Molding

[Matsumoto]

Gelatin Liquid



Drip the liquid onto the mold.

Put this mold into
a refrigerator to cool,
and then peel carefully.

Yokohama Nat. Univ. Matsumoto Laboratory

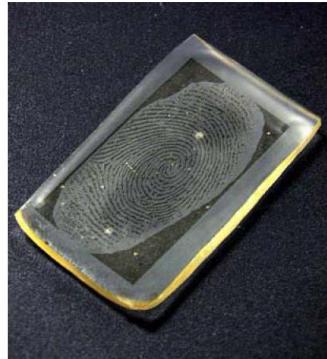
22

DTU Compute Technical University of Denmark

02239 – Data Security

The Mold and the Gummy Finger

[Matsumoto]



Mold: 70JPY/piece
(Ten molds can be obtained
in the PCB.)

Gummy Finger: 50JPY/piece

23

DTU Compute Technical University of Denmark

Yokohama Nat. Univ. Matsumoto Laboratory

02239 – Data Security

Side By Side

[Matsumoto]

Pores can be observed.



Enhanced Fingerprint



Captured Fingerprint Image of
the Gummy Finger
with the device H (a capacitive sensor)

24

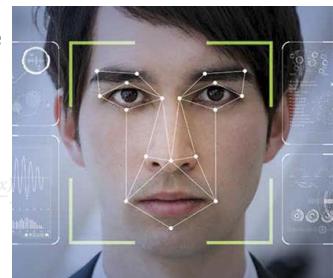
DTU Compute Technical University of Denmark

Yokohama Nat. Univ. Matsumoto Laboratory

02239 – Data Security

Face Recognition

- Face recognition is the oldest and most widespread form of identification
 - Manually used in photo ID (passport, ...)
 - Increasingly used in smartphones
- Uses off-the-shelf camera to measure the following facial features:
 - Distance between the eyes
 - Distance between the eyes and nose ridge
 - Angle of a cheek
 - Slope of the nose
 - Facial Temperatures (with IR camera)
- Multiple cameras allows 3D models
 - Stereo vision (2 normal cameras)
 - 1 normal camera + 1 IR camera

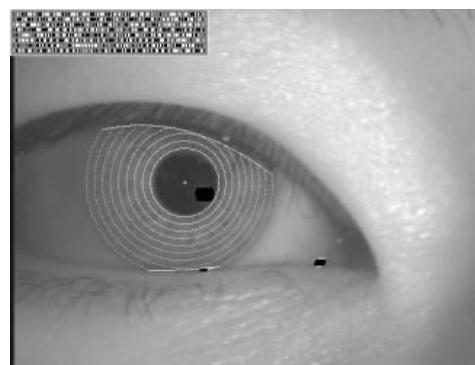


25 DTU Compute Technical University of Denmark

02239 – Data Security

Iris Scan

- Measures unique characteristics of the iris
 - Ridges (rings)
 - Furrows
 - Straitions (freckles)



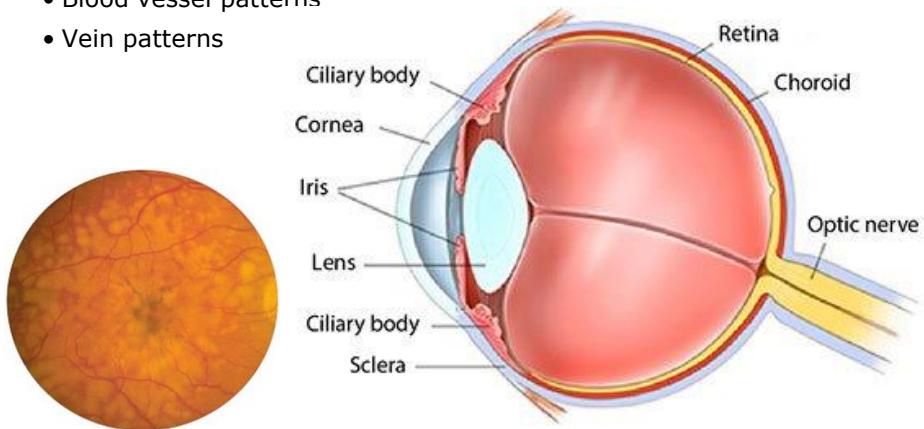
- Simple unattended systems can be defeated by photographs

26 DTU Compute Technical University of Denmark

02239 – Data Security

Retina Scan

- Measures unique characteristics of the retina (back of the eye ball)
- Blood vessel patterns
- Vein patterns



27 DTU Compute Technical University of Denmark

02239 – Data Security

Voice recognition

- Voice recognition identifies the speaker
 - Not to be confused with speech recognition (identifies what she says)
- Can be used for authentication over the phone
 - Include context to bind authentication to transaction:
"Transfer 200 Kr to account 123456789"
- Systems exist with < 1% error rate
- Tape recorders distort the voice enough to prevent "replay attacks"
- Digital recorders may be used to attack voice recognition systems in the future

$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$

$$\int_a^b \epsilon^{\Theta} + \Omega \int_a^b \delta e^{i\pi} = 2.718281828$$

$$\infty \approx x^2 \sum_{n=0}^{\infty}$$

28 DTU Compute Technical University of Denmark

02239 – Data Security

Biometric Authentication Summary

- Automated identification systems (scanners) have been developed
- Quality of system depends on accuracy (error rate)
- Error rate is often below 1%
 - What does error rate < 1% mean in practice?
 - Heathrow Airport has ~111.000 passengers arriving every day (2018)
 - Around 4625 passengers arrive every hour
 - Around 77 passengers arrive every minute
 - Around 1 error every second for passengers arriving at Heathrow
 - Reducing error rate by an order of magnitude makes little difference



29 DTU Compute Technical University of Denmark

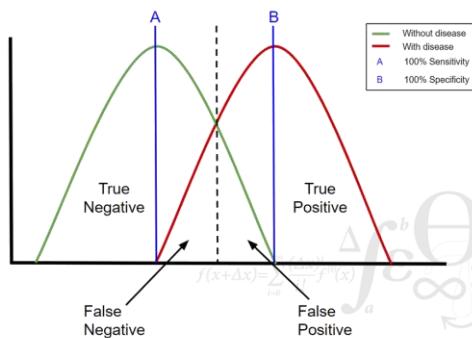


02239 – Data Security

Threshold Based Authentication Systems

- Comparison of presented features with stored template
 - Rarely an exact match, so verification is based on threshold

Sensitivity vs. Specificity



30 DTU Compute Technical University of Denmark

02239 – Data Security

Authentication Mechanism Quality Metrics

- Threshold based mechanisms give four possible results

	Is the person claimed	Is not the person claimed
Test is positive (there is a match)	True Positive	False Positive
Test is negative (there is no match)	False Negative	True Negative

$$\text{Sensitivity} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Specificity} = \frac{\text{True Negative}}{\text{False Positive} + \text{True Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{False Negative} + \text{True Negative}}$$

$$\text{Prevalence} = \frac{\text{True Positive} + \text{False Negative}}{\text{True Positive} + \text{False Positive} + \text{False Negative} + \text{True Negative}}$$

Break



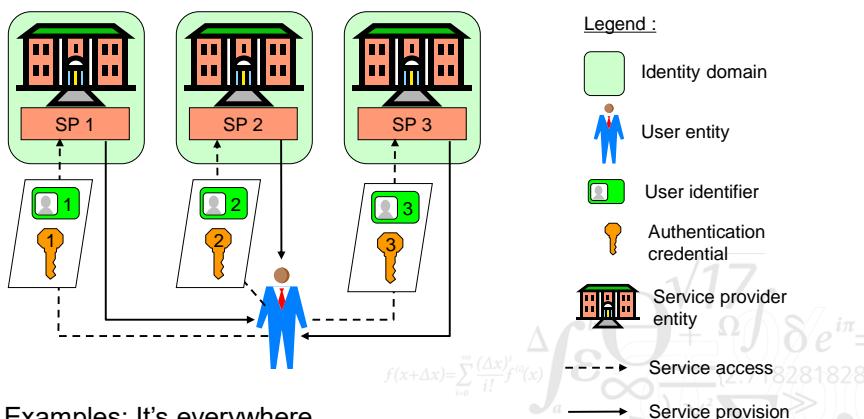
Identity Management Models

- Authentication in distributed systems require an agreed model f identities and authentication
- Three fundamental models:
 - Identity Silos
 - Single Sign-On systems
 - Federated Identity models

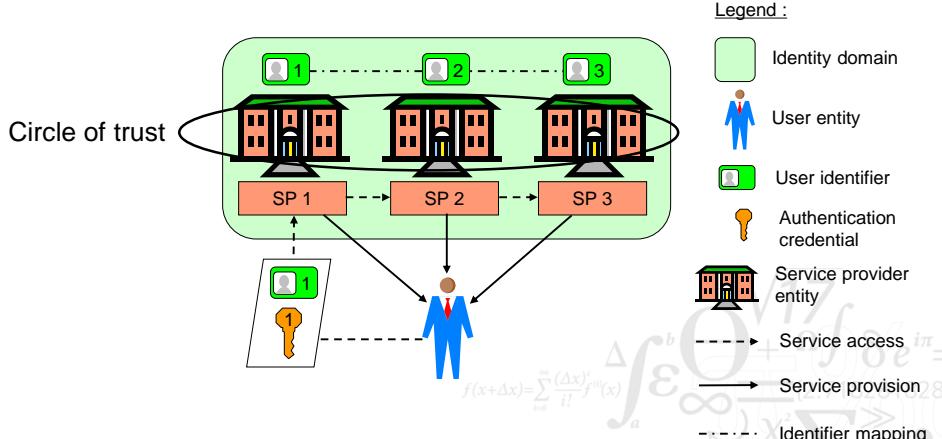


$$f(x+\Delta x) = \sum_{l=0}^{\infty} \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$
$$\int_a^b \Theta + \Omega \int \delta e^{i\pi} = [2.718281828]$$
$$\infty \sum \gg !$$

Isolated User Identity Model

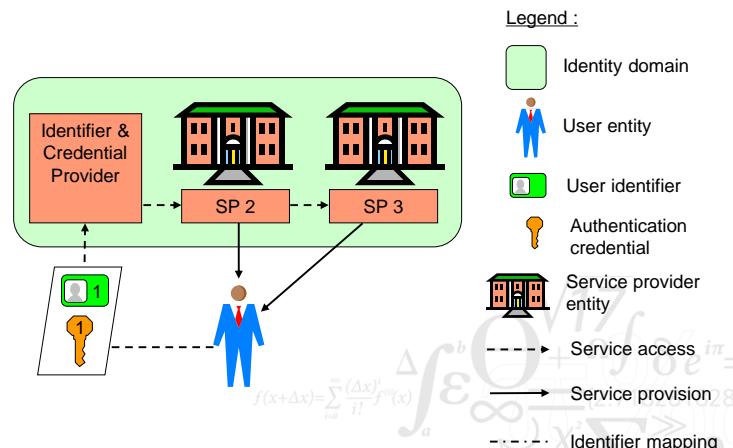


Federated Identity Model



Examples: SAML2.0, WS-Federation, Shibboleth, Eduroam

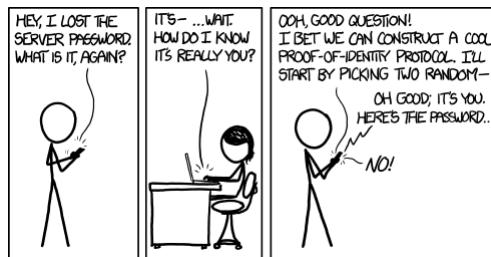
Single Sign On (SSO) Systems



Examples: Kerberos, Nem-ID

Authentication Protocols

- Authentication protocols extend authentication across networks
 - Authentication of remote users
 - Authentication of remote devices
 - Authentication of intention to authenticate
 - Protection against *relay attacks*
 - Protection against *replay attacks*



38 DTU Compute Technical University of Denmark

02239 – Data Security

Kerberos

- Project Athena at MIT (mid to late 1980s)
- Hundreds of diskless workstations
 - Open terminal access, no physical security
 - Insecure network
- Few servers (programs, files, print, ...)
 - Physically secure



39 DTU Compute Technical University of Denmark

02239 – Data Security

Simple Authentication

- One password per service is infeasible
 - Identity silos
- New authentication service (Charon) introduced
 - Single sign-on service
- Both users and services have passwords
- Charon identifies user by password
- Charon returns a "ticket" to the user
 - Ticket includes identity encrypted with the service's password
 - If the ticket decrypts properly, access to the service is granted
- How do we know that we got the right ticket?

Stronger Authentication

- Include service name in the ticket to prove that it was properly encrypted
- Include client workstation address to prevent network sniffers
- Remaining problems:
 - Re-authentication every time a new service is contacted
 - Password sent across the network in the clear

Ticket Granting Service

- TGS has access to the Charon database
- TGS provides service tickets to users with a TGS ticket (eliminate resubmitting password)
- Users obtain ticket granting tickets from Charon
- User sends username, receives TGS ticket encrypted with user's password
- Tickets can be reused

Insecure Workstations

- What happens to tickets after a user has logged out?
 - An opponent could log on to the workstation and use the tickets
 - Could be explicitly destroyed when user logs out
 - Sniffer could be used to capture tickets, hacker may then login to the same workstation and use the tickets (replay session)
- This demonstrates common problems in authentication
 - In-memory caches of data and re-use of physical resources
 - Problem with secure revocation of authorisations
 - Ensuring freshness of authentication data

Limiting Ticket Lifespan

- Charon timestamps ticket when it is issued
- Charon includes lifespan along with timestamp
- Remaining problems:
 - Workstation clocks must be synchronized
 - What should the lifespan of a ticket be?

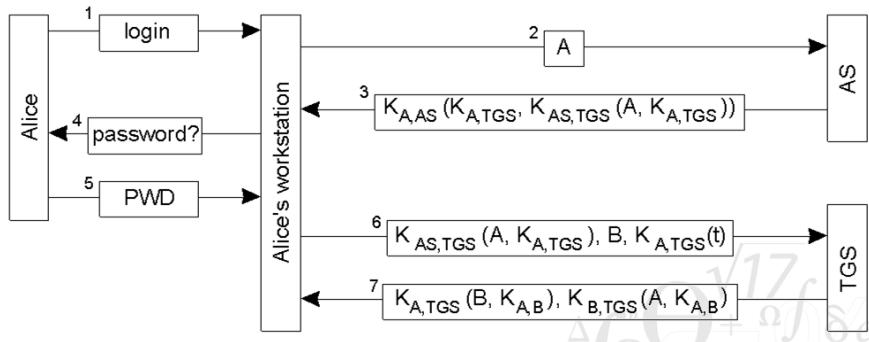
Verifying Tickets

- Authentication relies on the following tests:
 - Can the service decrypt the ticket?
 - Has the ticket expired?
 - Do the username and workstation address correspond?
- The tests prove:
 - The ticket came from Charon
 - The ticket is still valid
 - Failure proves that the ticket is false, success does not prove anything, tickets can be stolen and reused on the same workstation

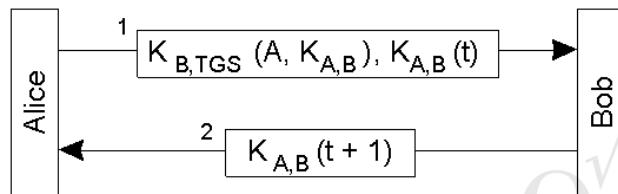
Session Key and Authenticator

- Session key returned along with ticket
 - Charon reply = [sessionkey, ticket]
 - Ticket = [sessionkey:username:address:servicename:lifespan:timestamp]
- Authenticator used to contact service
 - Authenticator = {username:address} encrypted with sessionkey
- Client sends authenticator and ticket to service
- The service now knows:
 - The ticket's lifespan and timestamp
 - The ticket-owner's name and address
 - The sessionkey
- The session key authenticates the authenticator and vice versa
- Both can be stolen at the same time and reused
- Include timestamp and *short* lifespan in authenticators
- The session key is also used to authenticate the service

Authentication in Kerberos

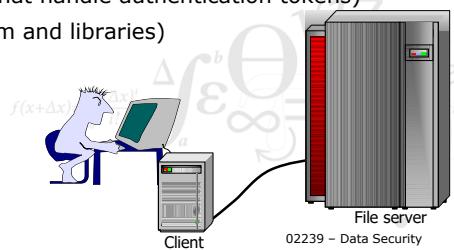


Setting up a secure channel in Kerberos



User Authentication in Distributed Systems

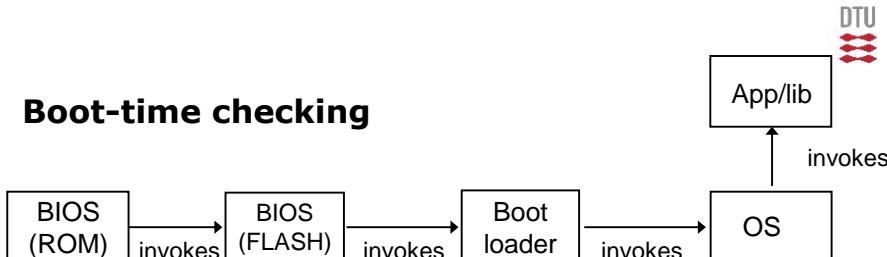
- How can we prove that a remote user is who he claims to be?
- User authentication takes place remotely at the client
 - Needs federated identity management
- Communication channel must be secure (CIA)
 - Integrity is required
 - Confidentiality and Availability as required
- Device authentication needed to determine if client is trusted
 - Client applications (programs that handle authentication tokens)
 - Client system (operating system and libraries)
 - Client hardware



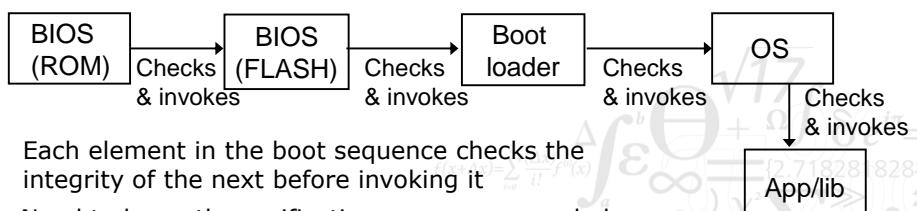
Trusted Computing Group

- Trusted Computing Group (TCG) defines the TPM standard
 - TPM defines a special processor
 - *Tamper resistant hardware (environment for secure storage and processing)*
 - *Support for cryptographic operations*
- TPM provides the following functionalities
 - Protected capabilities
 - *Commands with exclusive access to shielded locations*
 - Shielded locations
 - *Domain where it is safe to access sensitive (shielded) data*
- TCG does not control the implementation
 - Vendors are free to differentiate the TPM implementation
 - TPM must still meet the protected capabilities and shielded locations requirements

Boot-time checking



A well-defined sequence of software modules get executed at boot time.

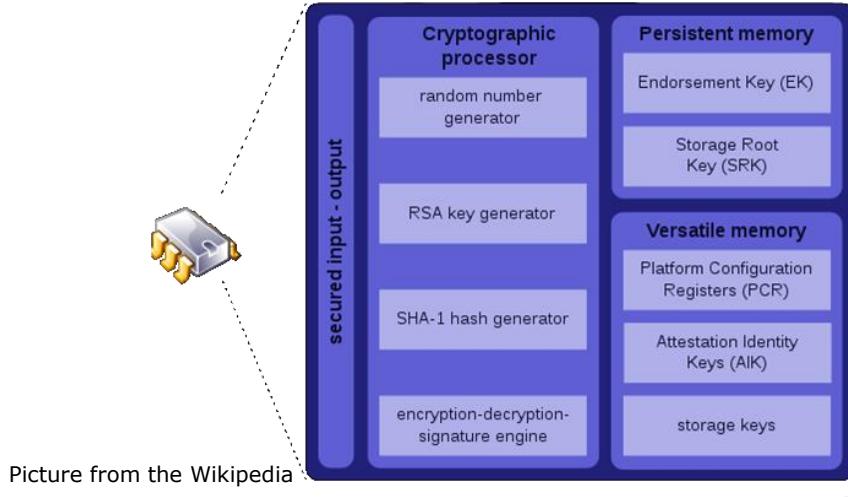


Each element in the boot sequence checks the integrity of the next before invoking it

Need to know the verification process succeeded

Trusted boot or secure boot

TPM Architecture



52 DTU Compute Technical University of Denmark

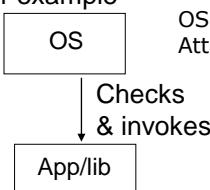
02239 – Data Security

Platform Configuration Registers (PCRs)

- PCRs are used to securely measure software (by computing hash) during boot
- Each PCR can contain a SHA-1 hash value (20bytes)
 - At least 16 PCRs
- PCRs are reset to 0 at boot time
- Write to a PCR # n by extending it – **hash extension**

$$\text{TPM_Extend}(n,D): \quad \text{PCR}[n] \leftarrow \text{SHA-1}(\text{PCR}[n] \parallel D)$$

For example



OS computes $h_3 = \text{SHA-1}(\text{module3})$; stores $\text{SHA-1}(0, h_3) \rightarrow \text{PCR}[3]$
 Attacker substitutes module3 with module3', $h_3' = \text{SHA-1}(\text{module3}')$

53 DTU Compute Technical University of Denmark

02239 – Data Security

Trusted/Secure Boot Sequence

- At power-up PCR[n] initialized to 0
- BIOS boot block executes
 - Calls `PCR_Extend(n, <BIOS code>)`
 - Then loads and runs BIOS post boot code
- BIOS executes:
 - Calls `PCR_Extend(n, <MBR code>)`
 - Then runs MBR (master boot record)
- MBR executes:
 - Calls `PCR_Extend(n, <OS loader code, config params>)`
 - Then runs OS loader

Which PCRs to use is defined by specifications

$$f(x+\Delta x) = \sum_{l=0}^n \frac{(\Delta x)^l}{l!} f^{(l)}(x)$$

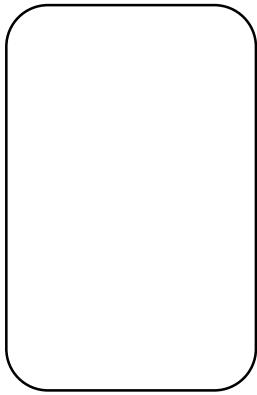
Thinking about Authentication





OAUTH 2.0

- Nate Barbetinitis slides??



$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\int_a^b \Theta_{\infty}^{\sqrt{17}} \delta e^{i\pi} = \frac{\Omega}{2.718281828} \sum_{n=1}^{\infty} \frac{(-1)^n}{n^n} x^n$$

Protection Mechanisms



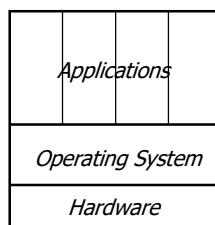
$$\Delta \int_a^b \varepsilon^{\sqrt{17}} \Theta + \Omega \int \delta e^{i\pi} = \{2.71828182845904523536028747135266249 \dots\}$$

DTU Compute

Department of Applied Mathematics and Computer Science

Protection in Operating Systems

- OS implements the fundamental security mechanisms



- What needs to be protected
 - Memory
 - Sharable I/O devices (disks, network interfaces, ...)
 - Serially reusable I/O devices (printers, tape drives, ...)
 - Shared programs and sub-procedures (services)
 - Shared data (files, databases, ...)

Separation of Subjects' Access to Objects

- Separation forms basis for most protection mechanisms
- Processes may have different security requirements
- Physical separation
 - Different processes use different physical objects (separate hardware)
- Temporal separation
 - Different processes are executed at different times
- Logical separation
 - OS creates illusion of physical separation
- Cryptographic separation
 - Processes conceal data and computations in a way that makes them unintelligible to outside processes
 - Encrypt data
 - Some algorithms for computation on encrypted data exist
 - Homomorphic encryption

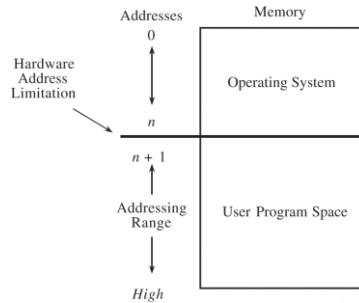
Principles of Protection

- Do not protect
 - Appropriate when physical/temporal separation is used
- Isolate
 - Processes are completely unaware of other processes (virtual machines)
- Share all or share nothing
 - Public or private data
- Selective sharing (*share via access limitations/share by capabilities*)
 - OS enforces a policy that defines how objects can be shared by users
 - Mandatory-/Discretionary policies
 - Generally implemented in a reference monitor
- Usage control (*limit use of an object*)
 - Restricts use of objects after access has been granted
 - Typical goal for DRM systems
 - Applications require support from hardware and OS

Fence

Memory and Address Space Protection I

- Separation between OS and user programs

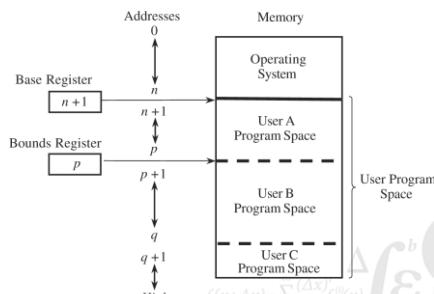


- Predefined memory address
 - Operating system resides below this address
 - Programs are loaded from this address and cannot access OS memory
 - Special *Fence Register* allows re-allocation of memory

Base/Bounds Registers

Memory and Address Space Protection II

- Fence only protects in one direction (underflow)
- Base/Bounds registers protect in both directions
 - Base register corresponds to fence

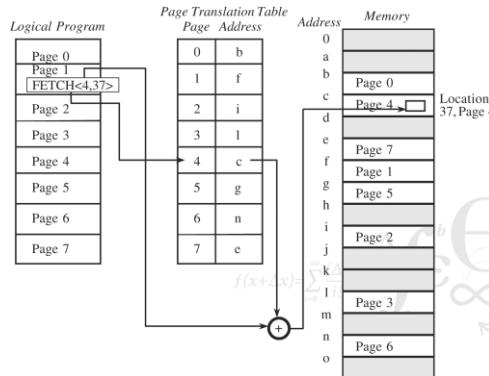


- Each process has its own pair of base/bounds registers
 - Protects processes from each other
 - One man's bounds is another man's base*

Paging

Memory and Address Space Protection III

- Variable size segments are difficult/expensive to manage
- Paging introduces fixed sized segments (page frames)
- Typically powers of 2 between 512 and 4096 bytes

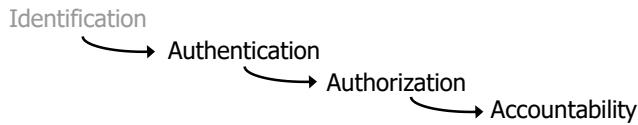


Paging II

Memory and Address Space Protection V

- Page translation tables define the addressable memory of a process
 - Managed by the OS
 - Prevent user processes from "mapping" OS memory into its address space
- There is no logical structure to memory pages
 - Data with different security requirements may reside on the same page
- Security benefits of paging include:
 - Address references can be checked for protection
 - When the relevant page is "mapped" (inserted in page translation table)
 - Users can share data by sharing physical memory pages
 - Access rights do not have to be the same for all users
 - Users cannot access main memory directly
- Most current systems implement a paging architecture

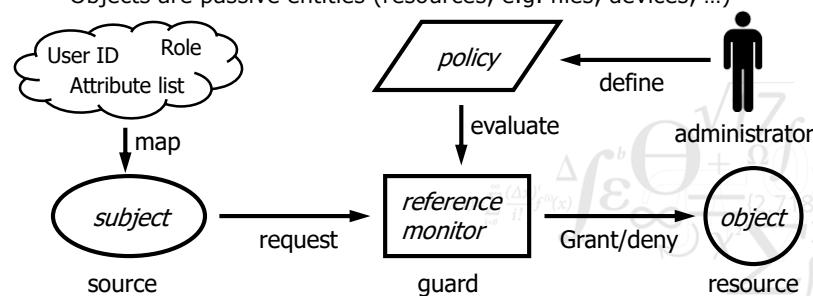
Classic view of security



- Authentication
 - Verifies the claimed identity of subjects
- Authorization
 - Enforces access control policy
 - Decides whether a subject has the right to perform an operation on an object
- Accountability
 - Records security relevant events
 - What happened? and who did what?

Access Control Model

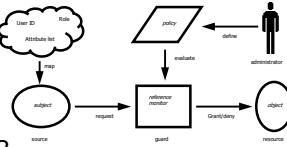
- Security policy is evaluated every time an object is accessed
 - Reference Monitor mediates all access by subjects to objects
 - Guards access to object
 - Interprets access control policy
 - Subjects are active entities (users, processes)
 - Objects are passive entities (resources, e.g. files, devices, ...)



Reference Monitors in Distributed Systems

- Concept developed for centralised Operating Systems

- Policy enforced by components in the OS
- Policy defined by local system administrators
- Policy based on local information



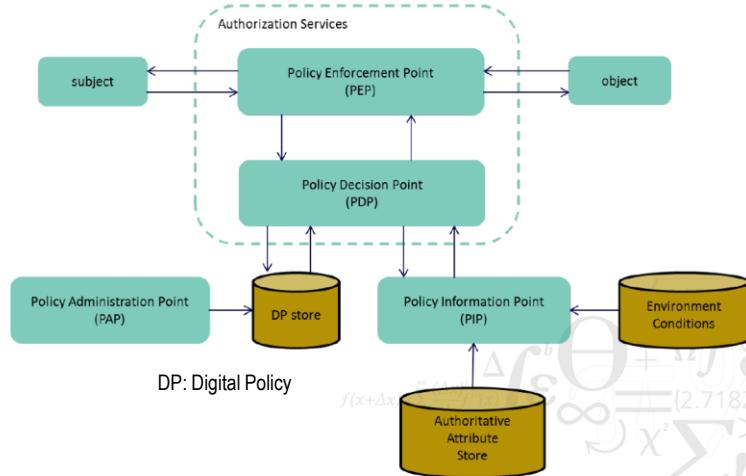
- How does this extend to distributed systems?

- Resources hosted on different machines
 - Possibly managed by different local administrators
 - Possibly belonging to different administrative domains
- Access Control decisions may be federation of local policies
 - Federated identity management
 - Federated access control policies
- Distributed enforcement of policies

Access Control Architectural Elements

- PEP: Policy Enforcement Point:
 - Grants or denies access
- PDP: Policy Decision Point:
 - Decides whether access should be granted or denied
 - Uses the policies recorded in the Policy Store
- PAP: Policy Administration Point
 - Manages the Policy Store: adds, removes and modifies policies
- PIP: Policy Information Point
 - Provides the information that the PDP needs to make decisions
 - Model parameters, roles, attributes, hierarchies, constraints
 - State of the environment:
 - Examples: Time of Day, Normal Working Hours, ...
 - Location of users and/or resources
 - Etc.

Access Control Architecture



13 DTU Compute Technical University of Denmark

02239 – Data Security

Mapping Subjects

- Identity Based Access Control
 - Permissions are granted directly to users
 - Unique system identifier (UID) for every user
 - User identity must be verified before use (authentication)
- Role Based Access Control
 - Permissions are granted to roles
 - Users assigned one or more roles
 - User identity must be verified before role is assumed (authentication)
- Attribute Based Access Control
 - Permissions depend on user's attributes
 - Users must prove possession of attributes
 - *Attributes are often encoded in certificates*
 - Use of certificates often require user's public-key
 - *Use of public-key certificate implies authentication*
- Ultimately, users must prove identity to exercise access rights

14 DTU Compute Technical University of Denmark

02239 – Data Security

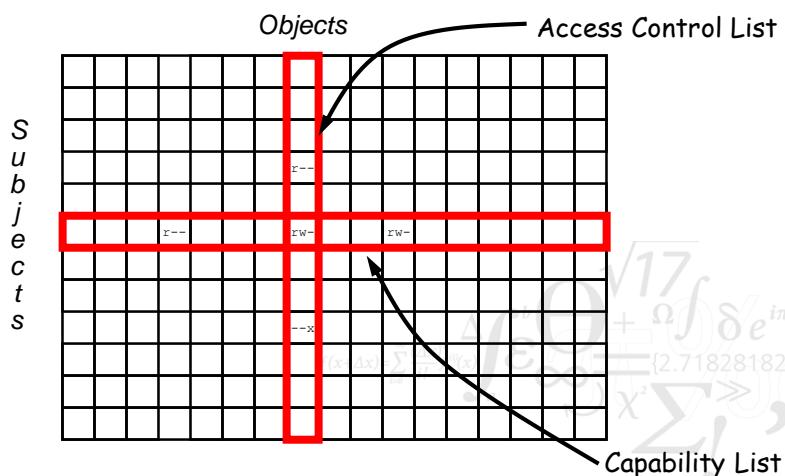
Access Control Matrix Model

- Access Control Matrix defined by
 - Set of subjects S (active entities in the system)
 - Set of objects O (passive entities in the system)
 - Set of rights R (defines operations that subjects can do on objects)
- A denotes the entire access control matrix
 - Encodes the access rights of subjects to objects
 - A is often a sparse matrix
- $a[s,o]$ denotes the element at row s , column o ; $a[s,o] \in R$

subjects	objects			
	file 1	file 2	process 1	process 2
process 1	read, write, own	read	read, write, execute, own	write
process 2	append	read, own	read	read, write, execute, own

Representing the Access Control Matrix

- The Access Control Matrix is often sparse



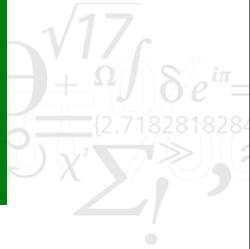
Access Control Lists

- Associated with every object in the system
 - List of pairs: $\langle \text{subject name}, \text{access rights} \rangle$
- Access is granted if
 - Subject name is in the list
 - Access rights include requested operation
 - Otherwise access is denied
- Some ACL systems allow special default actions (**grant** or **deny**)
 - Useful with negative access rights
 - *ACL becomes a list of people to exclude*
- Delegation is difficult
 - Requires the right to modify the ACL
- Questions about access rights
 - Easy to know who may access an object
 - Difficult to know what objects a subject may access

Capabilities

- List of capabilities is associated with every subject in the system
 - List of pairs: $\langle \text{unique object identifier}, \text{access rights} \rangle$
- Capabilities are used to reference the object
 - Without a capability, object cannot be addressed
 - Access is granted if rights in the capability includes requested operation
- Three types of capabilities
 - Hardware capabilities
 - Segregated capabilities
 - Encrypted capabilities
- Capabilities are easy to delegate
- Questions about access rights
 - Difficult to know who may access an object (who has a capability)
 - Easy to know what objects a subject may access (and how)

Break



19 DTU Compute Technical University of Denmark

02239 – Data Security

Security Policies

- Prevent disclosure or corruption of sensitive data
 - Controlled access to protected resources
 - Isolation (confinement)
 - Separation of functions (place order and sign check)
 - Well formed transactions
- Mandatory Access Control
 - System defines policies (users have little direct influence)
 - System "owns" resources
- Discretionary Access Control
 - Users define policies (system has little direct influence)
 - User "owns" resources

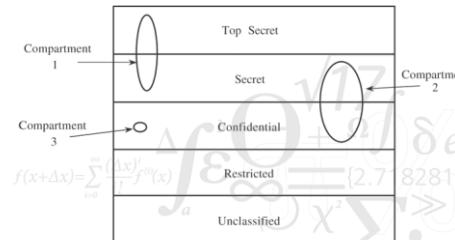
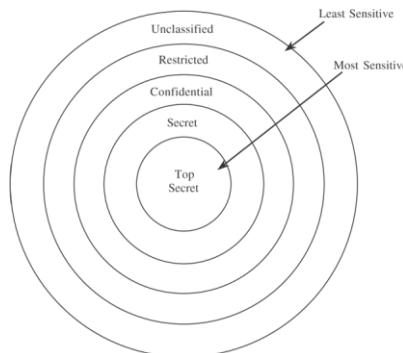
$$f(x+\Delta x) = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x)$$

20 DTU Compute Technical University of Denmark

02239 – Data Security

Military Access Control Policies

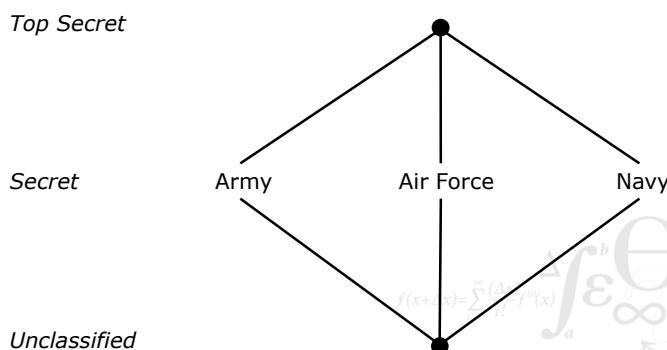
- Keeping military plans secret
 - Confidentiality is primary concern
 - *Need-to-know principle*
 - Traditional model based on safes and marked binders



21 DTU Compute Technical University of Denmark

02239 – Data Security

Access Control Lattice



22 DTU Compute Technical University of Denmark

02239 – Data Security

Bell & LaPadula

Multilevel Security

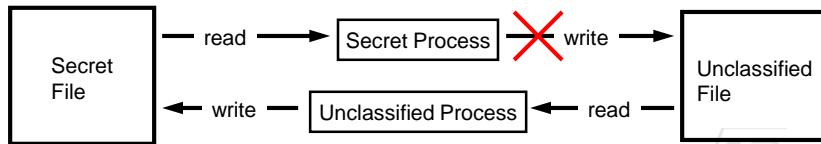
- Mandatory access control model
 - Separate users with multiple levels of privileges on the same system
 - Military system
 - *Security labels: unclassified ≤ restricted ≤ confidential ≤ secret ≤ top secret*
- Basic definitions:
 - **object:** passive entity, stores information
 - **subject:** active entity, manipulates information
 - **label:** identifies the *secrecy classification* of the object
 - **clearance:** specifies the most secret class of information available to the subject
 - **permission:** specifies the operations that the subject is allowed to invoke on the object, the model defines: *read, write, append, and execute* permissions

Bell & LaPadula II

- **Domination:** (relation)
 - Label (or clearance) A is said to *dominate* a label B , if a flow of information from B to A is authorized
 - A dominates B is written $A \geq B$
- **Security Rules:**
 - Simple security condition
 - *Subject s may only access an object o, if the clearance of s dominates the label of o*
 - The *-property
 - *Subject s may only use the content of an object o₁ to modify an object o₂, if the label of o₂ dominates the label of o₁*

*NB! A consequence of the *-property is that objects tend to rise slowly towards the highest classification*

Bell & LaPadula III



Bell & LaPadula IV

- Implementation issues:
 - Unavailability of passive objects
 - Objects must be activated before they are accessed
 - Tranquillity principle
 - The label of an active object cannot be changed
 - Initialization of objects
 - The initial state of an object does not depend on any previously allocated resource
- The system call open() is an example of activation

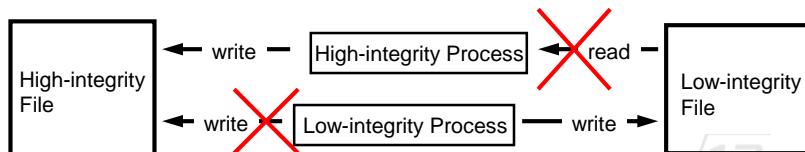
Biba Integrity Model

- In civilian systems, integrity is more important than secrecy
- Biba defines an integrity model similar to the Bell & LaPadula model
 - Introduces integrity classes
 - Prevents information from objects with low integrity to contaminate objects with a higher integrity

Integrity Rules:

1. Simple integrity: Subject s can only modify an object o if the integrity class of s dominates the integrity class of o
2. Confined integrity: Subject s can only read the content of an object o if the integrity class of o dominates the integrity class of s

Biba Integrity Model II



Role Based Access Control (RBAC)

- In many cases, authorization should be based on the function (role) of the subject in the manipulation of the object
 - Consider the following example:
 - Anne, accountant for DTU Compute, has access to financial records
 - She leaves
 - Eva is hired as the new accountant, so she now has access to those records
 - How are all the necessary permissions transferred from Anne to Eva?
- Examples of Roles:
 - Function in a bank
 - Teller clerk, Financial advisor, Branch manager, Regional manager, Bank director
 - Function in a hospital
 - Doctors (GP, consultant, treating doctor, ...), Nurses (ward nurse, nurse, ...), Hospital administrators
 - Functions at a university
 - Academics (teachers, research fellows, ...), Non-academic staff (secretaries, system administrators, ...), Students

Common RBAC Concepts

Definitions:

- **Active role:**

$AR(s : subject) = \{\text{the active role for subject } s\}$

- **Authorized roles:**

$RA(s : subject) = \{\text{authorized roles for subject } s\}$

- **Authorized transactions:**

$TA(r : role) = \{\text{authorized transactions for role } r\}$

- **Predicate exec:**

$\text{exec}(s : subject, t : transaction) = \text{true iff } s \text{ can execute } t$

- **Session:**

Binds a user to a set of currently activated roles

General RBAC Rules

Rules:

1. Role assignment:

$\forall s : subject, t : transaction \ (exec(s,t) \Rightarrow AR(s) \neq \emptyset)$

A subject can only execute a transaction if it has selected a role

2. Role authorization:

$\forall s : subject \ (AR(s) \subseteq RA(s))$

A subject's active role must be authorized for the subject

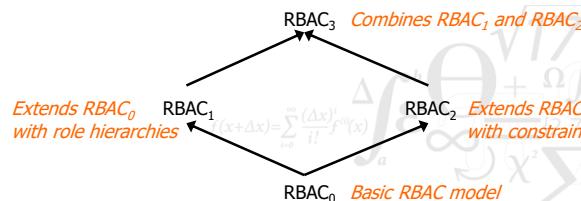
3. Transaction authorization:

$\forall s : subject, t : transaction \ (exec(s,t) \Rightarrow t \in TA(AR(s)))$

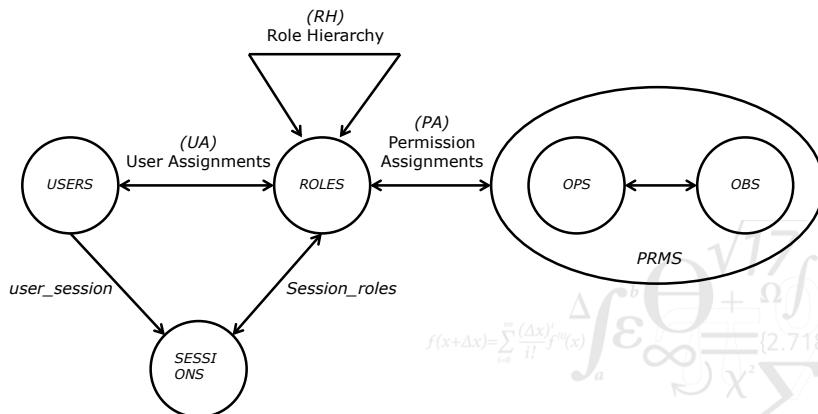
A subject can only execute a transaction if it is authorized for its active role

RBAC96

- Role-Based Access Control was initially defined by Ferraiolo & Kuhn from NIST in 1992
- A family of related RBAC models were defined by Sandhu et al. in 1996 – this family is commonly known as RBAC96
 - RBAC96 forms the basis for most of the continued work on Role-based Access Control
- RBAC96 defines the following models:



RBAC



Attribute Based Access Control (ABAC)

- KeyNote [RFC 2704] builds on “assertions” (credentials)
 - Blaze, Feigenbaum, Ioannidis, Keromytis; 1999
- An assertion consists of two parts
 - Identification of an agent (could be the public-key)
 - Specification of an allowed operation on a resource
 - An assertion is digitally signed by the issuer
- Assertions may be provided by:
 - The system (from the security policy)
 - The agent itself (“credential”)
- An operation is allowed if there exists an assertion that permits the operation
 - Explicit permission from the issuer
 - Implicit through other assertions from the same issuer
 - *This requires an inference engine to derive new assertions.*

Monotony in ABAC

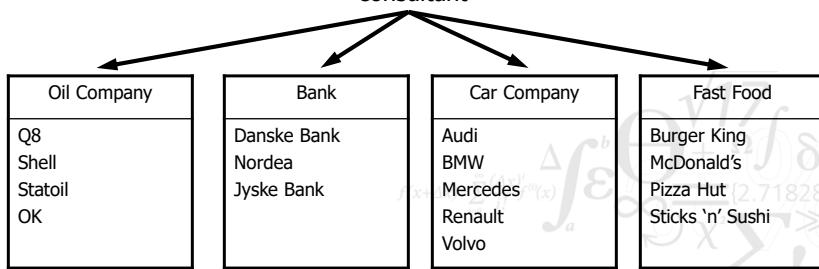
- Assertions are Monotonous
 - Addition of an assertion never disallows an operation
 - Deletion of an assertion never allows a prohibited operation
 - Everything is prohibited unless explicitly allowed
- Significance of monotony
 - Safe to use in distributed systems
 - *Lost assertions cannot break a policy*
 - Set of assertions that combines to allow an operation constitutes a proof that the security policy is enforced
 - Clients may collect signed assertions and send them to the server
 - *Offloads work from server to clients*
 - No conflicts are possible
 - *If an operation can be allowed based on the system's assertions, the operation will be allowed*

ABAC in Practice

- Suitable for large distributed systems
 - Decentralized specification of security policies
 - Decentralized (autonomous) enforcement of security policies
- Simultaneously gives permission and the justification for allowing an operation
 - Set of assertions used to authorize the operation
- Allows dynamic evolution of security policies
 - Addition of new assertions may add new users, roles permissions or resources
- Not obvious how context may be encoded in assertions
 - This is one potential obstacle to its application in pervasive computing environments

Chinese Wall Model

- Developed to avoid conflict of interest in consultants
- The consultancy firm divides clients into business areas
- Each consultant may work for several clients
 - a priori, no limitations are assumed
 - only *one* client in each business area is allowed



A consultant may work for any one company in each class

02239 Data Security Privacy

Sebastian Mödersheim



November 3, 2021

Outline

- ① Why Privacy?
- ② Defining Privacy
- ③ Privacy in Corona Apps
- ④ Non-Technical Approaches
- ⑤ Mixes
- ⑥ Zero-Knowledge Protocols

Outline

① Why Privacy?

② Defining Privacy

③ Privacy in Corona Apps

④ Non-Technical Approaches

⑤ Mixes

⑥ Zero-Knowledge Protocols

Extreme Views

- “An honest person has nothing to hide!”



- “1984”



An Example [from Mark Ryan]

- Parents and their teenage daughter:
 - ★ The daughter wants to go out, but not tell the parents where she goes.
 - ★ The parents want to know where she is in case of any emergency.
 - ★ Both are legitimate interests!
 - ★ Danger: the parents may overreach – out of concern for their daughter – not respecting her privacy.

An Example [from Mark Ryan]

- Parents and their teenage daughter:
 - ★ The daughter wants to go out, but not tell the parents where she goes.
 - ★ The parents want to know where she is in case of any emergency.
 - ★ Both are legitimate interests!
 - ★ Danger: the parents may overreach – out of concern for their daughter – not respecting her privacy.
- Non-electronic solution:
 - ★ The daughter writes where she is going in a sealed letter.
 - ★ The parents can open the letter, but are compelled not to — unless there is an emergency
- Technical solution: with trusted hardware (e.g. a TPM)

Why Privacy?



Why not vote in public?

Why Privacy?

- Being observed, or believing to be observed, can have an influence on ones behavior.
 - ★ Feeling compelled/coerced to act according to others' expectations.
 - ★ Can mean a subtle restriction of the freedom.
- This is getting more sensitive than 20 years ago. Life leaves more traces in different IT systems. Technology allows for:
 - ★ cheap storage of data
 - ★ cheap evaluations of data
 - ★ cheap surveillance
- Protesting against a totalitarian regime, exchange information with other opposition members.

Outline

① Why Privacy?

② Defining Privacy

③ Privacy in Corona Apps

④ Non-Technical Approaches

⑤ Mixes

⑥ Zero-Knowledge Protocols

Defining Privacy

What *is* privacy really?

Several informal and semi-formal notions:

- Anonymity: the identity of the actors are protected
- Unlinkability: different actions of the same actor cannot be associated
- ...

Formally Defining Privacy

- Difficult: it is not a classical secrecy problem!
 - ★ Example: poll where you can vote *yes* or *no*
 - ★ *yes* and *no* are values that the intruder knows, also the names of the voters may be no secrets.
 - ★ What is actually secret is: who voted *yes* and who voted *no*!

Formally Defining Privacy

- Difficult: it is not a classical secrecy problem!
 - ★ Example: poll where you can vote *yes* or *no*
 - ★ *yes* and *no* are values that the intruder knows, also the names of the voters may be no secrets.
 - ★ What is actually secret is: who voted *yes* and who voted *no*!
- Inspiration from Cryptography: security formulated as equivalence notions:

$$\text{crypt}(k, 0) \sim \text{crypt}(k, 1)?$$

The intruder knows that the plain-text is either 0 or 1, the challenge is for him to tell correctly whether it is 0 or 1.

Formally Defining Privacy

- Difficult: it is not a classical secrecy problem!
 - ★ Example: poll where you can vote *yes* or *no*
 - ★ *yes* and *no* are values that the intruder knows, also the names of the voters may be no secrets.
 - ★ What is actually secret is: who voted *yes* and who voted *no*!
- Inspiration from Cryptography: security formulated as equivalence notions:

$$\text{crypt}(k, 0) \sim \text{crypt}(k, 1)?$$

The intruder knows that the plain-text is either 0 or 1, the challenge is for him to tell correctly whether it is 0 or 1.

- Similar in formal verification (with perfect cryptography): static equivalence of frames.

Alpha-Beta Privacy

- Novel approach to formulating privacy:
 - ★ Specify by a logical formula α what information the intruder is allowed to know, e.g. election result:
$$\alpha \equiv v_1, \dots, v_n \in \{0, 1\} \wedge \sum_{i=0}^n v_i = 42$$
 - ★ Specify by a logical formula β what the intruder actually knows, e.g., observed cryptographic messages, what he knows about their structure, etc.
 - ★ Privacy: the intruder cannot derive anything from β except what already follows from α .
- Ongoing research effort by Luca Viganò, Sébastien Gondron, Laouen Fernet and myself.
 - ★ We welcome more colleagues to join us e.g. for a MSc thesis!

Outline

① Why Privacy?

② Defining Privacy

③ Privacy in Corona Apps

④ Non-Technical Approaches

⑤ Mixes

⑥ Zero-Knowledge Protocols

Privacy in Corona Apps

Several Corona-Apps for contact tracing using Bluetooth Low Energy have been developed:

- DP3-T: Decentralized Privacy-Preserving Proximity Tracing
 - ★ open platform
 - ★ developed by several universities and research centers
 - ★ used by many European countries
- GAEN: Apple/Google Exposure Notification:
 - ★ very similar to DP3-T
 - ★ by Apple & Google
 - ★ used for instance in Denmark
- PEPP-PT/PEPP: Pan-European Privacy-Preserving Proximity Tracing
 - ★ Different idea: more information centrally stored
 - ★ Many developers left the consortium (and did DP3-T)

DP3-T

- Clients generate day keys and ephemeral identities.
- Day keys: $SK_{i+1} := h(SK_i)$.
- Ephemeral identities: new identity in short intervals, say 15 minutes. $EphID_{i,j} := prg(SK_i, j)$ for the j -th period on day i .
- Normal operation: exchange current ephemeral ID with devices in proximity, store all received ephemeral IDs that were for ca. 10 minutes in proximity.
- When sick and user gives consent, publish the day keys SK_i for the relevant days on a server.
 - ★ In most circumstances, the server will also need a proof that the submitter has indeed tested positive.
- Every phone can download the published SK_i , compute the ephemeral IDs and compare them to the proximity list.
- How much is privacy is sacrificed here? What can an attacker learn about infected people and their identities?

Outline

- ① Why Privacy?
- ② Defining Privacy
- ③ Privacy in Corona Apps
- ④ Non-Technical Approaches
- ⑤ Mixes
- ⑥ Zero-Knowledge Protocols

The Law

Data Protection Laws

GDPR regulations of the EU

Personal Data must be “handled with care”

- Personal data includes e.g. name, date of birth, CPR-number, address, medical data, ...
- Always think of the **purpose** of storing such data
 - ★ Why the data has to be stored?
 - ★ Who shall have access to it?
 - ★ How long shall it be stored?
- Protect reasonably against theft
- Transparency
 - ★ Every person has the right to their data
 - ★ Consensus, information what is stored, right to correct errors, and right to be forgotten

The Tom Waits Approach



*I was born in the back seat of a Yellow Cab
in a hospital loading zone and with the meter
still running. I emerged needing a shave and
shouted 'Time Square, and step on it!'*

The Tom Waits Approach



*I was born in the back seat of a Yellow Cab
in a hospital loading zone and with the meter
still running. I emerged needing a shave and
shouted 'Time Square, and step on it!'*

- Tom Waits has told many absurd stories about his life.
- Hard to tell what is true. (Was he born in a taxi?)

Outline

① Why Privacy?

② Defining Privacy

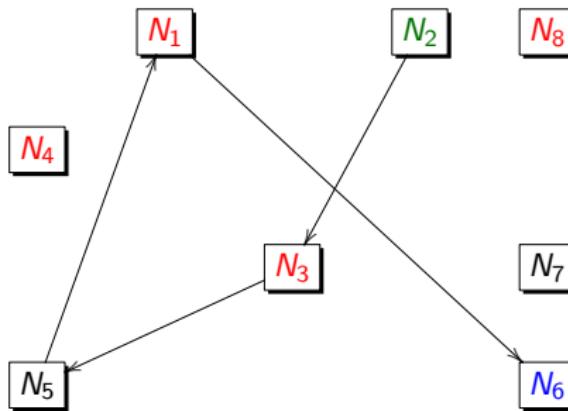
③ Privacy in Corona Apps

④ Non-Technical Approaches

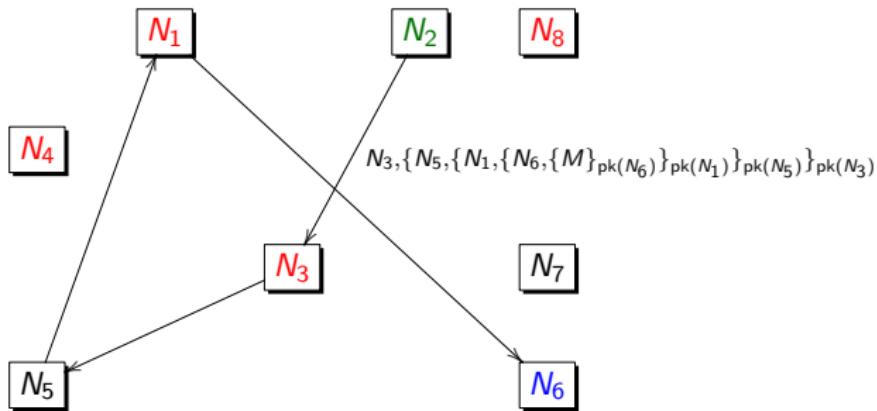
⑤ Mixes

⑥ Zero-Knowledge Protocols

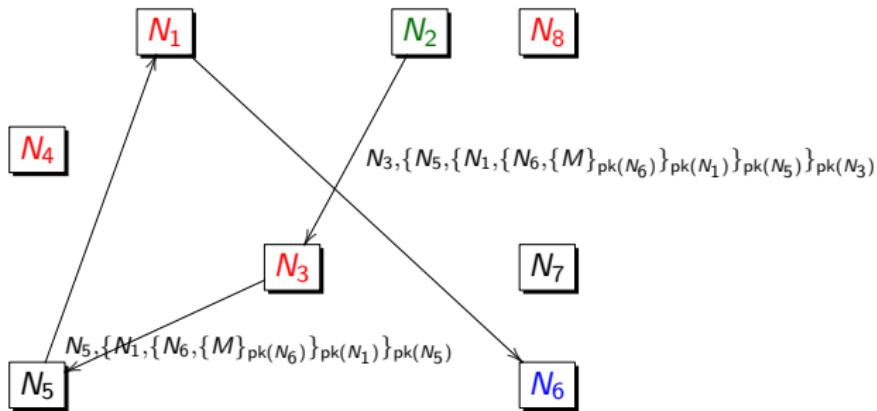
Mixes/Onion Routing



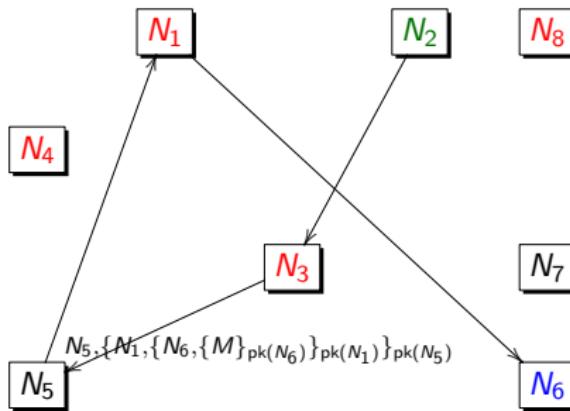
Mixes/Onion Routing



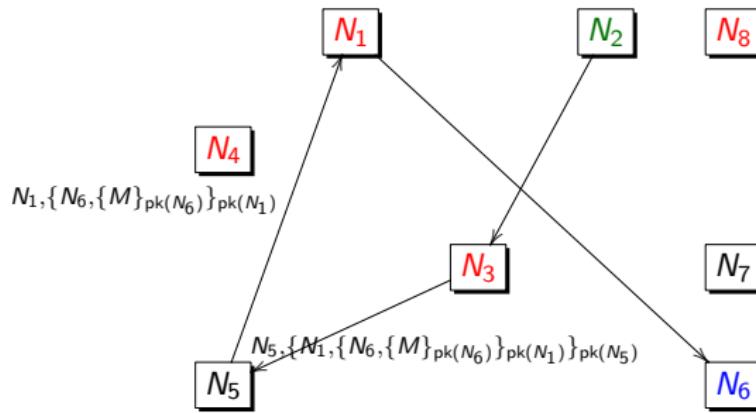
Mixes/Onion Routing



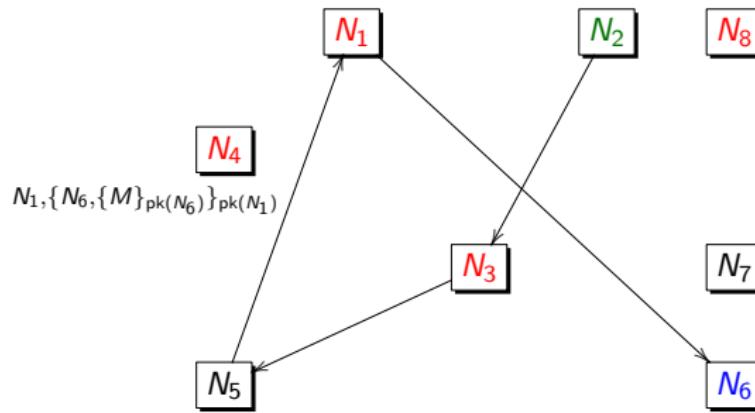
Mixes/Onion Routing



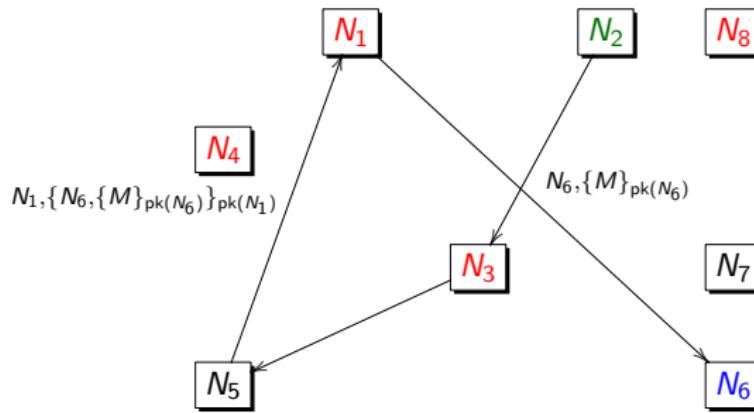
Mixes/Onion Routing



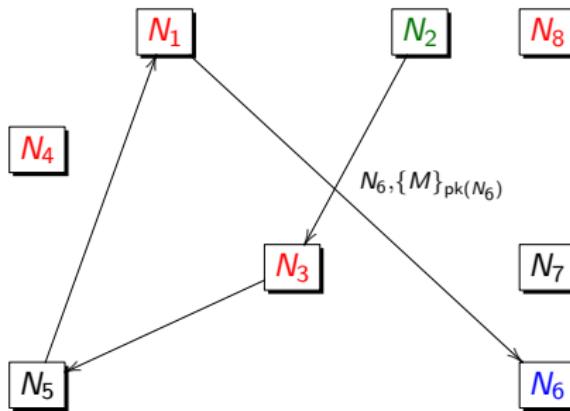
Mixes/Onion Routing



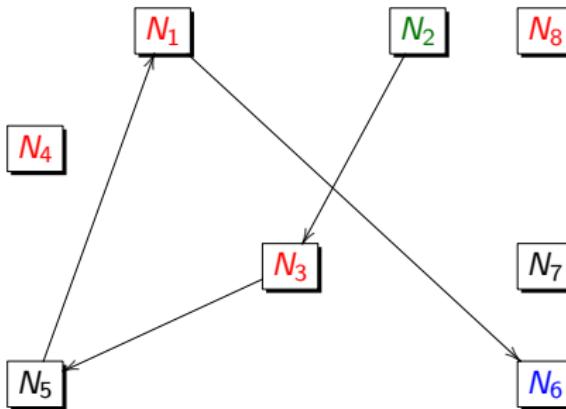
Mixes/Onion Routing



Mixes/Onion Routing

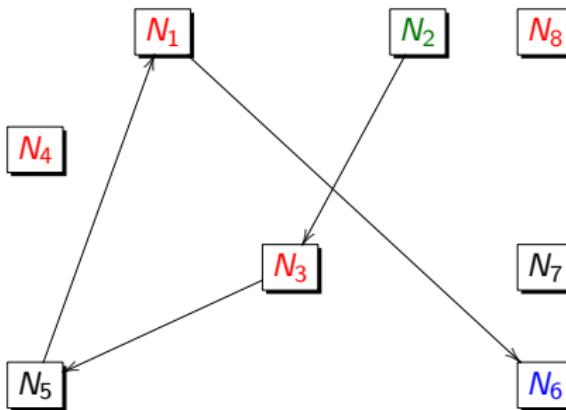


Mixes/Onion Routing



- Each node knows only who is the previous/next on the chain
- Only the **destination** learns the message M
- Nobody knows who is **source** except the source itself
- Nobody knows who is the **destination** (except source and destination)
- Attacker model: can see all exchanged messages and may control **some nodes** (may include **destination**)

Mixes/Onion Routing: Assumptions



- Sender knows the true public key of all nodes.
- At least one node on the path is not controlled by the attacker
- Traffic is evenly distributed between all nodes
- Replay of messages is prevented
- Length of an encrypted message does not reveal the number of encryption layers.
- Note also: if **destination** is not part of mix network, cleartext messages are transmitted on the last leg.

Outline

- ① Why Privacy?
- ② Defining Privacy
- ③ Privacy in Corona Apps
- ④ Non-Technical Approaches
- ⑤ Mixes
- ⑥ Zero-Knowledge Protocols

Idea

Zero-knowledge proofs

In zero-knowledge proofs we can usually specify a **statement** that is being proved.

- Definitely, that statement is revealed to the verifier
- The verifier (or others) should not learn anything else
- Everybody can draw conclusions from everything they learned

Zero-Knowledge Protocols

Scenario

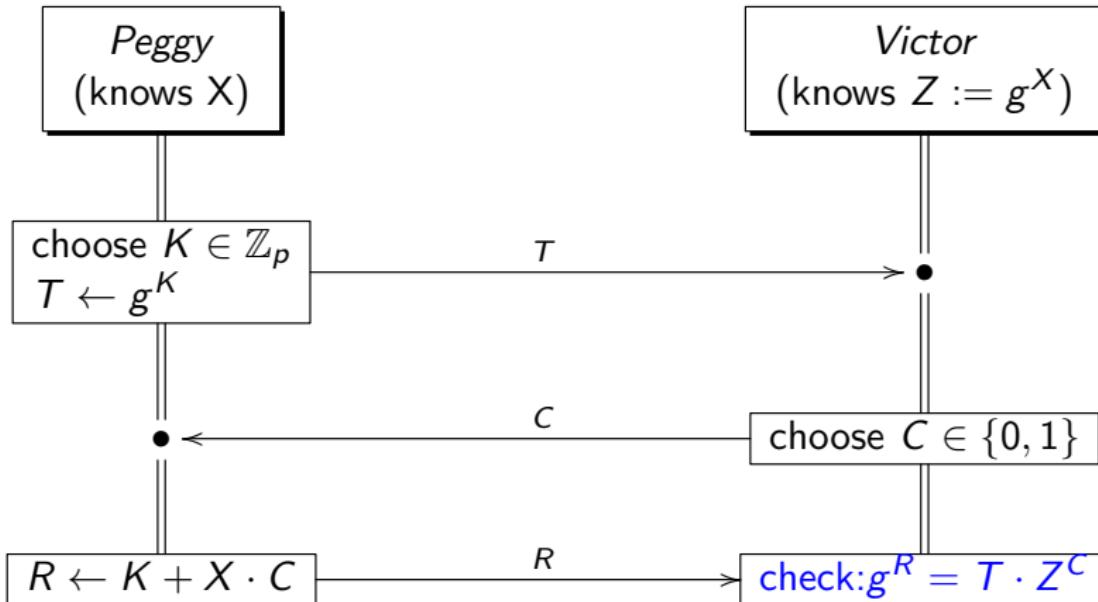
- A Prover Peggy and a Verifier Victor
- Peggy has a secret and wants to convince Victor of that fact **without** telling the secret.
- Example Schnorr protocol:
 - ★ Victor knows a public value $Z \in \mathbb{Z}_p^*$.
 - ★ Peggy wants to prove that she **knows** $X \in \mathbb{Z}_p^*$ with $Z \equiv_p g^X$.
- Why? What's the point of proving this?

Zero-Knowledge Protocols

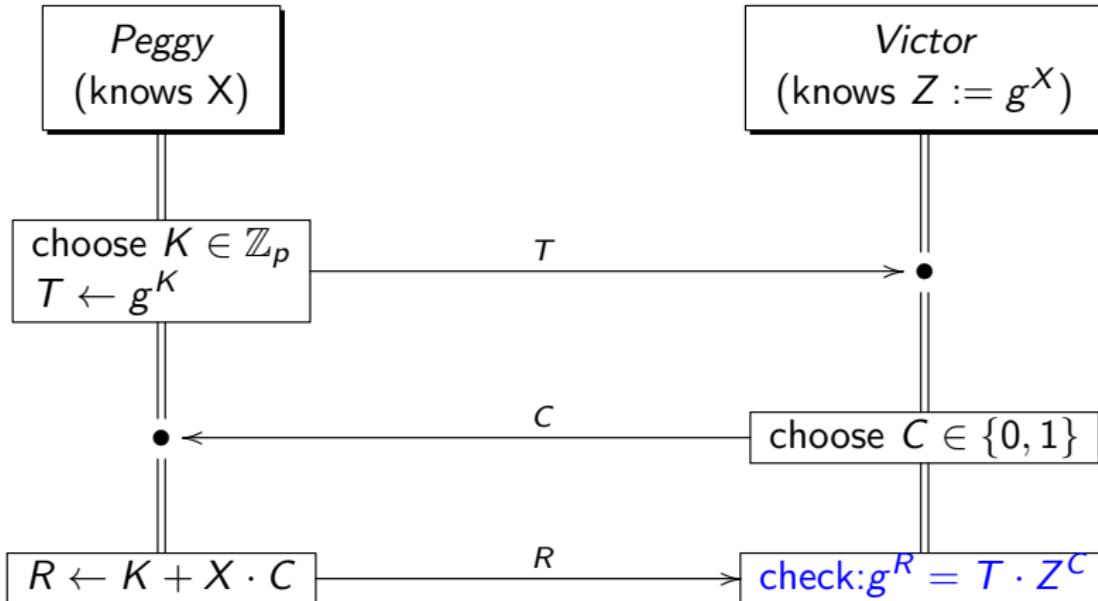
Scenario

- A Prover Peggy and a Verifier Victor
- Peggy has a secret and wants to convince Victor of that fact **without** telling the secret.
- Example Schnorr protocol:
 - ★ Victor knows a public value $Z \in \mathbb{Z}_p^*$.
 - ★ Peggy wants to prove that she **knows** $X \in \mathbb{Z}_p^*$ with $Z \equiv_p g^X$.
- Why? What's the point of proving this?
 - ★ Peggy can authenticate this way
 - ★ Anonymous credential systems ("Private authentication")
 - ★ Voting protocols like Helios/Belenios
 - ★ Alternatives to Proof-of-Work in blockchain implementations.

Schnorr: the Protocol



Schnorr: the Protocol



If Peggy worked correctly:

$$g^R = g^{K+X \cdot C} = g^K \cdot g^{X \cdot C} = T \cdot Z^C$$

Schnorr: Trying to Cheat

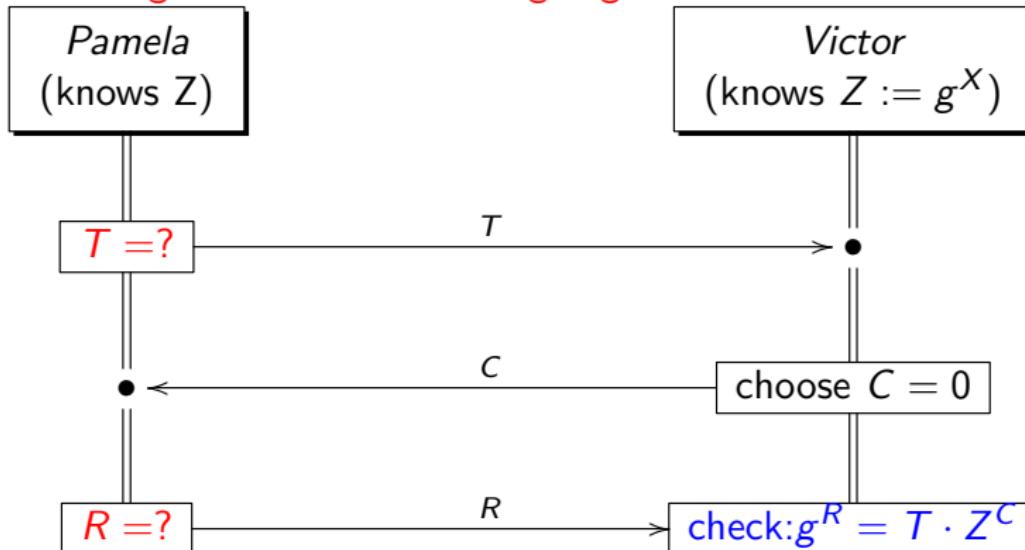
Pamela does not know the secret, but tries to prove its knowledge.

Pamela guesses that Victor is going to choose $C = 0$

Schnorr: Trying to Cheat

Pamela does not know the secret, but tries to prove its knowledge.

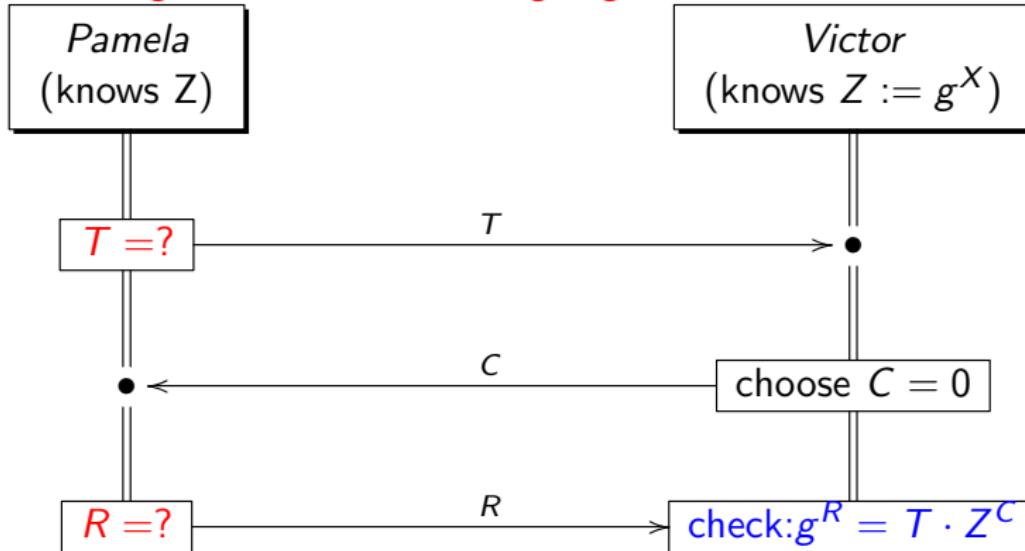
Pamela guesses that Victor is going to choose $C = 0$



Schnorr: Trying to Cheat

Pamela does not know the secret, but tries to prove its knowledge.

Pamela guesses that Victor is going to choose $C = 0$



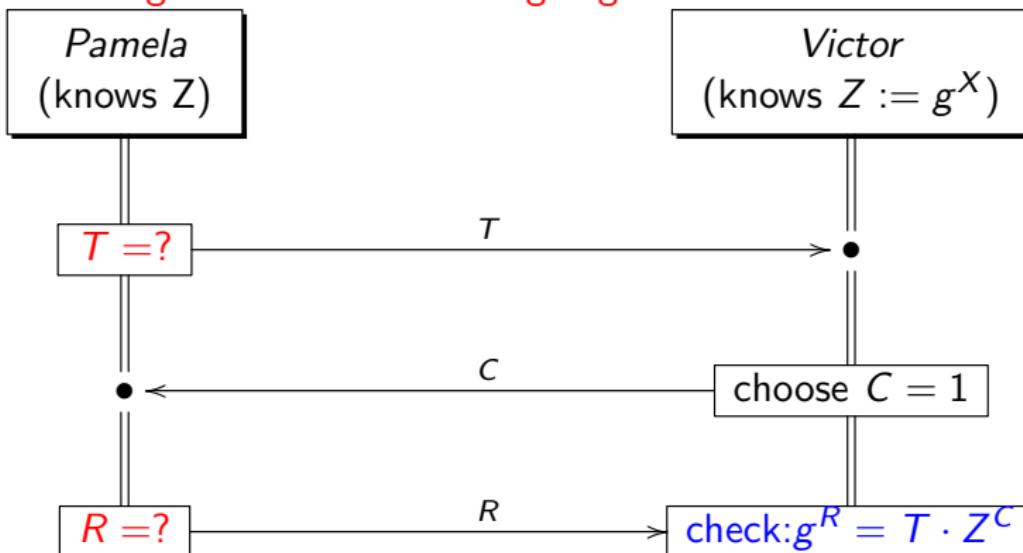
Strategy for Pamela: Choose R randomly and set $T = g^R$.

Schnorr: Trying to Cheat

Pamela guesses that Victor is going to choose $C = 1$

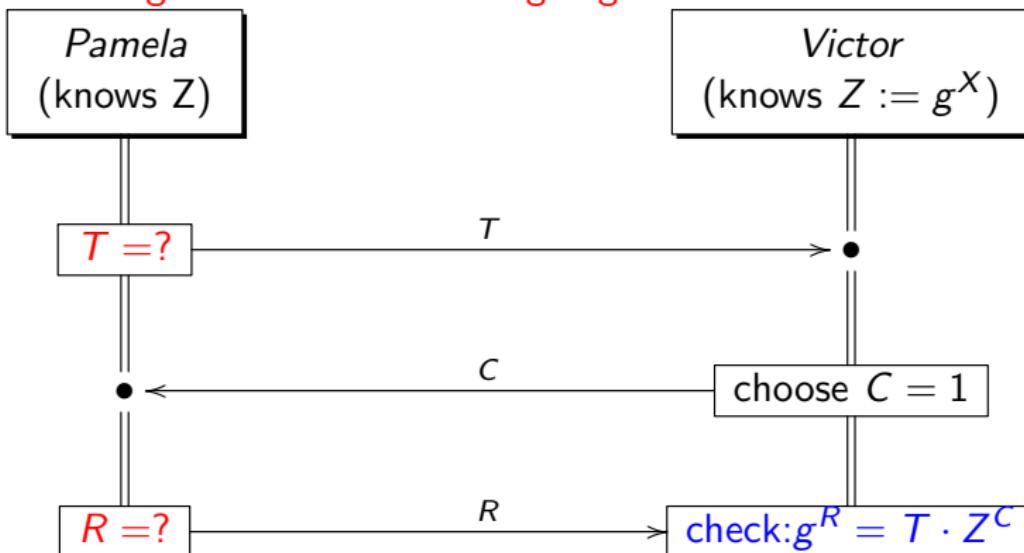
Schnorr: Trying to Cheat

Pamela guesses that Victor is going to choose $C = 1$



Schnorr: Trying to Cheat

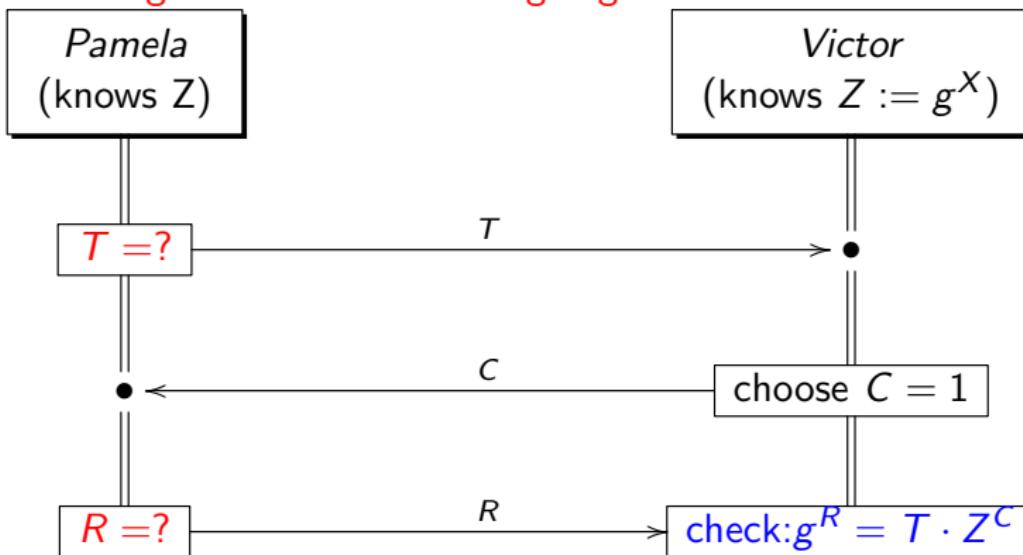
Pamela guesses that Victor is going to choose $C = 1$



Find numbers T and R such that $g^R = T \cdot Z$.

Schnorr: Trying to Cheat

Pamela guesses that Victor is going to choose $C = 1$

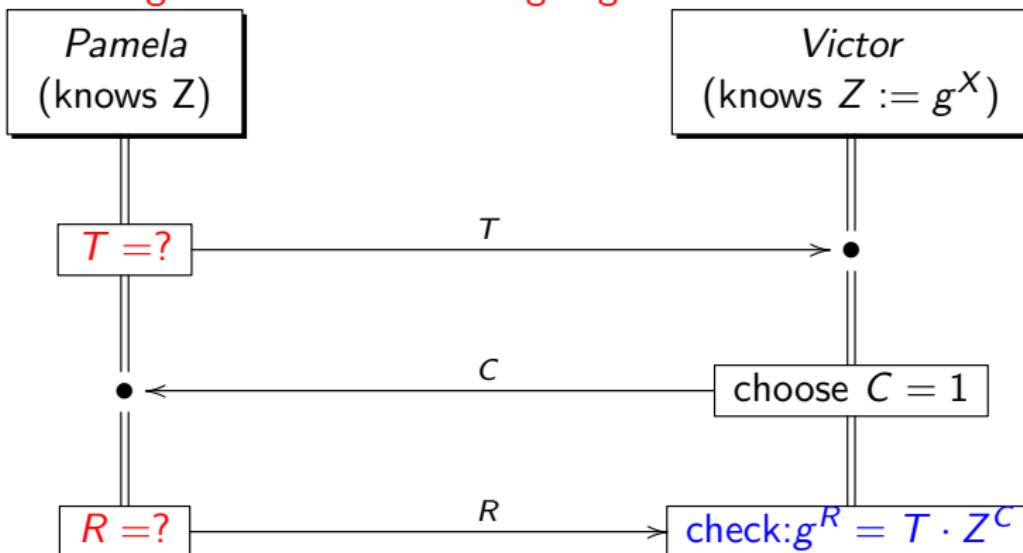


Find numbers T and R such that $g^R = T \cdot Z$.

Hint: division modulo p is also easy!

Schnorr: Trying to Cheat

Pamela guesses that Victor is going to choose $C = 1$



Find numbers T and R such that $g^R = T \cdot Z$.

Hint: division modulo p is also easy!

Strategy for Pamela: Choose R randomly, set $T = \frac{g^R}{Z}$.

Schnorr: Trying to Cheat

- Pamela has a strategy to cheat if $C = 0$:

Choose R randomly, set $T = g^R$.

Schnorr: Trying to Cheat

- Pamela has a strategy to cheat if $C = 0$:

Choose R randomly, set $T = g^R$.

- Pamela has a strategy to cheat if $C = 1$:

Choose R randomly, set $T = \frac{g^R}{Z}$

Schnorr: Trying to Cheat

- Pamela has a strategy to cheat if $C = 0$:

Choose R randomly, set $T = g^R$.

- Pamela has a strategy to cheat if $C = 1$:

Choose R randomly, set $T = \frac{g^R}{Z}$

- Since T must be sent before C is known, she can **prepare** only for one possibility
 - ★ Chance of $\frac{1}{2}$ to cheat (if $C \in \{0, 1\}$ uniformly chosen)

Schnorr: Trying to Cheat

- Pamela has a strategy to cheat if $C = 0$:

Choose R randomly, set $T = g^R$.

- Pamela has a strategy to cheat if $C = 1$:

Choose R randomly, set $T = \frac{g^R}{Z}$

- Since T must be sent before C is known, she can **prepare** only for one possibility
 - ★ Chance of $\frac{1}{2}$ to cheat (if $C \in \{0, 1\}$ uniformly chosen)
- Claim: Pamela cannot do better (to prove...)

Schnorr: Trying to Cheat

- Pamela has a strategy to cheat if $C = 0$:

Choose R randomly, set $T = g^R$.

- Pamela has a strategy to cheat if $C = 1$:

Choose R randomly, set $T = \frac{g^R}{Z}$

- Since T must be sent before C is known, she can **prepare** only for one possibility
 - ★ Chance of $\frac{1}{2}$ to cheat (if $C \in \{0, 1\}$ uniformly chosen)
- Claim: Pamela cannot do better (to prove...)
- If Victor accepts only after n successful rounds of the protocol, the chance to cheat is only 2^{-n} .

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.
- Then she has values T , R_0 , and R_1 such that
 - ★ (If Victor chooses $C = 0$:) $g^{R_0} = T$
 - ★ (If Victor chooses $C = 1$:) $g^{R_1} = T \cdot Z$

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.
- Then she has values T , R_0 , and R_1 such that
 - ★ (If Victor chooses $C = 0$:) $g^{R_0} = T$
 - ★ (If Victor chooses $C = 1$:) $g^{R_1} = T \cdot Z$
- Then she can compute $Q = R_1 - R_0$.

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.
- Then she has values T , R_0 , and R_1 such that
 - ★ (If Victor chooses $C = 0$:) $g^{R_0} = T$
 - ★ (If Victor chooses $C = 1$:) $g^{R_1} = T \cdot Z$
- Then she can compute $Q = R_1 - R_0$.
- It holds that $g^Q = g^{R_1 - R_0} = \frac{g^{R_1}}{g^{R_0}} = \frac{T \cdot Z}{T} = Z$.

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.
- Then she has values T , R_0 , and R_1 such that
 - ★ (If Victor chooses $C = 0$:) $g^{R_0} = T$
 - ★ (If Victor chooses $C = 1$:) $g^{R_1} = T \cdot Z$
- Then she can compute $Q = R_1 - R_0$.
- It holds that $g^Q = g^{R_1 - R_0} = \frac{g^{R_1}}{g^{R_0}} = \frac{T \cdot Z}{T} = Z$.
- So with $Q = X$ she indeed knows the discrete logarithm of Z .

Schnorr: Trying to Cheat

Claim

Pamela has no strategy to cheat for unpredictable C .

Proof sketch

- Suppose Pamela has a strategy that works both for $C = 0$ and for $C = 1$.
- Then she has values T , R_0 , and R_1 such that
 - ★ (If Victor chooses $C = 0$:) $g^{R_0} = T$
 - ★ (If Victor chooses $C = 1$:) $g^{R_1} = T \cdot Z$
- Then she can compute $Q = R_1 - R_0$.
- It holds that $g^Q = g^{R_1 - R_0} = \frac{g^{R_1}}{g^{R_0}} = \frac{T \cdot Z}{T} = Z$.
- So with $Q = X$ she indeed knows the discrete logarithm of Z .
- **So it is not cheating!**

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

Consider a transcript of an exchange between Peggy and Victor. (Assume Victor chooses challenges randomly.)

T_1	C_1	R_1
T_2	C_2	R_2
\dots		
T_n	C_n	R_n

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

Consider a transcript of an exchange between Peggy and Victor. (Assume Victor chooses challenges randomly.)

T_1	C_1	R_1
T_2	C_2	R_2
...		
T_n	C_n	R_n

Create a fake-transcript using random challenges C'_i

C'_1
C'_2
...
C'_n

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

Consider a transcript of an exchange between Peggy and Victor. (Assume Victor chooses challenges randomly.)

$$\begin{array}{lll} T_1 & C_1 & R_1 \\ T_2 & C_2 & R_2 \\ \dots & & \\ T_n & C_n & R_n \end{array}$$

Create a fake-transcript using random challenges C'_i and then filling the T'_i and R'_i according to the cheat strategies of Pamela for known challenges:

$$\begin{array}{lll} T'_1 & C'_1 & R'_1 \\ T'_2 & C'_2 & R'_2 \\ \dots & & \\ T'_n & C'_n & R'_n \end{array}$$

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

- Give the two transcripts to a third party. Can one tell the real from the fake?

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

- Give the two transcripts to a third party. Can one tell the real from the fake?
- **No:** statically indistinguishable. (Not even distinguishable with unbounded computing resources!)

Schnorr: Curious Victor

Victor would like to learn the secret X ...

Zero-Knowledge Property

Victor learns nothing except the proved statement.

Proof sketch

- Give the two transcripts to a third party. Can one tell the real from the fake?
- **No:** statically indistinguishable. (Not even distinguishable with unbounded computing resources!)
- So, whatever information Victor is getting out of the transcript, he may have created that himself without Peggy!

The Mafia-owned Restaurant

Claim (Shamir)

I can go to a Mafia-owned restaurant and pay with my zero-knowledge proof-based credit card frequently, and yet the mobsters cannot impersonate me.

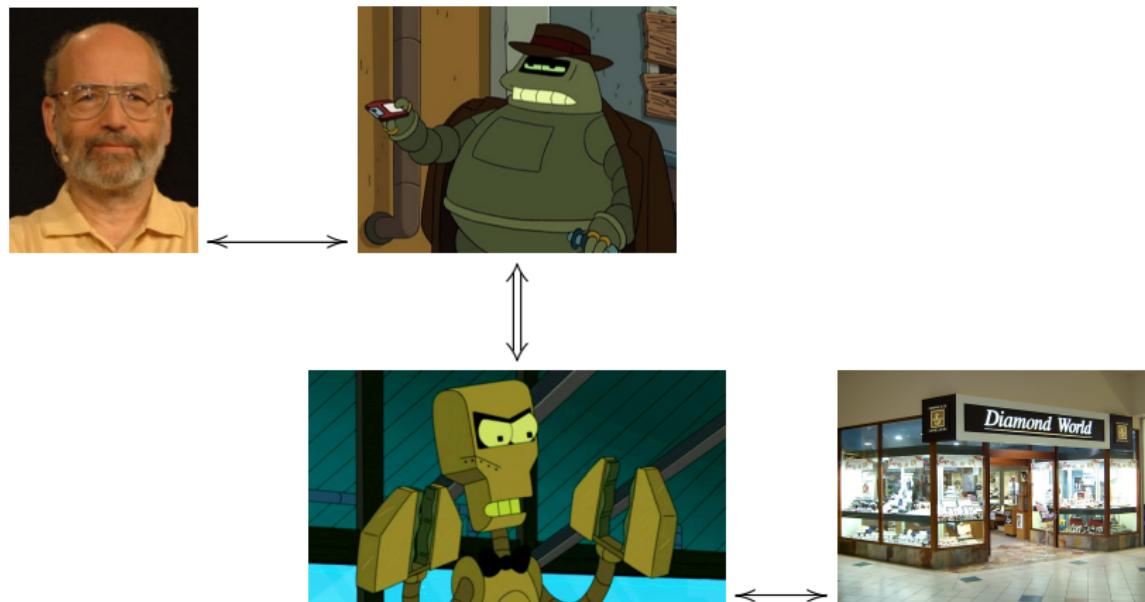
The Mafia-owned Restaurant

Claim (Shamir)

I can go to a Mafia-owned restaurant and pay with my zero-knowledge proof-based credit card frequently, and yet the mobsters cannot impersonate me.

They can, if they do it online.

The Mafia Attack



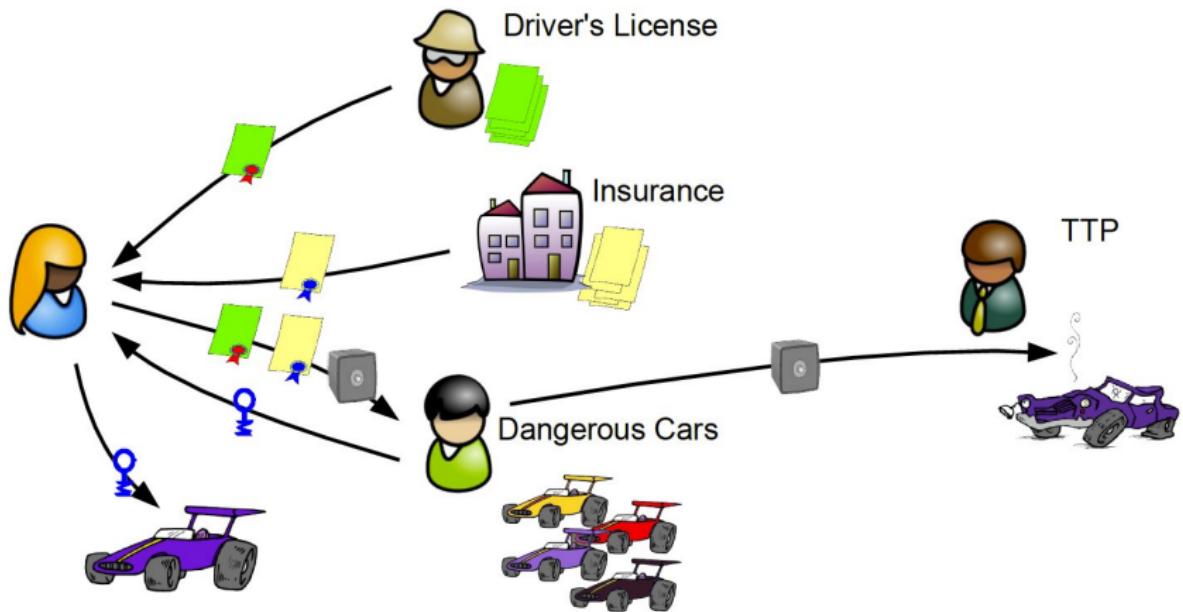
Classical Man in the middle attack: forward all messages!

Countermeasures

- Distance bounding protocols.
- Link the intended verifier to the statement being proved.

IBM's Idemix

A privacy friendly credential system



IBM's Idemix

- Credentials: list of attribute-value pairs, signed by an issuer.
`sign(copenhagen-commune; name="Johanna Gossner", birthdate=14/12/73, kind="Drivers License", class="3", ...)`
- Signatures are special: Camenisch-Lysyanskaya. Allows for special zero-knowledge proofs
 - ★ Proving possession of credential without transmitting it
 - ★ Revealing only some attributes (e.g. kind and class)
 - ★ Proving properties about attributes (e.g. birthdate < 9/10/2000)
 - ★ Proving relations between credentials (e.g. passport on same name)
 - ★ Signing a statement (e.g. "I confirm this electronic order...")

IBM's Idemix

- Credentials: list of attribute-value pairs, signed by an issuer.
`sign(copenhagen-commune; name="Johanna Gossner", birthdate=14/12/73, kind="Drivers License", class="3", ...)`
- Signatures are special: Camenisch-Lysyanskaya. Allows for special zero-knowledge proofs
 - ★ Proving possession of credential without transmitting it
 - ★ Revealing only some attributes (e.g. kind and class)
 - ★ Proving properties about attributes (e.g. birthdate < 9/10/2000)
 - ★ Proving relations between credentials (e.g. passport on same name)
 - ★ Signing a statement (e.g. "I confirm this electronic order...")
- Privacy: A verifier cannot find out more than what was proved.
 - ★ E.g. different transactions are unlinkable (unless linked by prover).
 - ★ Holds even if the issuer is dishonest and collaborates with a dishonest verifier.

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
- but not the privacy of the guilty?

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
 - but not the privacy of the guilty?
-
- Can be achieved using escrow with verifiable encryptions

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
 - but not the privacy of the guilty?
-
- Can be achieved using escrow with **verifiable encryptions**
 - Provers have to encrypt their real names with the public key of an authority.

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
 - but not the privacy of the guilty?
-
- Can be achieved using escrow with **verifiable encryptions**
 - Provers have to encrypt their real names with the public key of an authority.
 - Provers have to prove (in zero-knowledge) that they have encrypted the real name (and it is correctly encrypted).

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
 - but not the privacy of the guilty?
-
- Can be achieved using escrow with **verifiable encryptions**
 - Provers have to encrypt their real names with the public key of an authority.
 - Provers have to prove (in zero-knowledge) that they have encrypted the real name (and it is correctly encrypted).
 - Then the authority can **revoke** their privacy when needed.

Idemix: Privacy with Accountability?

How can we ensure that we

- protect the privacy of the innocent
 - but not the privacy of the guilty?
-
- Can be achieved using escrow with **verifiable encryptions**
 - Provers have to encrypt their real names with the public key of an authority.
 - Provers have to prove (in zero-knowledge) that they have encrypted the real name (and it is correctly encrypted).
 - Then the authority can **revoke** their privacy when needed.
 - This should be tied to a legal system, e.g. the privacy revocation happens only on court order.

Administering Security



$$\int_a^b \mathcal{E}^{\theta} = \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} f^{(n)}(x) + \Omega \int_a^b \delta e^{i\pi} = \{2.7182818284\}$$

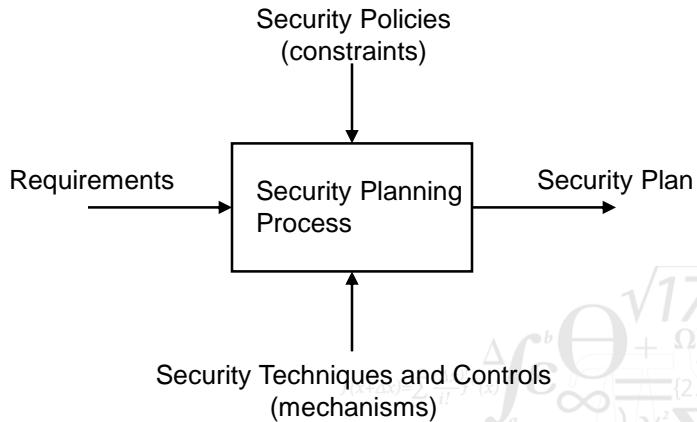
DTU Compute

Department of Applied Mathematics and Computer Science

Four Areas of Security Administration

- Security planning
 - Advance preparation for when things go wrong
 - Define measurable objectives
 - Establish procedures for system evolution
- Security policy definition and enforcement
 - Define goals and ensure that these goals are continuously met
- Physical security
 - Aspects of the physical environment that impact on security
- Risk analysis
 - Analyse threats
 - Weigh cost and benefits of controls

Security Planning Process



Contents of a Security Plan

- **Policy definition:** establish the overall security goals
- **Establish base line:** describe the current state of the system to provide a base line for the work
- **Identify requirements:** identify steps needed to achieve the defined security goals
- **Identify recommended controls:** identify controls to vulnerabilities defined in policy and requirements
- **Establish accountability:** delegate responsibility
- **Define timetable:** define deadlines for phased implementation
- **Show continual vigilance:** define responsibility for maintaining security when system evolves

Defining the Security Policy

- Security policies must have support from the top management
 - Ultimate arbiter in conflicts between security and business goals
 - Ensures buy-in from all levels in the organisation
 - *Requires that CEO walks the talk*
 - Allocates funds for the security work (people and equipment)
- Security policies must balance the need for security with the inconvenience to the users
 - Security mechanisms that are not understood will not be used
 - Security mechanisms that are not used will not be effective
 - Security mechanisms that are not effective are a waste of time and money



5 DTU Compute Technical University of Denmark

02239 – Data Security

Contents of a security Policy

- Security policy should specify:
 - The organizations *goals* for security, e.g., confidentiality, integrity and availability, and their relative importance
 - Responsibility for enforcing security policies
 - *Historically IT Department*
 - *Increasingly responsibility of Finance Department*
 - The organizations commitment to security
- Overall policy must include an access control policy:
 - Who should be allowed access to the system?
 - To what systems and resources should access be allowed?
 - What type of access should be allowed for each resource

6 DTU Compute Technical University of Denmark

02239 – Data Security

Defining the Security Requirements

- The security requirements must address all issues mentioned in the security policy
- Important characteristics:
 - *Correctness*: are requirements understandable and correct?
 - *Consistency*: are there conflicting or ambiguous requirements?
 - *Completeness*: are all possibilities covered?
 - *Realism*: is it possible to achieve the requirements?
 - *Need*: are the requirements unnecessarily restrictive?
 - *Verifiability*: can tests be written to verify that the requirements are met?
 - *Traceability*: can all requirements be traced to the related controls, so that changes in requirements can be easily implemented

Responsibility for Implementation

- Assigning responsibility must also identify coordinators for implementation of the security requirements
 - Who checks for new vulnerabilities
 - Who implements new controls when new vulnerabilities are discovered
- Different responsibilities may be assigned to different groups of users:
 - PC users may be responsible for their own PC
 - Project leaders may be responsible for the data and computers allocated to the project
 - Managers may be responsible for overseeing their project leaders
 - Database administrators are responsible for the DB system

Ensuring commitment to Security Plan

- Acceptance by the organisation is required for the implementation of any security plan
 - Security team must ensure backing from management
 - Security team must be sensitive to the needs of the group affected by the security policy
 - Those affected by the security policy must understand the importance of the imposed controls
 - *This includes awareness running programs*



- Environments change, so security policies must be able to evolve
 - This requires periodic review of security policies, including establishment of a new baseline
 - Evaluation and modification of unfortunate policies

Writing Security Policies

- Security policies are written for several purposes
 - Identify sensitive information assets
 - Clarify security responsibilities
 - Promote awareness for existing employees
 - Guiding new employees
- Security policies have several audiences
 - Users
 - Owners
 - Beneficiaries
 - It is important to balance the interest of all parties

Content of a Security Policy

- State purpose
 - Promote efficient business operation
 - Facilitate sharing of information throughout the organization
 - Safeguard business and personal information
 - Ensure accurate information is available to support business decisions
 - Ensure a safe and productive place to work
 - Comply with applicable laws and regulations
- Protected Resources
 - Explicitly list assets covered by the policy (*which resources*)
- Nature of Protection
 - Indicate *who* should have access and how access is granted/denied

Characteristics of a Good Security Policy

- Coverage
 - The policy should be comprehensive - any incident that may occur should be covered by the policy (generality)
- Durability
 - The policy should evolve with the system, but remain largely unchanged (keep it general)
- Realism
 - The policy should not set unobtainable goals or impose unworkable restrictions
- Usefulness
 - The policy should be seen to be useful, otherwise it will be ignored

Physical Security

- Physical security encompass all aspects of security that involve physical controls: walls, locks, guards, emergency generators, fire inhibitors, ...
- Physical security should consider:
 - Break-ins, theft, espionage, ...
 - Natural disasters
 - Loss of power
 - Loss of communication
 - Human vandals
 - Interception of sensitive information



Natural disasters; threats to availability

- Floods
 - Natural floods come from rising water (ground water, rivers)
 - Flooding is normally slow and some precautions can be made
 - Sudden floods come from bursting dams, water mains, ...
 - Flooding is difficult to prevent and few precautions can be made
- Fires
 - Fire is often sudden
 - Short time to react
 - Extinguished with water (implies flooding as well)
 - Equipment can be damaged by smoke as well as fire
- Storms, earthquakes, volcanoes
 - Often determined by geographical location

Loss of power and Communications

- Uninterruptible power supply (UPS)
 - Stores energy (battery or wheel) that can be released if the power is cut - normally only allows a clean shutdown
 - UPS can be complemented by a backup generator
- Surge suppressor
 - Some countries have problems with sudden drops, spikes and surges in electrical power, which may damage electrical equipment - surge suppressors limit variations in power
- Multiple ISPs
 - Prevent breakdown if one ISP fails
- Multiple phone lines?
 - In Europe one company (the national PTT) has historically installed the infrastructure and other operators simply lease access to the wire
 - this is particularly true of the local loop

Dependability of Data Center

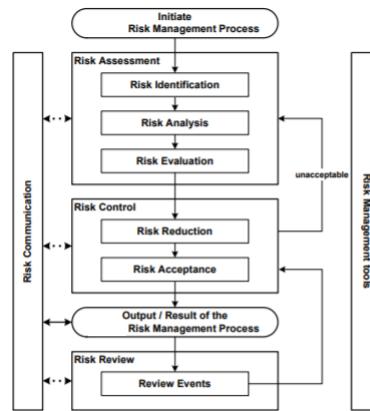
Tier	Requirement
I	<ul style="list-style-type: none">• Single non-redundant distribution path serving the critical loads• Non-redundant critical capacity components
II	<ul style="list-style-type: none">• Meets all Tier I requirements, in addition to:• Redundant critical capacity components• Critical capacity components must be able to be isolated and removed from service while still providing N capacity to the critical loads.
III	<ul style="list-style-type: none">• Meets all Tier II requirements in addition to:• Multiple independent distinct distribution paths serving the IT equipment critical loads• All IT equipment must be dual-powered provided with two redundant, distinct UPS feeders. Single-corded IT devices must use a Point of Use Transfer Switch to allow the device to receive power from and select between the two UPS feeders.• Each and every critical capacity component, distribution path and component of any critical system must be able to be fully compatible with the topology of a site's architecture isolated for planned events (replacement, maintenance, or upgrade) while still providing N capacity to the critical loads.• Onsite energy production systems (such as engine generator systems) must not have runtime limitations at the site conditions and design load.
IV	<ul style="list-style-type: none">• Meets all Tier III requirements in addition to:• Multiple independent distinct and active distribution paths serving the critical loads• Compartmentalization of critical capacity components and distribution paths• Critical systems must be able to autonomously provide N capacity to the critical loads after any single fault or failure• Continuous Cooling is required for IT and UPS systems.

Human Vandals

- Unauthorised access and use
 - Hire a guard to prevent outsiders from accessing the site
 - Lock servers in rooms that can only be accessed by authorised personnel
 - Smart cards, RFID or biometrics
- Theft
 - An increasing problem with ubiquitous computing
 - Example: loss of a MI5 laptop with classified information on it
 - Locks or “screamers” can be used to reduce the risk of theft
 - Smart cards, RFID or biometrics

Interception of sensitive information

- Be careful when disposing of physical media that contains sensitive information
 - A Danish credit information company (RKI) dumped all their paper files in the garbage where journalists got hold of them
 - Recycled hard disks may contain sensitive information
- Shred all paper
 - Especially sensitive information can be burned after shredding
- Overwriting magnetic media
 - Magnetic media retains some magnetisation even after several overwrite - smash media after overwriting
- Degaussing
 - Create a magnetic field that realign magnetic charges
- Emanation protection: Tempest

Break**It's time for my break****Basic Elements of Risk Management**

Risk Analysis

- Three elements have to be identified:
 - Loss associated with an event (*risk impact or cost*)
 - Likelihood that the event will occur
 - Degree to which we can change the outcome (*risk control*)
 - *By reducing cost or decreasing likelihood of a particular event*
- Exposure associated with a particular event (aka. risk)
 - Cost of event × likelihood of event
- Three strategies for dealing with risk
 - Avoiding the risk by changing the requirements
 - Transferring the risk, e.g., through insurance
 - Assuming the risk by accepting it; control it with the available resources and accept the loss if the event occurs
- Risk leverage is the relative benefit of reducing risk

$$\frac{(\text{exposure before reduction}) - (\text{exposure after reduction})}{(\text{cost of risk reduction})}$$

Difficulty of Measuring Risk

- Estimating likelihood of threats
 - Precise models must include everything
 - *It must effectively describe the whole world*
 - How relevant is past data to the calculation of future probabilities?
 - *The nature of future attacks is unpredictable*
 - *The actions of future attackers are unpredictable*
 - Subjective probabilities look most promising
- Businesses want to measure “costs” in money, but
 - Many assets are difficult to measure in this way
 - *Value of data and in-house software - no market value*
 - *Value of goodwill and customer confidence*
- Measurement of benefit from security measures
 - Problems with the difference of two approximate quantities
 - *How does an extra security measure affect a ~10⁻⁵ probability of an attack?*

Establish Risk Levels

- Precise monetary values give a false sense of precision
- Better to use levels:
 - High, Medium, Low or Essential-high-medium-low-not important
 - High: major impact on the organisation*
 - Medium: noticeable impact ("material" in auditing terms)*
 - Low: can be absorbed without difficulty*
 - Numerical rating: rating on a scale of 1 – 10
- 3x3 matrix
 - Commonly used in industry
 - Difficult to decide priority

Consequence = High,
Probability = Med

Consequence = Med,
Probability = High

	Probability		
High	3	6	9
Med	2	4	6
Low	1	2	3
	Low	Med	High

Consequence

23 DTU Compute Technical University of Denmark

02239 – Data Security

Basic Steps of Risk Analysis

- Identify assets
- Determine possible vulnerabilities
- Estimate likelihood of exploitation
- Compute expected annual loss
- Survey applicable controls and their costs
- Project annual savings of control



24 DTU Compute Technical University of Denmark

02239 – Data Security

Identify Assets

- **Hardware:** computers, peripheral equipment, communication infrastructure
- **Software:** source code, binary programs, operating systems
- **Data:** temporary data, stored data, printed data, backup media
- **People:** skills needed to run systems or programs
- **Documentation:** hardware, software, administrative procedures
- **Supplies:** paper, forms, toner cartridges, magnetic media (including CD-ROMs)

Determine Vulnerabilities

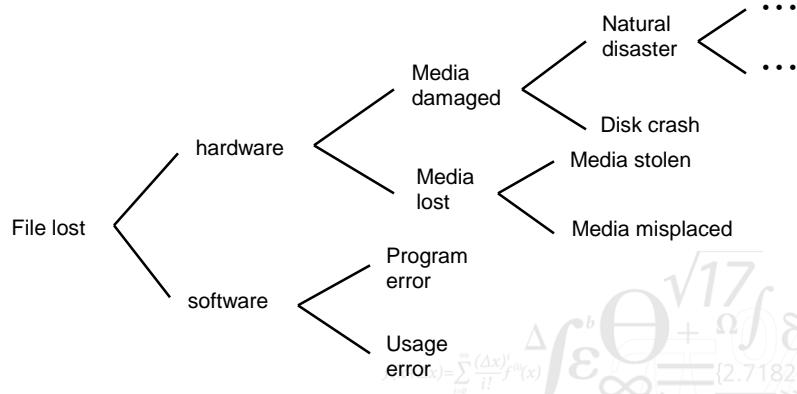
Asset	Confidentiality	Integrity	Availability
Hardware		tampered with	stolen, destroyed
Software	copied, pirated	Trojan horse, backdoor	deleted, license expire
Data	disclosure, inference	damaged	deleted, misplaced
People			quit, retire, vacation
Documentation	copied, stolen	tampered with	lost, stolen, destroyed
Supplies			lost, stolen, damaged

- Fill in the matrix

- Effects of unintentional errors: typos, insecure disposal of output
- Effects of malicious insiders: disgruntled employees, bribery, curious browsers (what is the salary of the CEO?)
- Effects of outsiders: network access, dial-in access, hackers, uninvited visitors, dumpster diving detectives
- Effects of natural disasters

Fault Tree Analysis

- Common method to determine vulnerabilities

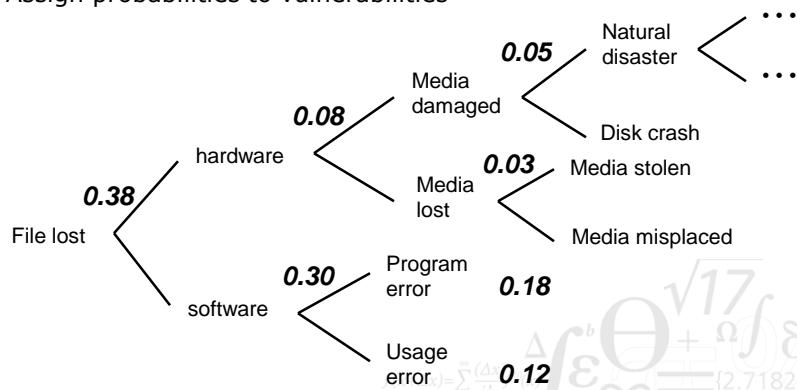


27 DTU Compute Technical University of Denmark

02239 – Data Security

Estimate Likelihood of Exploitation

- Assign probabilities to vulnerabilities



28 DTU Compute Technical University of Denmark

02239 – Data Security

Compute Expected Annual Loss

- Not all risks are equally severe
 - Safety critical: human lives are at stake
 - Mission critical: survival of the company is at stake
 - Costly: major impact on the earnings of the company
 - Negligible: costs are small compared with operational costs
- Knowing probability and cost of the event allows computation of the expected annual loss
- Example: Credit card companies accept a certain amount of fraud



29 DTU Compute Technical University of Denmark

02239 – Data Security

Measure All Costs

- Hidden costs are easy to forget:
 - Unavailability of the system, restore data from backups, reinstallation of software
- Consider the following questions:
 - What legal obligations exist for confidentiality and integrity of data
 - What contractual obligations may be applicable (*fines?*)
 - Could release of data harm individuals or organizations
 - *Will they litigate?*
 - Could event cause missed business in the future
 - What are the psychological effects of event
 - *Embarrassment, loss of credibility, loss of business, ...*
 - What is the value of access to data (worth a backup site)
 - What other problems could arise from loss of data
 - *Can data be restored*

30 DTU Compute Technical University of Denmark

02239 – Data Security

Management Frameworks and Governance Structures

- A number of standards and best practices are commonly used
- ISO 27000 series
 - Family of standards on Information Security Management Systems
 - *Available as "Dansk Standard" through the DTU Library*
- NIST Special Publication 800 series
 - NIST SP 800-39
 - NIST Cybersecurity Framework
 - *All publications available online at NIST*
- CIS Critical Security Controls
 - CIS defines a number of controls to improve security
 - Latest version is CIS v8 (earlier version is known as CIS20)
 - *Available at: <https://www.cisecurity.org/controls/v8/>*



02239 Data Security Software Security I

Sebastian Mödersheim

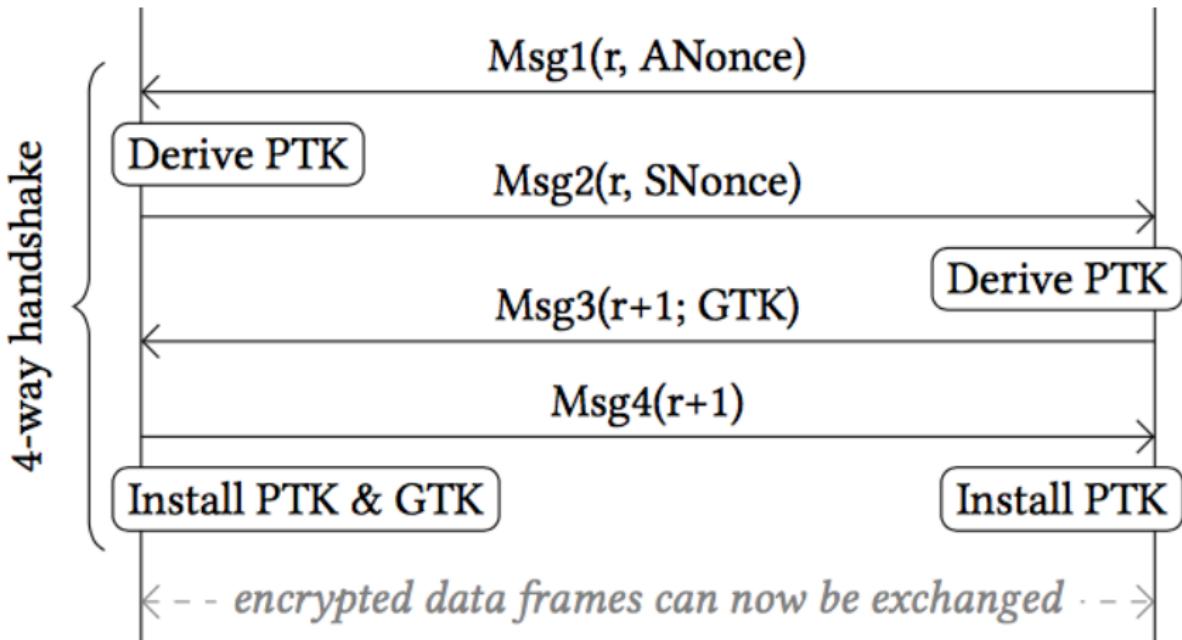
November 17, 2021

An Attack on WPA2

- M. Vanhoef and F. Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. (CCS 2017)
- On the border of protocol and software security:
 - ★ Cryptographic primitives have been analyzed
 - ★ Protocol has been analyzed
 - ★ Neither of them is in itself broken...

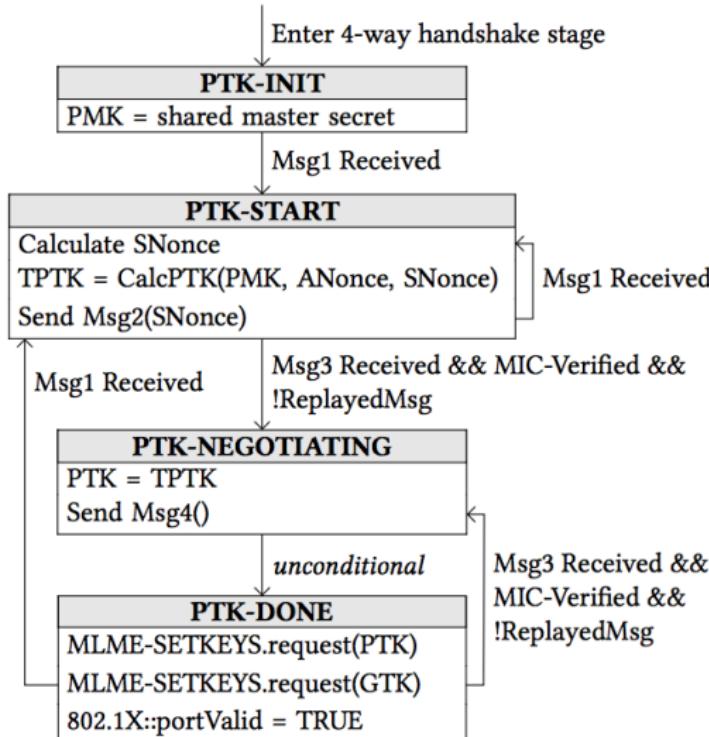
WPA2: 4-Way Handshake

Source: Vanhoef and Piessens 2017

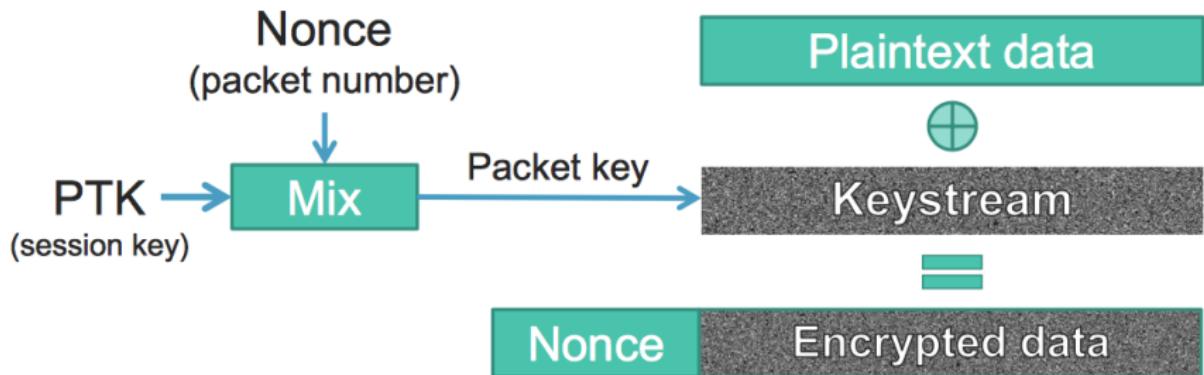


WPA2: A StateMachine

Source: Vanhoef and Piessens 2017



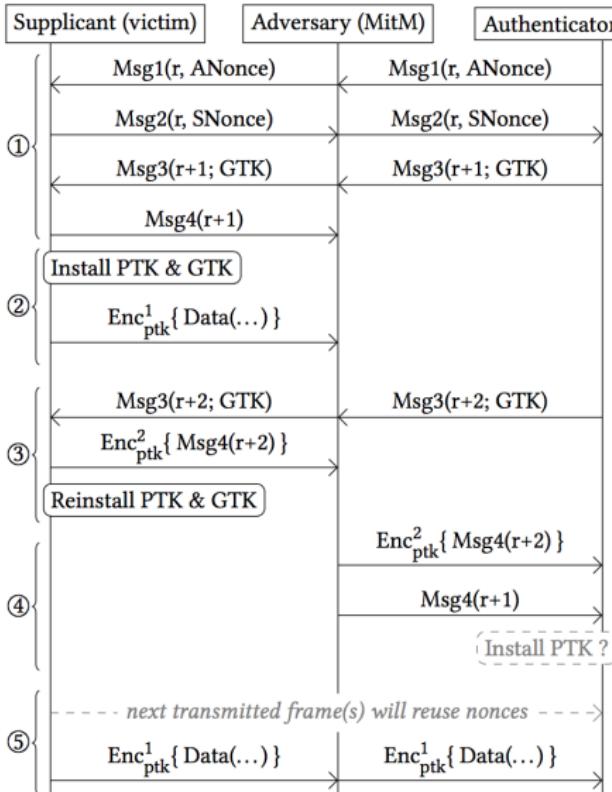
Frame encryption (simplified)



→ Nonce reuse implies keystream reuse (in all WPA2 ciphers)

WPA2: Simplemost Attack

Source: Vanhoef and Piessens 2017



Software Security Problems

Honest Mistakes

Programs may contain vulnerabilities that an attacker can exploit.

Mitigations:

- Developers: Prudent engineering practices, standards, libraries
- Analytic: Testing and verification
- Constructive: Secure by design and compilers

Trojans and the like

Executing code from a potentially malicious source:

- Installing new software, apps, plugins
- Web browser executing Javascript code
- Cloud hosting client applications

Typically offering a nice function for free, and including a hidden malicious function.

Buffer Overflow

What is a buffer overflow?

- Read/write beyond memory allocated to buffer
 - ★ Unchecked input
- May overwrite return addresses (e.g., stack overflow)
- May insert jumps to library code (e.g., heap overflow)

Beware

- “Basically a solved problem” (common belief in the 1990s)
- Some buffer overflows are easy to find
- Some are very hard to find/avoid

HOW THE HEARTBLEED BUG WORKS:



(Source: xkcd.com)

HMM...



BIRD



User Olivia likes random words pages about cars. Gees in car why". Note: Files for IP 375.381. 83.17 are in /tmp/files-3843. User Meg wants these 4 letters: **BIRD**. There are currently 34 connections open. User Brendan uploaded the file 15 minutes ago: 234b-062c-2acb04f6201-12-f69.

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "HAT" (500 LETTERS).



connection. Jake requested picture of user. User Meg wants these 500 letters: **HAT**. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long". User Karen wants to change account password to "CofBa2021".



BIRD. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CofBa2021".

(Source: [xkcd.com](https://xkcd.com/128))

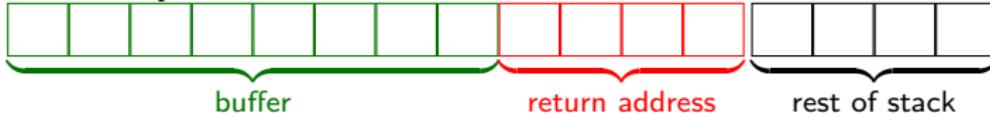
Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```

Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```

Stack Layout:



Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```

Stack Layout:



Attack: give an argument of 12 chars XXXXXXXXXXXXAAAA

Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```

Stack Layout:



Attack: give an argument of 12 chars XXXXXXXXAAAA

If you can guess address AAAA of buffer, you can execute code XXXXXXXX under the privilege of this application.

Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```



Gives the attacker a shell with the privilege of this application.
This type of buffer overflow is called **code injection**.

Example: Stack Overflow

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}
```



The code will continue at **address** (intruder's choice).
This type of buffer overflow is called **code reuse**.

But that is difficult to guess?

- The attacker actually needs to guess addresses.
- Actually, mounting a buffer overflow often requires a lot of trial and error for an attacker.
- But suppose that
 - ★ the attacker knows the programs source and binary code,
 - ★ can try the attack any number of times,
 - ★ he can write a script that automatically tries the attack with different input values...
- Techniques like stack obfuscation (reordering, encryptions of values, ...) and canaries (special value before the return address) are no ultimate solution, but make the life of the attacker harder.

Unsafe Functions

- Strings in C are `char` arrays of **arbitrary** length: they are terminated by a 0-byte.
- Several functions in the C library work on strings **without stopping** before reading a 0-byte.
- When copying into a buffer, then a too long string means overwriting memory areas following the buffer.
 - ★ If you are lucky, the program just crashes.
 - ★ If you are unlucky, the attacker succeeds without anybody noticing.
- Avoid `strcpy`, `gets`, `printf`, ... on unknown inputs
- Alternatives like `strncpy` and `fgets` are not magically making it safe. Always mind the size of buffers
 - ★ including space for the trailing 0 byte!!

Overflows

```
char *array;  
int size;  
...  
if (size>1024) size=1024;  
array = malloc(size);  
read(file,array,size);
```

Overflows

```
char *array;  
int size;  
...  
if (size>1024) size=1024;  
array = malloc(size);  
read(file,array,size);
```

What if `size<0`?

Overflows

```
char *array;  
int size;  
...  
if (size<0) size=0;  
if (size>1024) size=1024;  
array = malloc(size);  
read(file,array,size)
```

Why not check this automatically?

- Many languages like Java, JavaScript, ... do check bounds of arrays, overflows and null-pointers.

Why not check this automatically?

- Many languages like Java, JavaScript, ... do check bounds of arrays, overflows and null-pointers.
- “C/C++ does not do this for efficiency.”

Why not check this automatically?

- Many languages like Java, JavaScript, ... do check bounds of arrays, overflows and null-pointers.
- “C/C++ does not do this for efficiency.”
- Idea of static analysis: a compiler that tries to prove **statically** that a piece of code will never get out of bounds, and insert the check only when it fails to prove that.
 - ★ similar to type checking.

Why not check this automatically?

- Many languages like Java, JavaScript, ... do check bounds of arrays, overflows and null-pointers.
- “C/C++ does not do this for efficiency.”
- Idea of static analysis: a compiler that tries to prove **statically** that a piece of code will never get out of bounds, and insert the check only when it fails to prove that.
 - ★ similar to type checking.
- The concept of C/C++ does not allow for that in general!
 - ★ Arrays are just pointers without an implicit length information.
Allows freedom, but invites for mistakes.
 - ★ Using classes with checks from C++ like String, Vector, Array, can prevent much damage.

How to Prevent Bad Programming

Some Partial Solutions

- Teach programmers to be more careful
- Security mechanisms in languages, libraries, platforms
 - ★ Java bytecode verification, type systems, sandboxing
- More and better testing

Formal Methods

Techniques

- Model Checking
- Static Analysis
- Information Flow Analysis
- Certifying Compilers
- Monitors

Static Analysis

- Roots: optimizing compilers
- Static computation of dynamic behavior
 - ★ Approximation used to sidestep halting-problem

Splint: Light-Weight Annotation-Based Static Analysis

Secure Programming LINT

- Based on lint: well-known program checker
- Let the programmer annotate program
- Check that the program is consistent with annotations
- Can find many common errors

Other Tools

- VCC/Z3 (Microsoft)
- BLAST
- ...

Splint: Buffer Overflow

Example

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
}

$ splint +bounds buffer01.c
...
buffer01:9: Possible out-of-bounds store: strcpy(str,tmp)...
```

Splint: Buffer Overflow

Example

```
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[8];
    strcpy(buffer, argv[1], 7);
    buffer[7] = '\\0';
}
```

```
$ splint +bounds buffer01.c
```

```
...
```

```
Finished checking --- no warnings
```

Splint

Problems

- May need a lot of annotation/time/experience
- False positives: warnings often do not represent real problems
- False negatives: real problems may be missed
 - ★ Memory modeling
 - ★ Sharing
 - ★ Control Flow

Malicious Code

Some types of malicious code (incomplete, not disjoint):

Virus/Worm replicates, trying to infect more systems

Trojan horse hides malicious code behind a desired, primary effect

Root kit manipulates system routines to prevent detection

Malicious Code

Some types of malicious code (incomplete, not disjoint):

Virus/Worm replicates, trying to infect more systems

Trojan horse hides malicious code behind a desired, primary effect

Root kit manipulates system routines to prevent detection

- RootkitRevealer [Russinovich 2006f] compares output of system routines and physical data on hard disk
- Revealed the famous Sony root kit

Getting the malicious code executed

Malicious software can only harm when executed.

- Email: “Please install the attached software update for urgent security problems.”
- “Download this cool free protocol analysis tool today!”
- Apps for mobile phones
- Most websites have executable code like JavaScript.
 - ★ You can turn off JavaScript, but then many websites become unusable.
- Many documents (MS Word, PDF, ...) may contain scripts/executable code.
- Autoplay (do you know what your system executes automatically?)
- Exploits: intruder inserts code into system e.g. exploiting a buffer overflow.

Making Yourself Comfortable

Once malicious code has gained control of a computer, how can it protect against detection and removal?

- Inject copy of the code into normal applications or the boot sector of hard disk
 - ★ Ensure that it gets executed again after a restart
- Manipulate system routines so that virus scanners or user will not detect it.
- Obfuscation by encrypting/changing part of the code (avoiding a “fingerprint”).
- Become part of a botnet!

Some Famous Examples

The Brain Virus (1986)

- Floppy disc bootsector virus

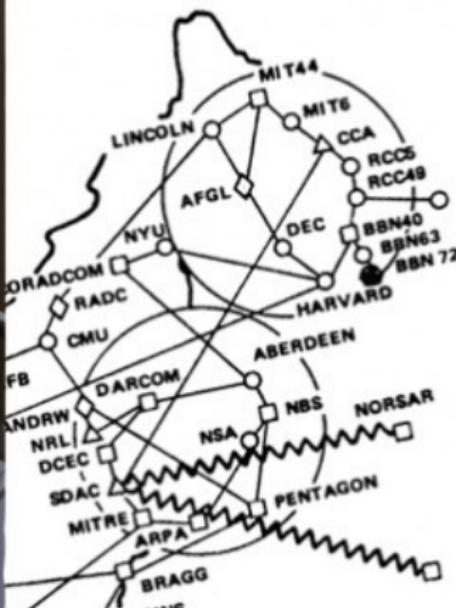
A screenshot of the ht 2.0.16 hex editor. The title bar says "ht 2.0.16". The menu bar includes File, Edit, Windows, Help, Local-Hex, and a timestamp "14:17 07.01.2010". The main window displays a hex dump of the "brain_sector" file. The code starts with a series of zeros and then contains ASCII text. The text includes "Welcome to the Dungeon", copyright information for 1986 Basit & Amjad, and details about the virus's origin in Lahore, Pakistan. It also contains a warning message and a call for vaccination.

```
[...]= ...ples\brain_sector\8de894dc6f27e10664fc7db1137ef3ef0af62d5.bin ==2
00000000 fa e9 4a 01 34 12 01 08-06 00 01 00 00 00 00 20 0J04t@ 0
00000010 20 20 20 20 20 57 65-6c 63 6f 6d 65 20 74 6f
00000020 20 74 68 65 20 44 75 6e-67 65 6f 6e 20 20 20 20
00000030 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20 20
00000040 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20 20
00000050 20 28 63 29 20 31 39 38-36 20 42 61 73 69 74 20
00000060 26 20 41 6d 6a 61 64 20-28 70 76 74 29 20 4c 74
00000070 64 2e 20 20 20 20 20 20-20 20 20 20 20 20 20 20
00000080 20 42 52 41 49 4e 20 43-4f 4d 50 55 54 45 52 20
00000090 53 45 52 56 49 43 45 53-2e 2e 37 33 30 20 4e 49
000000a0 5a 41 4d 20 42 4c 4f 43-4b 20 41 4c 4c 41 4d 41
000000b0 20 49 51 42 41 4c 20 54-4f 57 4e 20 20 20 20 20
000000c0 20 20 20 20 20 20 20 20-20 20 20 4c 41 48 4f 52
000000d0 45 2d 50 41 4b 49 53 54-41 4e 2e 2e 50 48 4f 4e
000000e0 45 20 3a 34 33 30 37 39-31 2c 34 34 33 32 34 38
000000f0 2c 32 38 30 35 33 30 2e-20 20 20 20 20 20 20 20
00000100 20 20 42 65 77 61 72 65-20 6f 66 20 74 68 69 73
00000110 20 56 49 52 55 53 2e 2e-2e 2e 2e 43 6f 6e 74 61
00000120 63 74 20 75 73 20 66 6f-72 20 76 61 63 63 69 6e
00000130 61 74 69 6f 6e 2e 2e 2e-2e 2e 2e 2e 2e 2e 2e 2e
00000140 2e 2e 2e 2e 20 24 23 40-25 24 40 21 21 20 8c c8
view e0h/224
1help 2save 3open 4edit 5goto 6mode 7search 8resize 9viewin.0quit
```

Some Famous Examples

The Internet Worm (1988)

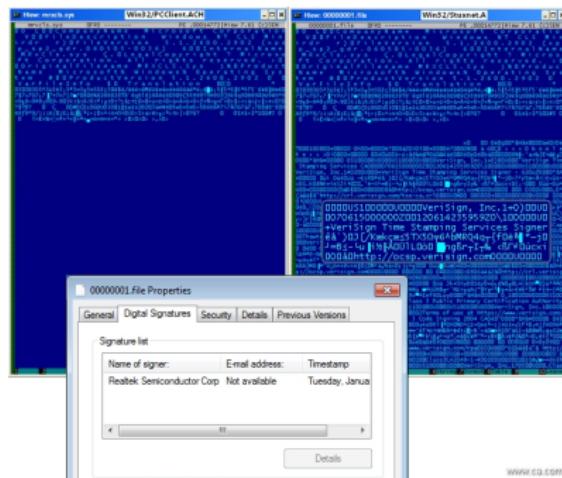
- Meant to just determine how far it can spread;
- By **honest mistake** (!) became a denial-of-service attack.



Some Famous Examples

Stuxnet (2010)

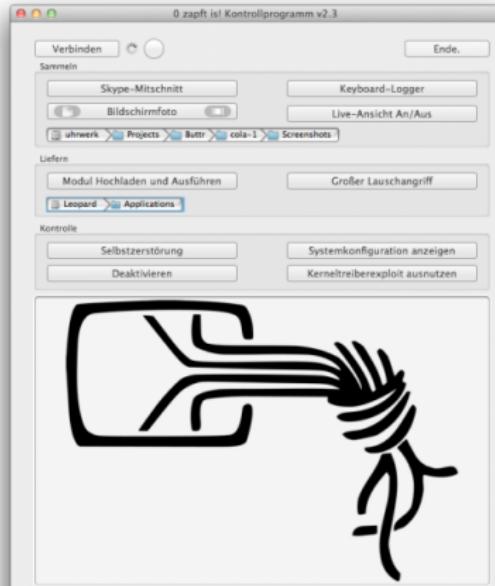
- Targeted at particular industrial software
- 4 zero-day exploits
- signed using stolen certificates
- Speculations that it was written by a secret service



Some Famous Examples

“Staatstrojaner” (2008/2011)

- Chaos Computer Club claims that it is malware used by law enforcement in several German states for Internet-phone surveillance of suspects (after warrant from a judge).



Some Famous Examples

WannaCry (2017)

- Ransomware: encrypt all hard disk and require ransom (in bitcoin) for decryption.
- Used [EternalBlue](#), a Windows exploit first discovered by NSA.
 - ★ NSA had not notified Microsoft about the vulnerability.
 - ★ Apparently this exploit was stolen and leaked.
- US government blamed North Korea.



The Lazy Mobile Intruder

How to verify sandboxing?

- Security of a platform that executes potentially malicious code
- Goal: the code cannot break out of its sandbox
- Research Paper:
Sebastian Mödersheim, Flemming Nielson, Hanne Riis Nielson.
Lazy Mobile Intruders. POST 2013.

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

Question: does a program P exist such that

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

Question: does a program P exist such that

- $K_0 \vdash P$ the intruder can write P

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

Question: does a program P exist such that

- $K_0 \vdash P$ the intruder can write P
- $C[P] \rightarrow^* S$ when $C[\cdot]$ executes P we reach a state S ...

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $attack(S)$

Question: does a program P exist such that

- $K_0 \vdash P$ the intruder can write P
- $C[P] \rightarrow^* S$ when $C[\cdot]$ executes P we reach a state S ...
- $attack(S)$... that is an attack

?

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

Question: does a program P exist such that

- $K_0 \vdash P$ the intruder can write P
- $C[P] \rightarrow^* S$ when $C[\cdot]$ executes P we reach a state S ...
- $\text{attack}(S)$... that is an attack

?

Obviously we cannot search the space of all programs $\{P \mid K_0 \vdash P\}$.

The Lazy Mobile Intruder

The Problem

Given:

- Platform $C[\cdot]$ that executes potentially malicious code
 - ★ E.g. web browser, mobile phone, cloud provider
- Initial intruder knowledge K_0
- Attack predicate $\text{attack}(S)$

Question: does a program P exist such that

- $K_0 \vdash P$ the intruder can write P
- $C[P] \rightarrow^* S$ when $C[\cdot]$ executes P we reach a state $S\dots$
- $\text{attack}(S) \dots$ that is an attack

?

Obviously we cannot search the space of all programs $\{P \mid K_0 \vdash P\}$.

Uses the **Lazy Intruder** technique from OFMC

Rules for Symbolic Intruder Processes

Example: Communication rule

$$\underbrace{(x).P}_{p1} \parallel \underbrace{\langle M \rangle}_{p2} \rightarrow P[x \mapsto M]$$

Rules for Symbolic Intruder Processes

Example: Communication rule

$$\underbrace{(x).P}_{p1} \parallel \underbrace{\langle M \rangle}_{p2} \rightarrow P[x \mapsto M]$$

intruder receiving

$$[K] \parallel \langle M \rangle \Rightarrow [K \cup \{M\}]$$

Rules for Symbolic Intruder Processes

Example: Communication rule

$$\underbrace{(x).P}_{p1} \parallel \underbrace{\langle M \rangle}_{p2} \rightarrow P[x \mapsto M]$$

intruder receiving

$$\boxed{K} \parallel \langle M \rangle \Rightarrow \boxed{K \cup \{M\}}$$

intruder sending

$$(x).P \parallel \boxed{K} \Rightarrow P \parallel \boxed{K} \text{ and } \psi = K \vdash x$$

Rules for Symbolic Intruder Processes

Example: Communication rule

$$\underbrace{(x).P \parallel \langle M \rangle}_{p1} \rightarrow \underbrace{P[x \mapsto M]}_{p2}$$

intruder receiving

$$\boxed{K} \parallel \langle M \rangle \Rightarrow \boxed{K \cup \{M\}}$$

intruder sending

$$(x).P \parallel \boxed{K} \Rightarrow P \parallel \boxed{K} \text{ and } \psi = K \vdash x$$

intruder both sending and receiving

$$\boxed{K} \parallel \boxed{K'} \Rightarrow \boxed{K \cup K'}$$

02239 Data Security Software Security II

Sebastian Mödersheim

November 24, 2021

This Week

- Most common software vulnerabilities
- Information flow
- Side channel attacks

Top 25 Errors

MITRE/SANS Common Weakness Enumeration

<http://cwe.mitre.org/top25>

Collection of the most common security vulnerabilities of software:

- Based on reported security problems.
- Particularly dangerous flaws because **low-hanging fruit** for attackers.
- Software developers shall be made aware of the problems.
- Provide “standardized” ways to avoid the problems.
- Updated regularly to reflect increasing/decreasing significance of problems.
- Also contains specialized recommendations depending on programming language used, preferences, and educational emphasis.
- **Should one publish such a list?**

SQL Injection

Webpage

login:

→ \$user

password:

→ \$pass

PHP script generating an SQL query

```
$SQLquery = "SELECT * FROM customers  
WHERE username = '$user' AND password='$pass';"
```

Given a correct username-password pair, this retrieves all information about this user from the database customers. This can be put into a nice webpage by a script again.

SQL Injection

Webpage

login: → \$user
password: → \$pass

Generated SQL query

```
$SQLquery = "SELECT * FROM customers  
WHERE username = '' OR 1; ---' AND password=''"
```

which returns the entire customer database.

SQL Injection

Webpage

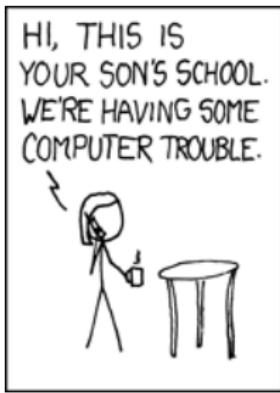
login: → \$user
password: → \$pass

Generated SQL query

```
$SQLquery = "SELECT * FROM customers  
WHERE username = ''; DELETE * FROM customers; ---'..."
```

which deletes the entire customer database.

SQL Injection: Exploits of a Mom



Source: xkcd.com/327

SQL Injection: Mitigation

- Re-iterate: all input to your programs could be malicious
- Input needs to be sanitized.
- In this case: ensure there is no ' in the input, but is this everything?
- Enforce separation between data and code; strong typing.
- Use different solutions for access-control than SQL databases.
- Try to isolate: separate databases, introduce special account with low privileges

XSS – Cross Site-Scripting

- Web applications **generate webpages**, based on input from users.
- That input may contain **malicious code** e.g. in JavaScript.
- Without proper checking the input, the generated webpage contains the malicious code.
- Code is thus executed under the **domain** of the webpage.

Cross Site-Scripting: Example

Intruder's webpage or email

You can find everything you need [here](#).

Cross Site-Scripting: Example

Intruder's webpage or email

You can find everything you need [here](#).

where [here](#) links to

www.dtu.dk/someNonExistingPage<script type="text/javascript">...

Cross Site-Scripting: Example

Intruder's webpage or email

You can find everything you need [here](#).

where [here](#) links to

[www.dtu.dk/someNonExistingPage<script type="text/javascript">...](http://www.dtu.dk/someNonExistingPage<script type='text/javascript'>...)

[www.dtu.dk](#) might reply

404 Page not found: The page

[someNonExistingPage<script type="text/javascript">...](http://someNonExistingPage<script type='text/javascript'>...)

was not found on this server.

Cross Site-Scripting: Example

Intruder's webpage or email

You can find everything you need [here](#).

where [here](#) links to

[www.dtu.dk/someNonExistingPage<script type="text/javascript">...](http://www.dtu.dk/someNonExistingPage<script type='text/javascript'>...)

[www.dtu.dk](#) might reply

404 Page not found: The page

[someNonExistingPage<script type="text/javascript">...](http://someNonExistingPage<script type='text/javascript'>...)

was not found on this server.

And now there is a piece of **malicious code** that comes from DTU!

Cross Site-Scripting: Same Origin Policy

Same Origin Policy

- JavaScript on one webpage may access the data and methods of other webpages **on the same domain**.
- Most importantly cookies: often contain session and authentication data.
- It is essential that cookies can be accessed only by the domain to which they belong.

It thus makes a difference whether the malicious code is on the intruder's webpage or on DTU's webpage.

Cross Site-Scripting: Same Origin Policy

Same Origin Policy

- JavaScript on one webpage may access the data and methods of other webpages **on the same domain**.
- Most importantly cookies: often contain session and authentication data.
- It is essential that cookies can be accessed only by the domain to which they belong.

It thus makes a difference whether the malicious code is on the intruder's webpage or on DTU's webpage.

But sorry hackers: DTU pages are already secured against this particular attack.

Cross Site-Scripting: Mitigation

- General rule: **all input to your programs could be malicious**
- Specify: what is acceptable input for your case?
- Refuse/filter all input that does not follow this specification.
- Not trivial: anticipate everything that may could cause trouble.
- Generally, producing HTML/webpages: filter user input for everything that is not pure text.
 - ★ ... but what is pure text? (character encoding!)
 - ★ understand context in which data will be used, what encodings are present
- Understand all areas where untrusted input can enter your software
- Using special libraries and also firewalls can help

Cross-Site Request Forgery

- Any social media websites allow users to submit/post content in some form.
- Usually they will allow posting images by giving a URL. How's about:

```

```

- Every user who reads the post will send the request `execute?action=...` to `mywebemail.dk`.
- Suppose `mywebemail.dk` has authentication based on cookies.
- Suppose some reader is also legal user of `mywebmail.dk`.
- Suppose this user is currently logged into `mywebmail.dk`.
- ...

CSRF + XSS

Samy worm combined two of the vulnerabilities:

- Samy Kamkar performed a XSS attack by introducing some Javascript into his Myspace page.
- As a payload of the XSS, he had a CSRF attack. The victim would involuntarily
 - ★ call the **add-as-friend** function of Myspace for Samy
 - ★ insert **Samy is my hero** into the victims profile
 - ★ along with a copy of the worm (the Javascript code)
- Over a million infections within 24h.

CSRF + XSS

Samy worm combined two of the vulnerabilities:

- Samy Kamkar performed a XSS attack by introducing some Javascript into his Myspace page.
- As a payload of the XSS, he had a CSRF attack. The victim would involuntarily
 - ★ call the **add-as-friend** function of Myspace for Samy
 - ★ insert **Samy is my hero** into the victims profile
 - ★ along with a copy of the worm (the Javascript code)
- Over a million infections within 24h.
- Stupid joke (and he got probation), but shows the potential.

CSRF: mitigation

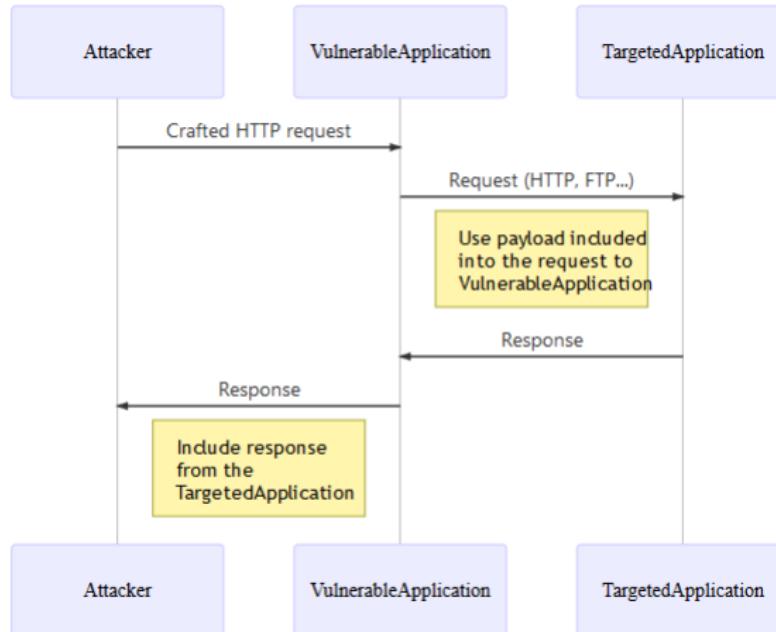
- First ensure that XSS is prevented, because it can bypass CSRF protections.
- Introduce challenge-response on requests using a server-generated nonce.
- Asking back for dangerous operations
- Check http referrer tag.

SSRF: Server Side Request Forgery

Scenario

- A web application in a user's browser sends requests to a server-side API
 - ★ e.g. apiCall=https://kurser.dtu.dk/course/02244 to get a course description
- The request causes the server API to access a specified resource (e.g. said url) under permissions of web server
- An attacker observes this behavior and formulates malicious requests (imitating the web application)
 - ★ e.g. apiCall=https://karakterindberetning.dtu.dk/...
-

SSRF: Server Side Request Forgery



Source: <https://owasp.org>

OWASP Top Ten 2021



-
- 1 Broken Access Control
 - 2 Cryptographic Failures
 - 3 Injection
 - 4 Insecure Design
 - 5 Security Misconfiguration
 - 6 Vulnerable and Outdated Components
 - 7 Identification and Authentication Failures
 - 8 Software and Data Integrity Failures
 - 9 Security Logging and Monitoring Failures
 - 10 Server-Side Request Forgery
-

<https://owasp.org/www-project-top-ten/>

Monster Mitigations

Effective strategies and principles to address a large number of errors:

- Establish and maintain control over all of your inputs.
- Establish and maintain control over all of your outputs.
- Lock down your environment.
- Assume that external components can be subverted, and your code can be read by anyone.
- Use industry-accepted security features instead of inventing your own.
- Use libraries and frameworks that make it easier to avoid introducing weaknesses.
- Integrate security into the entire software development lifecycle.
- Use a broad mix of methods to comprehensively find and prevent weaknesses.
- Allow locked-down clients to interact with your software.

Lessons Learned?

- What can we learn from this?
 - ★ Security is an extremely subtle and complicated topic
 - ★ Actually the same mistakes and attacks appear again and again with slight variations
 - ★ The fact that even basic security properties can be broken with relatively simple attacks is scary.
- Good engineering practices:
 - ★ The Top25 list has good suggestions for avoiding top problems.
 - ★ Better education and training can help
 - ★ Development processes that better support security

Scientific Point of View

Engineering practices to avoid some mistakes are helpful but scientifically unsatisfactory!

- Think of virus scanners that can just check for well-known patterns.
- Attackers become aware of how defenses work and find new holes.
- Our goal is thus to develop:
 - ★ More systematic/general solutions
 - ▶ Systems that by construction prevent entire class of attacks
 - ★ Mathematical proofs that systems/constructions are correct
 - ▶ ... against **any** attack, not just the known ones
 - ★ This requires mathematical models and security definitions that can capture all relevant aspects of reality

Information Flow: Example

```
classified int b=3;  
int y=12;  
int result=1;  
while (y>0){  
    result=result*b;  
    y=y-1;  
}  
output(result);
```

- Some variables may be considered to hold **classified** information
- Then they should not be output ever!
- Actually nothing should be output that can give an attacker **any clue** on the classified information!

Information Flow: Example

```
classified int b=3;  
int y=12;  
int result=1;  
while (y>0){  
    result=result*b;  
    y=y-1;  
}  
output(result);
```

- Some variables may be considered to hold **classified** information
- Then they should not be output ever!
- Actually nothing should be output that can give an attacker **any clue** on the classified information!
- Violated in this example: a term that depends on **b** is written into the unclassified variable **result** and that is output.
This is called an **explicit flow**.

Information Flow: Example

```
classified int b=3;  
int y=12;  
int result=1;  
while (y>0){  
    result=result*b;  
    y=y-1;  
}  
output(result);
```

- Some variables may be considered to hold **classified** information
- Then they should not be output ever!
- Actually nothing should be output that can give an attacker **any clue** on the classified information!
- Violated in this example: a term that depends on **b** is written into the unclassified variable **result** and that is output.
This is called an **explicit flow**.
- Rationale: whoever can learn a value, can potentially see all information that has “flown to it”.

Information Flow: Example

```
int b=3;  
classified int y=12;  
int result=1;  
while (y>0){  
    result=result*b;  
    y=y-1;  
}  
output(result);
```

- What about this one?

Information Flow: Example

```
int b=3;
classified int y=12;
int result=1;
while (y>0){
    result=result*b;
    y=y-1;
}
output(result);
```

- What about this one?
- Actually there is no assignment from classified variables to unclassified ones, so this is ok?

Information Flow: Example

```
int b=3;  
classified int y=12;  
int result=1;  
while (y>0){  
    result=result*b;  
    y=y-1;  
}  
output(result);
```

- What about this one?
- Actually there is no assignment from classified variables to unclassified ones, so this is ok?
- No, the value of the unclassified `result` may depend on the condition $y > 0$. This is called an **implicit flow**.

Application: Information Flow

- High security guarantee: the code does not leak any information about the classified variables to an attacker who can only observe the unclassified variables.

Application: Information Flow

- High security guarantee: the code does not leak any information about the classified variables to an attacker who can only observe the unclassified variables.
- Except, actually there may be side channels, e.g., how much time an algorithm needs may leak information about the data involved.

Static Information Flow Analysis

A type-checking-like analysis:

- We have two security classes `public < confidential`
 - ★ In general we have a lattice of security classes
- **Explicit flow:** For every assignment $x=e$ we check
 - ★ that security class of x is at least as high as that of e .
 - ★ The security class of e is the highest of any variable in e (and `public` if there are no variables in e).
- **Implicit flow:** For every `while e { P }` we check
 - ★ that for every assignment $x=e$ in P , the level of x is at least as high as e .
 - ★ Similar checks for `if e { P } else { Q }`.

Some similar analysis can be done for checking **integrity** of variables.

Static Analysis

- With the type-checking we have again static analysis
 - ★ 02244 Logic for Security
- Types can sometimes be automatically inferred.
- Well-studied problems, so a lot of tool support.
- Over-approximation/false positives:
 - ★ an ill-typed program may sometimes be fine. We just cannot prove it with types.
 - ★ Compromise to sidetrack the halting problem.
 - ★ Soundness: when type-checked you can be sure that the information flow is fine.

Side Channels

Computations can have **side effects**:

- Power consumption
- Timing (computation, memory access vs. cache)
- Noise (audio, electro-magnetic,...)

Side Channel Analysis

- An intruder may be able to **observe** these side effects
- ... and obtain information about the computation itself.
- Typically this was **not intended** by its designers.

Spectre



Spectre is a novel side-channel attack:¹

- Exploit combines two common performance optimization of modern processors.
 - ★ Caching
 - ★ Speculative execution.
- Essentially: using speculative execution to leave “traces” in the cache and obtain these traces.
- Works on a wide variety of processors.

¹Paul Kocher et al.: *Spectre Attacks: Exploiting Speculative Execution*, Security and Privacy 2019. See also spectreattack.com

Speculative Execution

```
a=3;  
b=Array [6];  
c=f(a);
```

- Accessing Array[6] may be slow if not in cache.
- Instead of waiting, we could start computing f(a).

Speculative Execution

```
a=3;  
if ( Array[6]>3)  
    c=f(a);  
else  
    c=g(a);
```

- If `Array[6]` takes time, we do not know for a while what is the next command.

Speculative Execution

```
a=3;  
if ( Array[6]>3)  
    c=f(a);  
else  
    c=g(a);
```

- If `Array[6]` takes time, we do not know for a while what is the next command.
- Branch prediction: if we know `Array[6]>3` has been often true in the past, we could **speculate** it is going to be true again.

Speculative Execution

```
a=3;  
if ( Array[6]>3 )  
    c=f(a);  
else  
    c=g(a);
```

- If `Array[6]` takes time, we do not know for a while what is the next command.
- Branch prediction: if we know `Array[6]>3` has been often true in the past, we could **speculate** it is going to be true again.
 - ★ Go ahead computing `f(a)`
 - ★ If `Array[6]>3` turns out to be true, we win.
 - ★ Otherwise, we have to **revert** the processor,
but we did not lose anything.

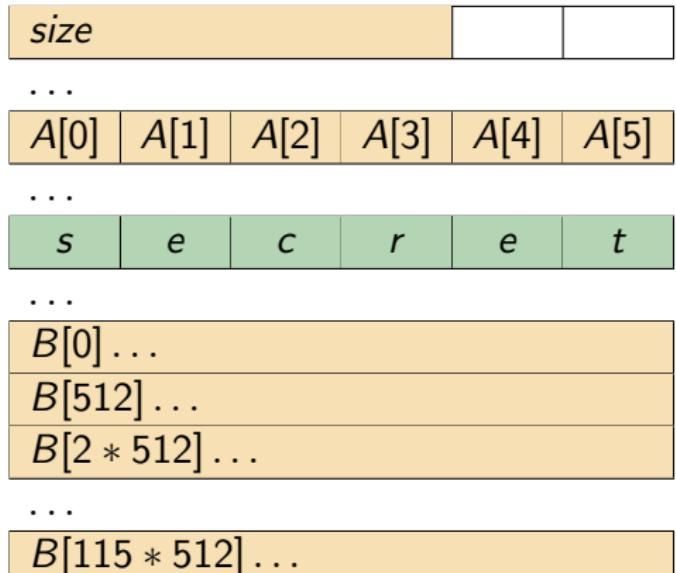
Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Dramatis personae:

- y : chosen by the intruder.
- A : array of size size .
- The intruder is allowed to know arrays A and B .
- There is a *secret* in the application memory somewhere after A .



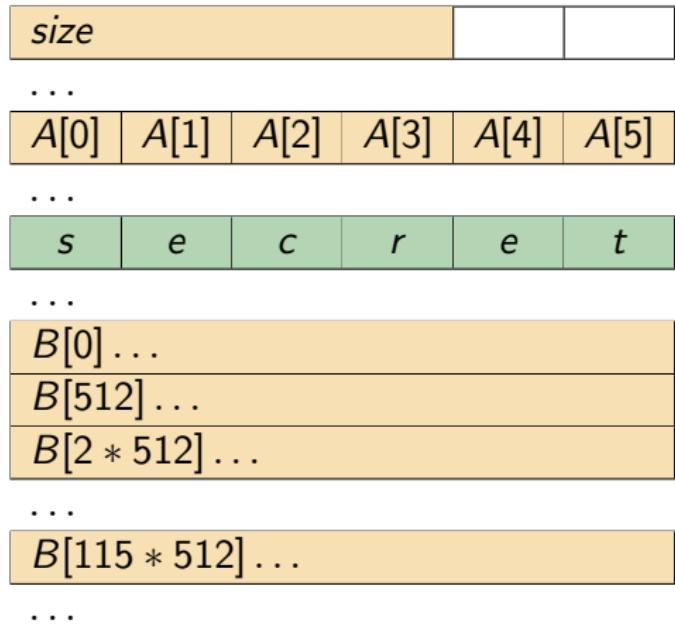
Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Suppose

- size is **not in** the cache.
- A is **not in** the cache.
- B is **not in** the cache.
- *secret* is **in** the cache.
- Branch prediction expects `y < size` to be **true**.

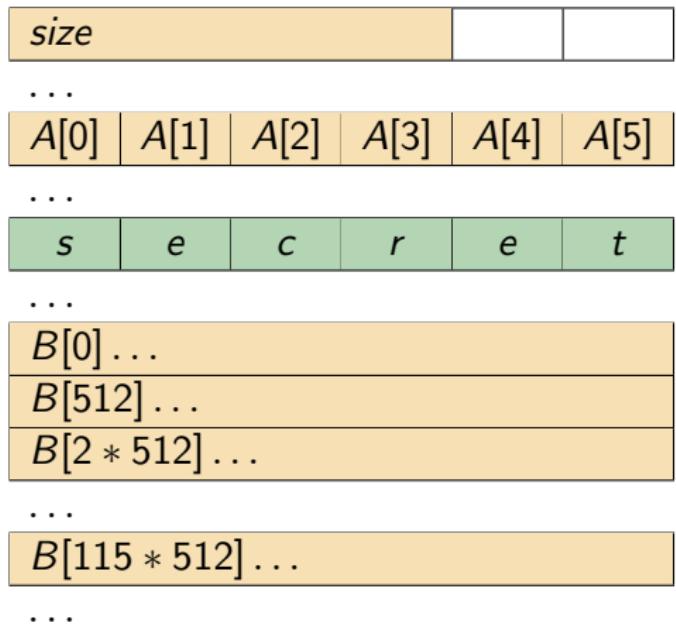


Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

- The intruder guesses y , so that $A + y$ is the address of **secret**.
 - ★ So: $y \geq size$!

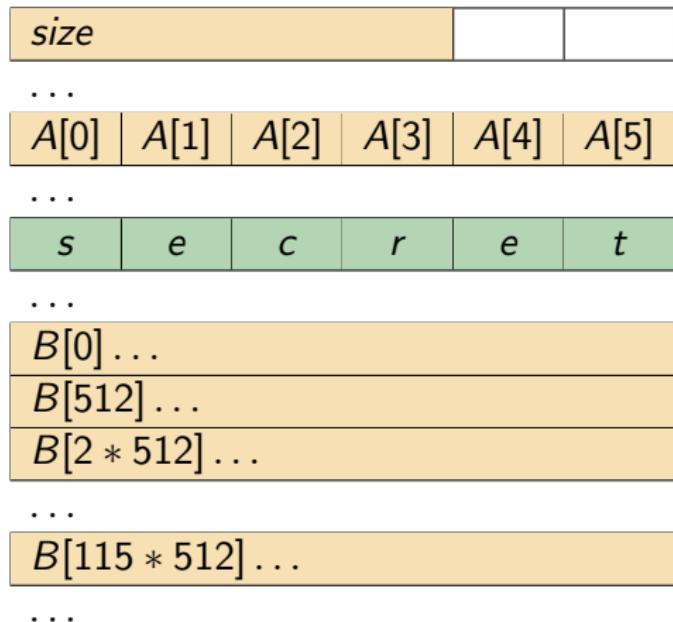


Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

- The intruder guesses y , so that $A + y$ is the address of **secret**.
 - ★ So: $y \geq size$!
- Evaluate $y < size$:
 - ★ $size$ not cached,
 - ★ branch prediction is positive

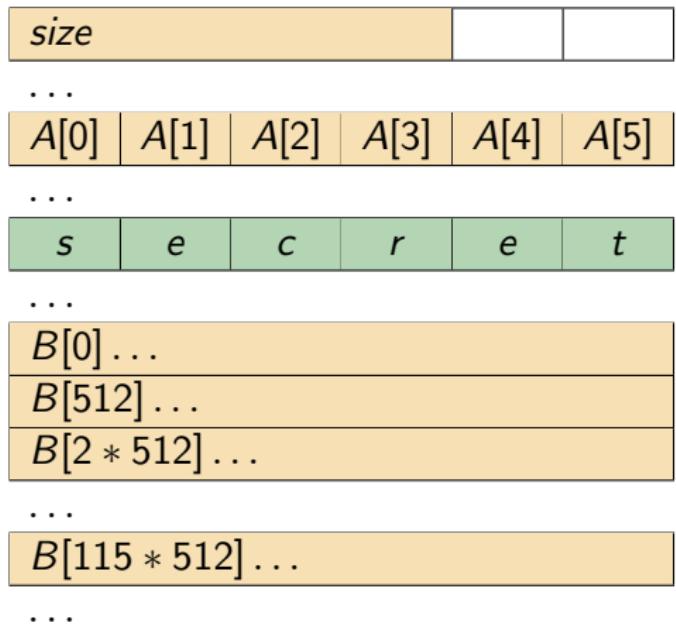


Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

- The intruder guesses y , so that $A + y$ is the address of **secret**.
 - ★ So: $y \geq size$!
- Evaluate $y < size$:
 - ★ $size$ not cached,
 - ★ branch prediction is positive



Thus start **speculative execution** of $B[A[y]*512]$.

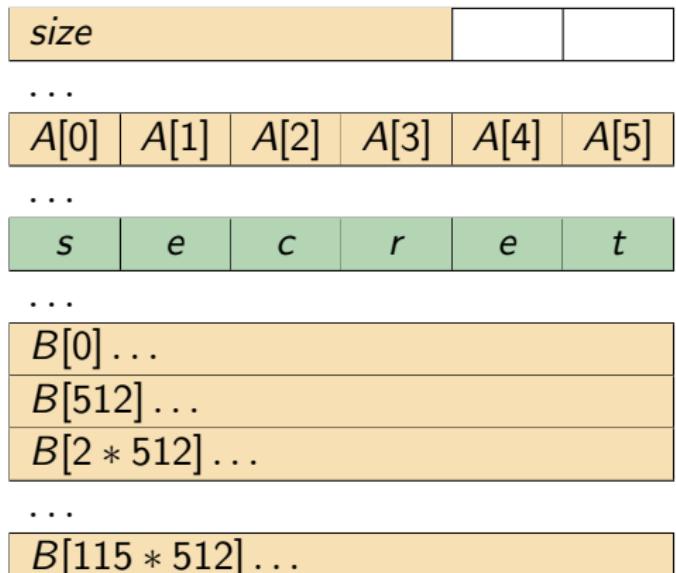
Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Speculative execution of
 $B[A[y]*512]$

- $A + y$ is address of secret.



Spectre: Example

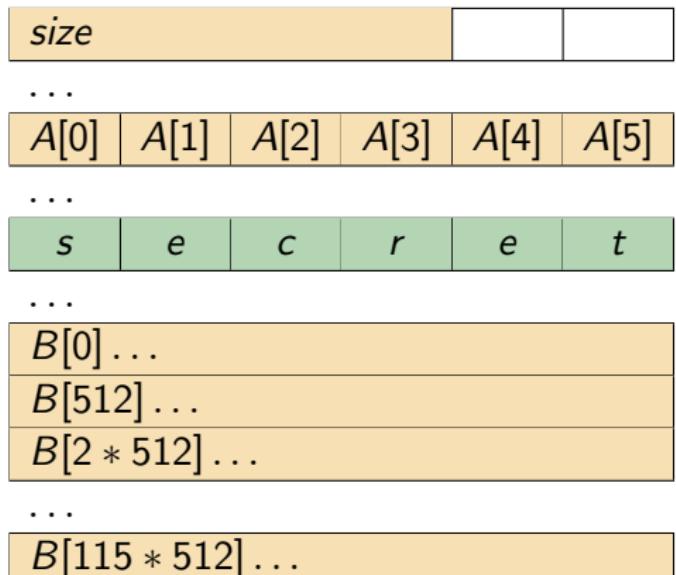
Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Speculative execution of

$B[A[y]*512]$

- $A + y$ is address of **secret**.
- So $A[y]$ is in the cache!



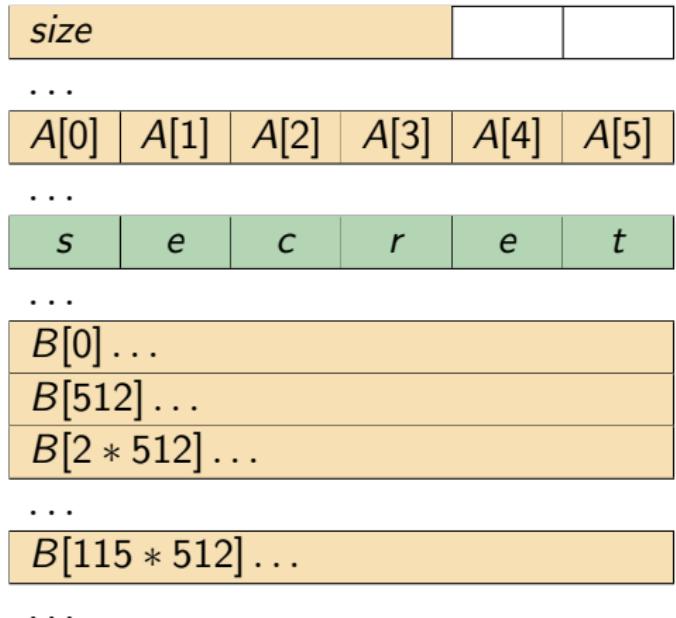
Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Speculative execution of
 $B[A[y]*512]$

- $A + y$ is address of **secret**.
- So $A[y]$ is in the cache!
- So processor loads
 $B[A[y] * 512] =$
 $B[115 * 512]$



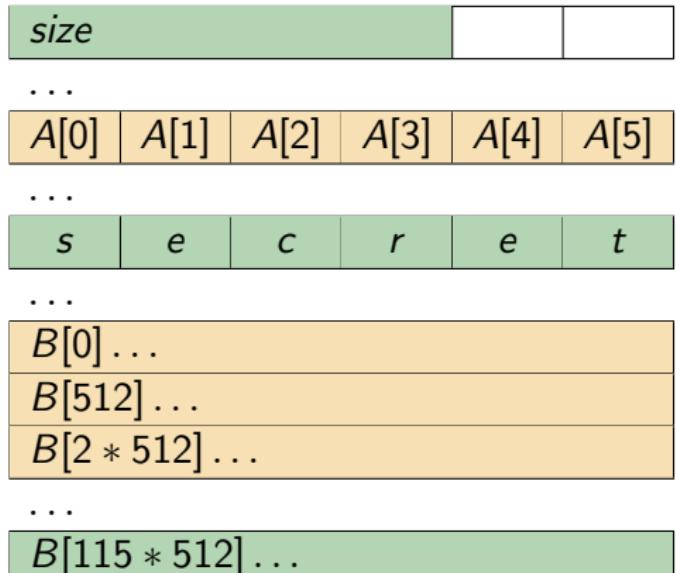
Spectre: Example

Victim code

```
if (y < size)
    temp &= B[A[y]*512]
```

Speculative execution of
 $B[A[y]*512]$

- $A + y$ is address of **secret**.
- So $A[y]$ is in the cache!
- So processor loads
 $B[A[y] * 512] =$
 $B[115 * 512]$
- ... it is now cached.



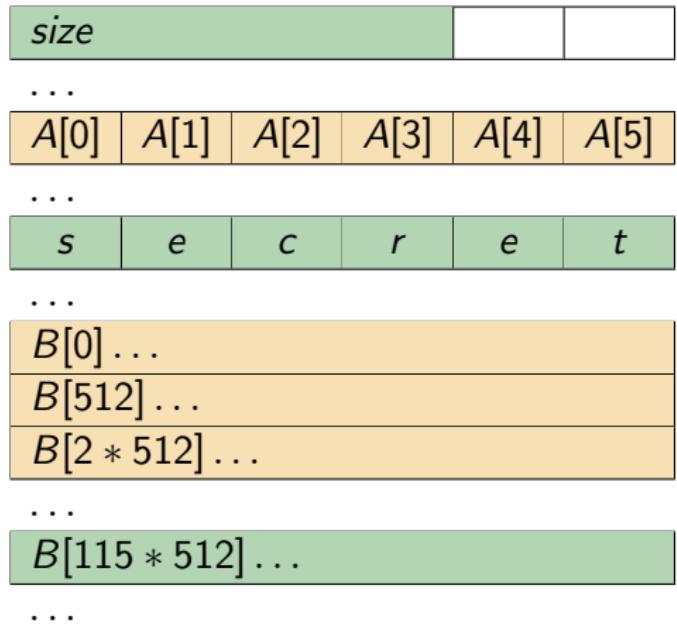
Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

Also `size` has arrived

- Thus, `y<size` can be computed to be false.
- Thus, the processor reverts to the state before.



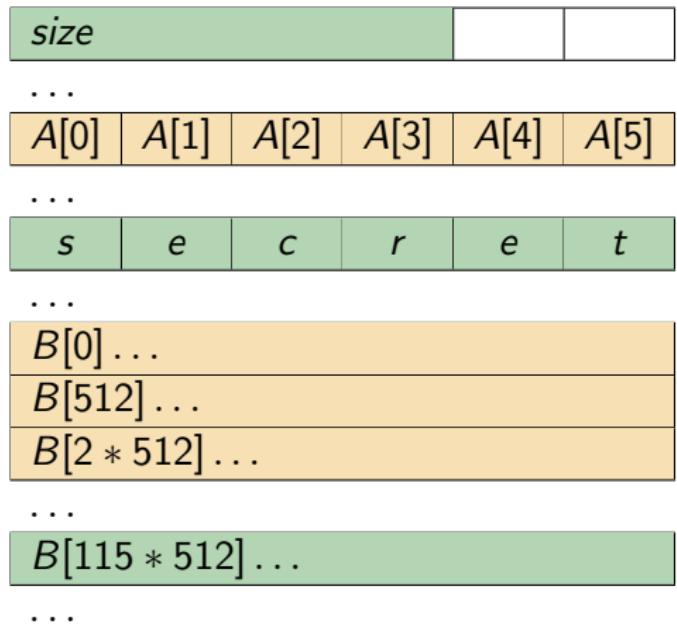
Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

Also `size` has arrived

- Thus, $y < \text{size}$ can be computed to be false.
- Thus, the processor reverts to the state before.
- **But the cache remains!**



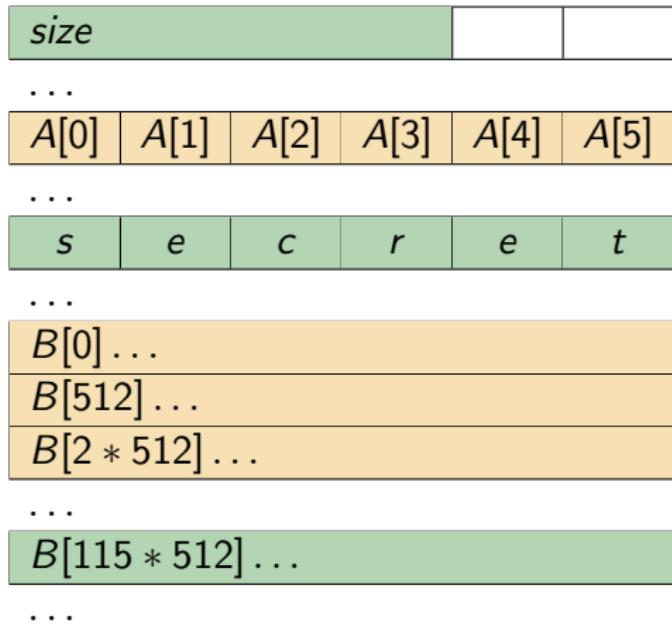
Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

Also `size` has arrived

- Thus, $y < \text{size}$ can be computed to be false.
- Thus, the processor reverts to the state before.
- **But the cache remains!**



The intruder can now try to access $B[k * 512]$ for $k \in \{0..255\}$.

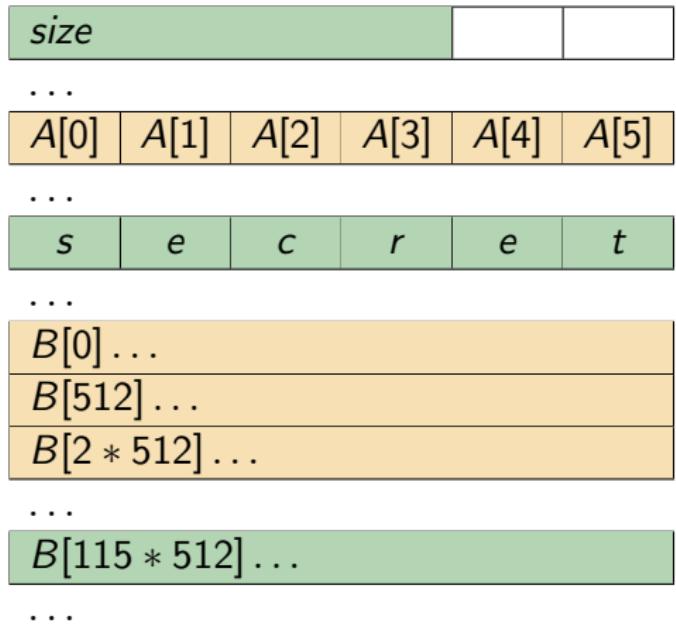
Spectre: Example

Victim code

```
if (y<size)
    temp &= B[A[y]*512]
```

Also `size` has arrived

- Thus, $y < \text{size}$ can be computed to be false.
- Thus, the processor reverts to the state before.
- **But the cache remains!**



The intruder can now try to access $B[k * 512]$ for $k \in \{0..255\}$. This is fast iff $k = 115$: he knows the first letter of the secret.

Spectre: Countermeasures?

What to do about this?

- Speculative load hardening²
- Idea: the compiler Clang/LLVM inserts extra operations to mitigate the leak.

²Chandler Carruth: *Speculative load hardening*
<https://llvm.org/docs/SpeculativeLoadHardening.html>

Speculative load hardening

The **hardened code** contains a few more operations:

```
mov      size, %rax
mov      y, %rbx
mov      $0, $rdx
cmp      %rbx, %rax
jbe     END
cmovbe $-1, %rdx
mov      A(%rbx), %rax
shl      $9, %rax
or       %rdx, %rax
mov      B(%rax), %rax
or       %rdx, %rax
and      %rax, temp
END:    ...
```

Side Channel Analysis

- Side channels are usually leaks that the designers never have thought of.
- Many problems exist for years without getting noticed – how much else is out there?
- Spectre: tons of variations emerging e.g. remote attacks.³
- Countermeasures: for instance compiler-based techniques.⁴
- Very active research field:
 - ★ In order to prove a solution correct, one needs a precise model first.
 - ★ For instance Speculative non-interference and the Spectector tool.⁵

³Michael Schwarz et al.: *NetSpectre*, ESORICS 2019.

⁴Chandler Carruth: *Speculative load hardening*

<https://llvm.org/docs/SpeculativeLoadHardening.html>

⁵Marco Guarnieri et al.: *Spectector*, Security & Privacy 2020.



Conclusions

- Software security is extremely subtle
 - ★ It is very hard to design systems that are impeccable
- Every small imperfection can be disastrous
 - ★ Imperfections can be amplified by an attacker in unforeseen ways.
 - ★ Don't rely on "*this cannot be exploited*"-statements
- Suggestions for secure software
 - ★ Prudent: Follow good engineering practices
 - ★ Formalize: Try to be precise about your assumptions and goals
 - ★ Proof: Try to formally test/verify automatically or manually.

Students who like this course also liked... :-)

There are several related courses, e.g.

- on security: 02244 Logic for Security
- on formal methods: model-checking, program analysis, computer science modeling

If you are looking for a Bsc/Msc thesis topic in security or formal methods, come by for a chat!

Legal Aspects of Computer Security



$$\Theta^{\sqrt{17}} + \Omega^{\int \delta e^{i\pi} = \infty} = \sum_{x^2 > 0}$$

DTU Compute

Department of Applied Mathematics and Computer Science

Basic Ideas

- Legal issues are related to questions such as:
 - What support does the law give to the protection of computers, programs and data?
 - What support does the law give to the protection of intellectual property?
 - What rights does the law give "legal persons" with respect to computers, programs and data?
- More specific issues:
 - Computer crime (definition, evidence, punishment,...)
 - Legal protection of code and data (copyright, patents,...)
 - Programmers' and employees' rights
 - Protection of personal data
 - Consumer protection

The Role of Computers in Crime

- Computer (or network) as target
 - Using computer(s) to attack a victim's computer
 - Attack on Confidentiality, Integrity or Availability of data or systems
 - Cyber Terrorism and Cyber Warfare
- Computer as tool
 - Fraud - Phishing, Nigerian 419 (after Fraud § in Nigerian Criminal Code)
 - Ransomware (WannaCry, NotPetya, ...)
 - Gambling
 - Copyright infringements (aka piracy)
 - Harassment (aka cyber bullying), stalking, etc.
- Computer as accomplice
 - Personal information (diaries, downloaded e-mails,)
 - *Other evidence unknown to suspect - web-history, cookies, ...*
 - Contraband - digital goods, copyrighted material, (child) pornography
 - Stolen information – trade secrets, credit card data
 - Monetizing proceeds of cyber crime (online marketplaces, Bitcoins, ...)

Cyber Crime

- Problem of jurisdiction
 - Laws are mostly national, cyber crime is typically transnational
 - *International treaties/conventions may codify crime in several countries*
 - *Netiquette may (self-)regulate some unwanted behaviour*
 - Where is crime committed?
 - *Who should investigate, prosecute and sentence*
 - Location of victim, criminal or beneficiary?
 - *The crime may not be a legal offence in all relevant jurisdictions*
 - How to investigate
 - *Collecting evidence requires collaboration among law enforcement agencies*
 - How to get hold of the accused person(s) / evidence?
 - *Extradition agreements between national states*
 - How to punish criminals
 - *Some criminals may be tried in absentia*

Problem of Jurisdiction



Convention on Cyber Crime

- Convention established by the Council of Europe
 - 30 states sign Convention at opening ceremony in Budapest in 2001
- First international treaty on cyber crimes, dealing particularly with:
 - Infringements of copyright
 - Computer-related fraud
 - Child pornography
 - Violations of network security
 - Hate crimes and Racism as an addendum (optional extras)
- Contains a series of powers and procedures such as search of computer networks and interception
- Main objective is to pursue a common criminal policy aimed at protection of society against cyber-crime, especially by adopting appropriate legislation and fostering international co-operation

Protecting Intellectual Property

- Three legislative frameworks are applicable to programs and data:
 - Copyright law (publication of works of art)
 - *Copyright law was conceived to protect works of art, music, literature and written scholarship*
 - *Provides incentive to produce works of art*
 - Patent law (public information about inventions)
 - *Patent law was conceived to protect inventions and innovation in science, technology and engineering*
 - *Provides an incentive to inventors to disclose their inventions*
 - Trade secrets law (secret information incl. data and processes)
 - *Trade secrets identify information that must be kept secret*
 - Punish people who reveal the secret to outsiders
 - *Provides a legal framework to deal with disclosure of confidential info.*

Copyrights, Patents & Trade Secrets

	Copyright	Patent	Trade Secret
Protects	Expression of idea, not idea itself	Invention – the way something works	A secret, competitive advantage
Protected objects made public	Yes, intention is to promote publication	Design filed at Patent Office	No
Requirement to distribute	Yes	No	No
Ease of filing	Very easy, do-it-yourself	Very complicated; specialist lawyer suggested	No filing
Duration	Life of human originator plus 70 years, or total of 95 years for a company	19 years	Indefinite
Legal protection	Sue if unauthorized copy sold	Sue if invention copied	Sue if secret improperly obtained



Data Protection

- Large amounts of data are being collected about all of us



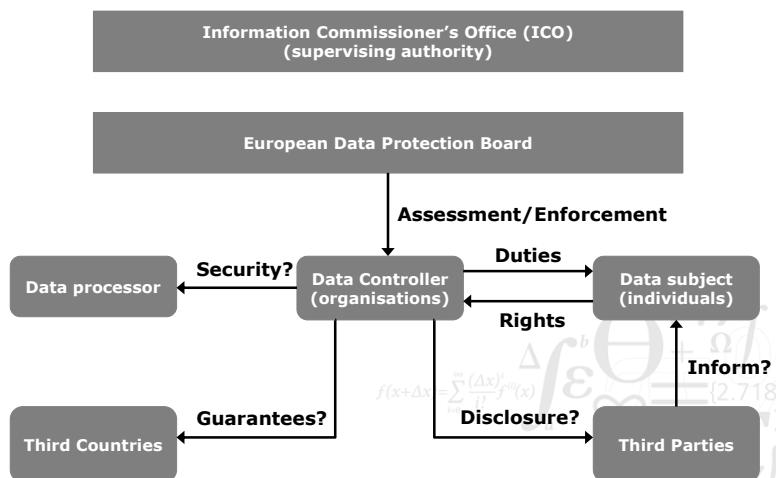
9

DTU Compute Technical University of Denmark

02239 – Data Security



GDPR - Data Protection Model



10

DTU Compute Technical University of Denmark

02239 – Data Security

GDPR - Definitions

- Natural person = a living individual
- Natural persons have rights associated with:
 - The protection of personal data
 - The protection of the processing of personal data
 - The unrestricted movement of personal data within the EU
- In material scope:
 - Personal data that is processed wholly or partly by automated means;
 - Personal data that is part of a filing system, or intended to be
- The Regulation applies to controllers and processors in the EU irrespective of where processing takes place
- It applies to controllers not in the EU

GDPR - Remedies, liabilities and penalties

- Natural Persons have rights
 - Judicial remedy where their rights have been infringed as a result of the processing of personal data.
 - *In the courts of the Member State where the controller or processor has an establishment*
 - *In the courts of the Member State where the data subject habitually resides*
 - Any person who has suffered material, or non-material, damage shall have the right to receive compensation from the controller or processor
 - Controller involved in processing shall be liable for damage caused by processing
- Administrative fines
 - Imposition of administrative fines will in each case be effective, proportionate, and dissuasive (*No administrative fines in Denmark, fines imposed by the courts*)
 - *taking into account technical and organisational measures implemented;*
 - €10,000,000 or, in the case of an undertaking, up to 2% of the total worldwide annual turnover of the preceding financial year
 - €20,000,000 or, in case of an undertaking, 4% total worldwide annual turnover in the preceding financial year

GDPR - Personal Data Breaches (Article 33)

- The definition of a Personal Data Breach in GDPR:
 - A 'personal data breach' means a breach of security leading to the accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed.
- Obligation for data processor to notify data controller
 - Notification without undue delay after becoming aware
 - No exemptions
 - All data breaches have to be reported
- Obligation for data controller to notify the supervisory authority
 - Notification without undue delay and not later than 72 hours
 - Unnecessary in certain circumstances
 - Description of the nature of the breach
 - No requirement to notify if unlikely to result in a high risk to the rights and freedoms of natural persons
 - Failure to report within 72 hours must be explained

GDPR - Rights of Data Subjects

- The controller shall take appropriate measures to provide any information ... relating to processing to the data subject in a concise, transparent, intelligible and easily accessible form, using clear and plain language (Article 11-1)
- The controller shall facilitate the exercise of data subject rights (Article 11-2)
 - Rights to
 - *Consent*
 - *Access*
 - *Rectification*
 - *Erasure*
 - *Objection*
 - the right to data portability;
 - the right to withdraw consent at any time;
 - the right to lodge a complaint with a supervisory authority;
 - The right to be informed of the existence of automated decision-making, including profiling, as well as the anticipated consequences for the data subject

GDPR - the Principle of Accountability

- Governance: Board accountability
 - Corporate risk register
 - Nominated responsible director
- Clear roles and responsibilities
 - Data Protection Officer
- Privacy Compliance Framework
 - PIMS/ISMS
 - Cyber incident response
 - Cyber Essentials a minimum security standard
 - Certification and data seals (Article 42) –ISO 27001
- Data Protection by Design and by Default
 - Data Flow Audits
 - Data Protection Impact Assessments (DPIA)
 - *Mandatory for many organizations*
 - *Legal requirements around how performed and data collected*

GDPR – Lawfulness (Article 5 & 6)

- Secure against accidental loss, destruction or damage
- Processing must be lawful –which means, inter alia:
 - Data subject must give consent for specific purposes
 - Other specific circumstances where consent is not required
 - *So that controller can comply with legal obligations etc.*
- One month to respond to Subject Access Requests & no charges
- Controllers and processors clearly distinguished
 - Clearly identified obligations
 - Controllers responsible for ensuring processors comply with contractual terms for processing information
 - Processors must operate under a legally binding contract
 - *Note issues around extra-territoriality*

GDPR – Consent (Article 7-9)

- Consent must be clear and affirmative
 - Must be able to demonstrate that consent was given
 - Silence or inactivity does not constitute consent
 - Consent must be clear, intelligible, easily accessible, to be binding
 - Consent can be withdrawn at any time, and it must be as easy to withdraw consent as give it
- Special conditions apply for children (under 16) to give consent
- Explicit consent necessary for processing sensitive personal data
 - Race, ethnic origin, gender, etc.
 - Specific circumstances allow non-consensual processing,
 - *Regulatory or legal requirements*
 - *To protect vital interests of the data subject*
 - ...
- Secure against accidental loss, destruction or damage (article 5)

GDPR – Transparency (Article 12 – 18)

- Any communications with a data subject must be concise, transparent, intelligible
 - This excludes legal jargon
- Controller must be transparent in providing information about itself and the purposes of the processing
- Controller must provide data subject with information about their rights
- Specific provisions (Article 14) covering data not obtained directly from the data subject
- Rights to access, rectification, erasure ('right to be forgotten'), to restriction of processing, and data portability

GDPR - Privacy by Design (Article 25 et seq.)

- Privacy must now be designed into data processing by default
- Data controllers/processors not established in the EU must designate a representative
- Data Privacy Impact Assessments mandatory (article 35)
 - For technologies and processes that are likely to result in a high risk to rights of data subjects
- Data audits
 - GDPR applies to existing data, as well as future data
 - Privacy may have to be designed in retrospectively
 - Organizations need to identify what PII they hold, where, on what grounds, and how it is secured in a way that will meet requirements of GDPR

GDPR - Security of Personal Data (Article 32)

- Sixth Principle: Data must be processed "*in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures*"
- A requirement for data controllers and data processors to implement a level of security appropriate to the risk, including:
 - pseudonymisation and encryption of personal data;
 - ensure the ongoing confidentiality, integrity and availability of systems;
 - a process for regularly testing, assessing and evaluating the effectiveness of security measures;
 - security measures taken need to comply with the concept of privacy by design;
- Certifications demonstrate intent: Cyber Essentials, ISO 27001

GDPR - Data Protection Officer (DPO)

- DPO mandatory in organizations processing substantial volumes of PII (Article 37)
- A protected position, reporting directly to senior management
 - Appropriately qualified
 - Consulted in respect of all data processing activities
- Will be a 'good practice' appointment outside the mandatory appointments
- Most staff dealing with PII (e.g. HR, marketing, etc.) will need at least basic training
- Staff awareness training also critical (accidental release of PII could have financially damaging consequences)

GDPR - International Transfers (Article 44)

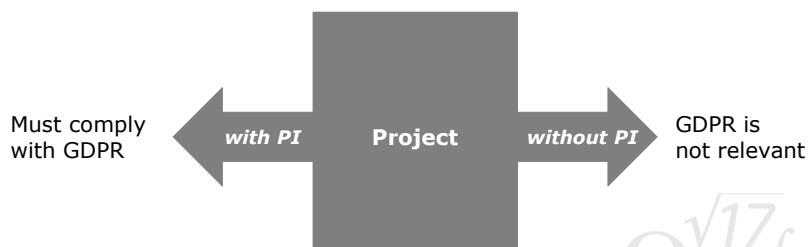
- Any transfer of personal data by controller or processor shall take place only if certain conditions are complied with:
 - Transfers on the basis of adequacy;
 - Transfers subject to the appropriate safeguards
 - Binding corporate rules apply
- All provisions shall be applied to ensure the protection of natural persons is not undermined
- To countries with similar data protection regulations
 - Cloud providers are a key risk area
 - *Schrems II decision in European Court of Justice raises questions about transfer to US and US owned companies*
 - Highest penalties apply to breaches of these provisions

Nine Steps to GDPR compliance

1. Establish governance framework – board awareness, risk register, accountability framework, review
 2. Appoint and train a DPO
 3. Data inventory – identify processors, unlawfully held data
 4. Data flow audit
 5. Compliance gap analysis
 - Ensure FPN and SAR documents and processes are robust and legal
 6. DPIA and security gap analysis
 - Penetration testing
 7. Remediate
 1. Privacy compliance framework
 2. Cyber Essentials/Ten Steps to Cyber Security/ISO 27001
 8. Data breach response process (NB: Test!)
 9. Monitor, audit and continually improve
- NB: steps can be tackled in parallel

GDPR in Practice

- When considering the data you collect when implanting a project



- GDPR can be ignored *if you do not collect personal data*
 - This is one reason why privacy enhancing technologies are so important

Protocol Security Verification Tutorial

Sebastian Mödersheim,
DTU Compute, samo@dtu.dk

Version of March 2020

1 Roadmap

This tutorial gives an introduction to modeling security protocols and the methods for automated verification that is hopefully easier accessible than research papers. For concreteness it uses OFMC, an automated protocol verification tool written by the author, and thus this document also serves as a tutorial for OFMC. Several other methods and tools are briefly discussed in order to give a broader perspective.

In the first part, we will entirely focus on precisely describing security protocols, their security goals and a model of the intruder, so that “the protocol is secure” is a mathematical statement that is either true or false, and there is a chance to prove or disprove this statement. Disprove also entails finding a counter-example to security: an attack.

The second part is concerned with methods to automatically find the correct answer, i.e., to find a proof of security or an attack. This is difficult since in general there will be an infinite number of things that can happen in a protocol, so that exhaustive search is impossible. Even under reasonable restrictions that make the search space finite, this is often still practically infeasible. We will focus on two techniques that can in practice often deal well with protocol security problems. One is based on symbolic representation with constraints and it can often find attacks quickly; the other is based on abstract interpretation and it can often find proofs of security. We will also discuss why the problem is in general undecidable, i.e., there is no hope for finding any verification method that will always answer correctly for all protocols.

In the third part, two more advanced topics, namely channels and compositionality. The idea is that on the Internet we use a lot of protocols in parallel and in a stacked fashion, e.g., using TLS to establish a secure channel and run a banking application over that channel. While one could theoretically verify such a composed system, this becomes easily too complex to handle. Also, we would like protocol designers to design a protocol like TLS independent of the actually payload protocols that it will be used for later. The key of compositional reasoning is to just allow this component-wise development, i.e., that the composed system is secure if the components are.

Contents

1 Roadmap	1
I Modeling Protocols	4
2 Example: Building a Key-Establishment Protocol	4
2.1 First Attempt	4
2.2 Second Attempt	8
2.3 Third Attempt	11
2.4 Fourth Attempt	13
2.5 Fifth Attempt	15
2.6 Final Version	16
3 Example: TLS	16
3.1 Unilateral TLS	18
3.2 Diffie-Hellman	18
4 The Syntax of AnB	20
5 From Alice and Bob to Strands – Intuition	23
6 Term Algebra and All That	26
6.1 Signatures	26
6.2 Algebra	27
6.3 The Free Algebra	28
6.4 \star Quotient Algebra	29
7 The Dolev-Yao Intruder Model	31
7.1 Intruder Deduction	31
7.2 Automating Dolev-Yao	33
8 Transition Systems	34
8.1 \star Instantiation	35
8.2 States and Transitions	36
8.2.1 Transition: Sending	37
8.2.2 Transition: Receiving	37
8.2.3 Transition: Checking	38
8.2.4 Transition: Events	39
9 Security Goals	41
9.1 Secrecy	41
9.2 Authentication	41
10 \star The Algebraic AnB Semantics	43
10.1 Message model	43
II Automated Verification	46
11 Introduction	46
11.1 The Sources of Infinity	46
11.2 \star An Undecidability Result	47

12 Symbolic Transition Systems	49
12.1 Transition: Receiving	52
12.2 Transition: Sending	52
12.3 Events and Equations	53
13 The Lazy Intruder	53
13.1 Solving Constraints	54
13.2 Composition	54
13.3 Unification	55
13.3.1 Computing the Most General Unifier – mgu	55
13.3.2 The Lazy Intruder Unification Rule	56
13.4 Simple Analysis	57
13.5 ★ Full Analysis	57
13.6 Constraint Solving Complete Example	59
13.7 Summary	65
13.8 Simple Constraints	66
13.9 ★ Correctness of the Lazy Intruder	66
14 Abstract Interpretation	68
14.1 An Example	69
14.2 Two Abstractions	72
III Advanced Topics	75
15 Channels and Composition	75
15.1 Bullet Notation	75
15.2 A Cryptographic Implementation	77
15.3 Channels as Assumptions – Channels as Goals	78
15.4 Compositionality	78
15.5 Pseudonymous Channels	79
16 Guessing Attacks	81

Part I

Modeling Protocols

A danger in designing formal models in computer science lies in mixing the model with algorithmic aspects, i.e. the question *what* we want to check with the question *how* to perform the check. This can lead to models that are somewhat un-intuitive and hard to use, because they are a poor compromise between what one wants to express and what one can handle with a particular analysis method. Therefore we try to clearly separate the modeling in this part and the formal verification methods in the next part, and obtain here a model regardless of whether this can be automatically analyzed or not.

A clear, simple and declarative modeling language is actually the aim of the main input language of the OFMC tool: *AnB* [23]. AnB is based on the popular Alice-and-Bob notation that is informally used in many textbooks. However AnB is a formal language – like a programming language or a logic – because it has:

- a syntax (what is a valid protocol in the language AnB?)
- and a formal semantics (what does an AnB specification mean?)

That is, it has a programming language flavor, since there is a translator that generates executable protocol implementations in JavaScript; since this is however for an extension of AnB called *SPS* that is currently still in a prototypical stage, we refer here only to the literature [1].

OFMC also uses a lower-level input language, called *IF* (Intermediate Format) [2]. This is based on set-rewriting and considerably more tailored to the needs of the automation, in fact it is OFMC’s “native language”. Internally, AnB is translated into IF. IF is more expressive than AnB, but hard to use directly (i.e., without translation from a language like AnB). We do not discuss IF in this tutorial since its technical details may be too distracting, but will rather use a much simpler and nicer formalism to describe the behavior of honest agents in a protocol: *strands*. In fact, we define the semantics of AnB by translation into strands and how strands – together with the intruder – give rise to a *state transition system*.

Notation: A few central points are summarized in such a box.

OFMC GUI A graphical user interface for OFMC has been developed by Úlfur Jóhann Edvardsson and Veronica Julie Lodskov Hoffmann, see <https://github.com/ulfur88/OFMCGUI>. It provides syntax highlighting, and a view both of protocols and attacks as sequence charts, highlighting corresponding steps of the attack and the protocol description, as shown in Fig. [1]. We will in the following focus on OFMC and describe outputs as done by OMFC (in the standardized AVISPA output format), but discuss briefly where GUI actually slightly rewrites attacks to make them easier to read.

2 Example: Building a Key-Establishment Protocol

Before we go into formal details, we want to first introduce AnB informally at hand of some examples. We follow here an example of protocol development found at the beginning of the book *Protocols for Authentication and Key-Establishment* by Colin Boyd and Anish Mathuria [6]. The point in that book is to show the stepwise development of a protocol and motivate each step as an answer to a security problem. Here, we use it to illustrate AnB and the output of OFMC instead.

2.1 First Attempt

We want to write a simple *key-exchange protocol* that establishes a shared symmetric key between two parties Alice and Bob that do not have a security relationship so far. This newly established

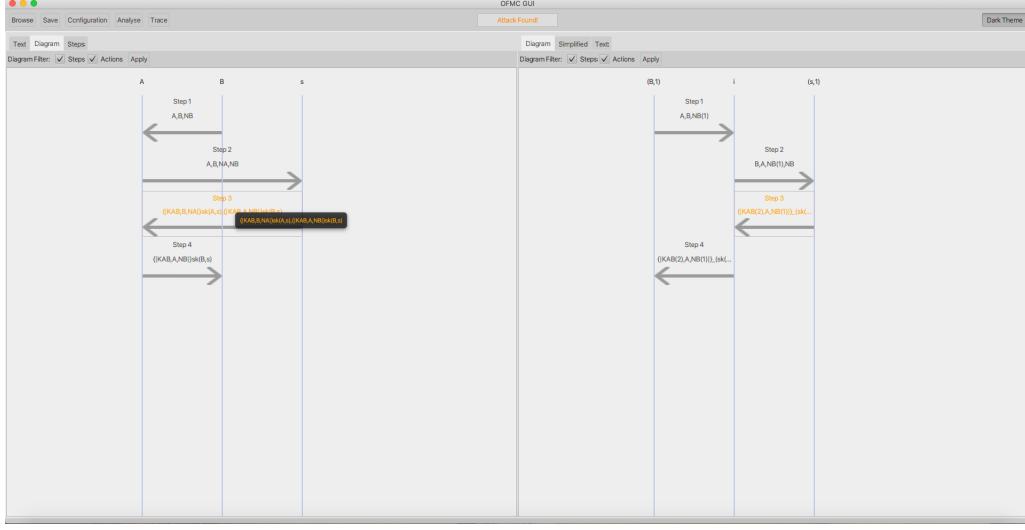


Figure 1: OFMC GUI

key shall then allow them to communicate securely by encrypting messages with that key. Since we cannot establish such a secure connection out of thin air, we need some form of existing relationship to begin with, and here it will be a *trusted third party* s (for “server”). If we entirely omit all the cryptography for now, a very simple (and trivially insecure) protocol is the following:

Protocol: KeyEx # First Attempt

Types: Agent A,B,s;
Symmetric_key KAB

Knowledge:

A: A,B,s;
B: A,B,s;
s: A,B,s

Actions:

A->s: A,B
s creates key KAB
s->A: KAB
A->B: A,KAB

Goals:

A authenticates s on KAB,B
B authenticates s on KAB,A
KAB secret between A,B,s

Actions and the Communication Medium Let us begin with the **Actions** section. Here we see the exchange of three messages: first, A tells the server s that she¹ would like to talk to B . The server creates a *fresh* symmetric key KAB and sends it back to her in the second step. In

¹Throughout this tutorial, we assume that A (Alice) is female, B (Bob) and i (the intruder) are male, and all others (servers etc.) are neutrum.

the third step she forwards the key to B and they can start communicating.

In fact, the communication here is *asynchronous*. The notation $A \rightarrow B : M$ indicates two things:

- that A sends a message M on an *insecure* communication medium;
- that B waits for receiving a message of the given form and then (and only then) continues with his next step.

The medium could be the Internet or a wireless connection: it is possible that an unauthorized third party listens to (and records) the transmitted messages and inserts messages under a fake sender ID. Moreover there is no guarantee that a sent message will arrive at the intended destination.

AnB also requires that in a sequence of messages, the receiver of one message is the sender of the next message.²

Variables and Constants A , B and KAB —and all identifiers that begin with upper-case letters—are *variables*. That means that they are placeholders for a concrete value (the real name of an agent or a concrete symmetric key in this case) that will be filled in when the protocol is actually executed. Identifiers that start with a lower-case like s are *constants*.

Roles Variables and constants that are declared to be of type `Agent` are called *roles*. The use of variables and constants is crucial here: we will allow that variables of type `Agent` can be instantiated arbitrarily with agent names. This includes the special agent `i` — the *intruder*. We will discuss below in more detail what the intruder can and cannot do, but for now it is worth pointing out that by default all protocol roles should be specified as variables of type `Agent`, allowing everybody to participate in the respective role under their real name, including a *dishonest* person. This basically models that not all protocol participants are necessarily honest. It turns out that many protocols have surprising attacks when allowing dishonest participants. A specification of a constant like s here is only to be used to model a *trusted third party*: a party that we require to be honest for the protocol to work.³ In a key-exchange protocol, a dishonest server can often trivially break the security goals. Therefore, when we want to model such an honest participant, we specify a constant like s that cannot be instantiated by the intruder.

Knowledge For each role of the protocol, one needs to specify an initial knowledge. This knowledge is essential to the meaning of the AnB specification as we will discuss at several points throughout this tutorial. In particular, we will check for every role and every message that they have to send, whether this message can be constructed by that role from the initial knowledge plus all messages it has received before. If this is not the case, then the specification has an error: it is *unexecutable* and will be rejected by the translator to IF.

Variables in Knowledge MUST be of Type Agent The initial knowledge will usually include the knowledge of all roles of the protocol.

It is crucial that all terms in the initial knowledge contain only variables of type agent. For instance, in our specification it would be an error to declare the variable KAB as part of the knowledge.

We show in the next version of the protocol, how to model long-term keys (using functions).

²This is not a limitation, since for instance to model $A \rightarrow B : M_1$ followed by $C \rightarrow D : M_2$ one could insert another message $B \rightarrow C : \text{dummy}$ where `dummy` is part of the initial knowledge of B . See, however, the more efficient solution of “piggy-backing” in the next variants of the protocol.

³The word “trust” has often lead to confusions, since the statement “ A trusts B ” has nothing to do with the question whether B is actually *trustworthy*. We actually do not work with trust-statements, but rather only with honest/dishonest. Terms like “trusted third party” are so common however, that we use them here as well in the sense of “honest party”.

Fresh Values All variables that are not part of the initial knowledge are freshly generated by the agent who first uses them, in our example, KAB is freshly generated, and the generator is s since it sends the first message that KAB occurs in. Fresh means in reality: an unpredictable random number; in the abstract formal world it means: when executing this step, the variable is instantiated with a new constant (and the intruder initially does not know this constant).

Secrecy Goals The most simple goal is secrecy: we denote a term and say between whom it shall be secret. In this case, the secret is KAB and it is shared between all participants of the protocol. The specification of a group of people that share the secret is necessary: we allow the intruder to play role A or role B and in this case, he is of course allowed to know the shared key of that particular protocol run. It is however an attack, if the intruder finds out a shared key of a protocol run between two honest agents playing in roles A and B . (And due to lack of encryption, this secrecy goal is trivially violated in the given protocol.)

We postpone the discussion of the more involved authentication goals to a later example.

Interpreting Attacks We run OFMC with the command line `ofmc KeyEx1.AnB` (the file is found in the tutorial folder of OFMC). This will start a search with a *bounded number of sessions*, i.e., it limits how many runs of each role of the protocol we have (although with arbitrary agents playing the roles). OFMC starts with 1 session (i.e., one “copy” of each role), then 2 sessions, and so on, until it finds an attack or the user interrupts. So for a correct protocol, OFMC does not terminate (in this setting).⁴

For our first protocol we get the following output in the *AVISPA Output Format*:

```
SUMMARY
  ATTACK_FOUND
GOAL
  secrets
...
ATTACK TRACE
i -> (s,1): x29,x28
(s,1) -> i: KAB(1)
i -> (i,17): KAB(1)
i -> (i,17): KAB(1)
```

This indicates, unsurprisingly, that we have an attack against the secrecy goals. In the first step of the attack, the intruder sends a message to the server s . Here $(s, 1)$ indicates that it is the server in session 1. The point of this identifier is to tell us, when several sessions of the protocol run in parallel, which messages belong to the same run of an agent. The message that the intruder sends is $x29, x28$. This is a pair of variables. The variables $x29$ and $x28$ indicate that it is completely irrelevant for the attack what the intruder chooses here. In this case, the server expects to receive two agent names, but just any will do for this attack.⁵

A side note for users of the OFMC GUI: there, most of these variables in the attack are renamed to have more intuitive names than $x29$ and the like. For variables that represent Agent names, it chooses role names of the protocol, in this case A and B . Decisive for the naming is the first message in which they occur, namely as which role of the protocol the recipient of that message would read them. If several different variables occur in the output that should correspond to the same protocol role, then GUI OFMC will choose similar names, e.g. $A2$ and $B2$.

Back to the attack description, note that the intruder started the protocol with the server without the agent $x29$ having done anything. This is because the network is asynchronous, i.e., there is no guarantee that the intended recipient will actually receive the message. In fact it is

⁴To enforce termination, one may also specify a fixed number of sessions, say 3, with option `--numSess 3`.

⁵To be entirely precise, there are two choices of agent names that would not work: $x29 = i$ or $x28 = i$; in these cases the trace would not be a violation of secrecy. The exclusion of particular values is currently not shown by OFMC.

often the goal of a protocol to get to a state where all parties “are on the same page”. The server now responds to the request by creating a new key and sending it. Fresh values in an attack will always be the name in AnB followed by a unique number (which is in fact a session number).

Usually attacks will consist of pairs of steps where the intruder sends a message to an honest agent and receives an answer from that agent, like the first two messages in the attack here. This reflects an efficient view of the protocol analysis problem: the intruder *is* the communication medium in the sense that all messages received by an honest agent come from the intruder and all messages sent by an honest agent are received by the intruder. Thus the intruder uses the honest agents like *oracles*.

With this answer from the server (the second message of the attack), the attack is already completed: the intruder now knows the shared key of two agents $x29$ and $x28$ that he can freely choose—violating secrecy. The last two lines of the attack are just a technicality of OFMC: all steps of the form $(i, 17)$ are just result of an internal check that the intruder could produce a secrets that he was not supposed to see.

2.2 Second Attempt

We clearly need to protect the transmission of the secret shared key KAB and for that, we would like to assume that every agent (including the intruder) *initially* has a shared key with the server. We may for instance imagine that s provides wireless access, but everyone who wants to use it has to first register. Let us say this registration happens offline (possibly checking a photo ID) and involves installing a unique username and a secret key with the server. In principle this secret key may be a password, but there are a number of problems with that as discussed in subsequent sections. So let us for now assume that s has with every user A a strong shared key $sk(A, s)$, i.e., chosen by a cryptographic random-number generator.

Modeling Long-Term Keys The important thing about the shared key is that it is not freshly generated in a session but it is perpetual information. Recall that we are not allowed to have any variable like KAB that is not of type Agent in the initial knowledge of a role. However, we can declare new function symbols and use them to model long-term keys as a function of the agents who share them, e.g., use $sk(A, s)$ to represent the shared key of A and s . As such a term contains only variables of type Agent, it is allowed to include it in the initial knowledge of a role.

The second attempt to our protocol is now as follows, where AnB uses the notation $\{|M|\}K$ for the symmetric encryption of message M with key K :

```

Protocol: KeyEx # second attempt

Types: Agent A,B,s;
       Symmetric_key KAB;
       Function sk

Knowledge:
A: A,B,s,sk(A,s);
B: A,B,s,sk(B,s);
s: A,B,s,sk(A,s),sk(B,s)

Actions:

A->s: A,B
s->A: {| KAB |}sk(A,s), {| KAB |}sk(B,s)
A->B: A,{| KAB |}sk(B,s)

```

and the goals are the same as before.

Use of Functions Note that we have declared `sk` as a “constant” of type `Function`. Here, OFMC currently does not do any type checking, e.g., we did not specify that `sk` should be a function of two arguments, and if we use it with a different number of arguments, OFMC will not complain (this may be a source of errors).

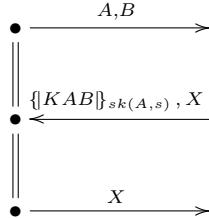
Obviously every agent initially knows his or her shared key with the server. For the server we specify only the knowledge of the shared keys with the other two roles.

Executability In this new version of the protocol, the server does not transmit the key KAB unprotected as in the first version. Instead, it creates two encryptions, one using the shared key with A and another one using the shared key with B . These two encrypted messages are sent to A . According to her knowledge, A can only decrypt the first of the two messages it receives, while the second one cannot be analyzed by A . A is supposed to forward this second package to B who has the necessary shared key to decrypt that message. So at least in a run where the intruder does not interfere, all agents have enough knowledge to produce all messages they have to and end up with a copy of the shared key KAB .

A central observation here is that A cannot check the second part of the message from the server. Especially, if we think of an intruder producing such a message (possibly recycling older messages he has seen on the communication medium), only the first part needs to have correct format, while the second part can be any message X . A will then pass X on to B . If we look at A in isolation, we may describe it as a program of the form

```
send(A,B); receive({|KAB|}sk(A,s),X); send(X);
```

In fact, this sequence of send and receive events is called a *strand* [15] and can be used to describe the behavior of an honest agent. We sometimes also like to use the following graphical notation for strands where outgoing arrows represent sending messages and incoming arrows represent receiving messages:



The Model of Symmetric Encryption Many cryptographers may associate with the term “symmetric encryption” only the pure encryption, without any means of protecting integrity such as a message authentication code (MAC). Such a pure encryption would be vulnerable to the intruder manipulating bits of the ciphertext and thereby changing the encrypted text so that the recipient cannot detect the manipulation. We believe that there are only very few cases in protocol verification when we actually need the pure symmetric encryption, but almost always we also need the integrity. We therefore model in AnB with $\{|M|\}K$ a primitive that includes the integrity. For our concrete example that means, when A receives the two encrypted messages from the server, she will decrypt the first one to which she has the key; the integrity mechanism of the primitive allows her to check (with overwhelming probability) that the received message is indeed correctly encrypted with the right symmetric key $sk(A, s)$ and not some message manipulated by the intruder. Put another way, if the intruder sends any other message that is not of the form $\{|M|\}sk(A, s)$, then A will detect that and refuse it. We actually do not even model that the intruder tries sending ill-formed messages to honest agents that they will refuse.

An Attack Against Weak Authentication Goals Running this second example with OFMC, we get the following attack:

SUMMARY

```

ATTACK_FOUND
GOAL
    weak_auth
...
ATTACK TRACE
i -> (s,1): x29,x401
(s,1) -> i: {|KAB(1)|}_(sk(x29,s)),{|KAB(1)|}_(sk(x401,s))
i -> (x401,1): x27,{|KAB(1)|}_(sk(x401,s))

% Reached State:
%
% request(x401,s,pBsKABA,KAB(1),x27,1)
% witness(s,x29,pAsKABB,KAB(1),x401)
% witness(s,x401,pBsKABA,KAB(1),x29)
% ...

```

The attack is a violation against *weak authentication* (which corresponds to Lowe's *non-injective agreement* [19]). The weak authentication is part of the standard (strong) authentication goal (which corresponds to Lowe's *injective agreement* [19]) that we have specified. We have displayed in the attack trace three facts from the comments that OFMC gives out as part of the *Reached State comment*. These facts are sometimes helpful in understanding an authentication attack as they reflect what the honest agents "think" has happened from their point of view and that we review in detail now. But let us first understand it just from the message trace.

What the intruder has done here is that he chose two arbitrary agent names `x29,x401` and sent them to the server, who then created a new shared key `KAB(1)` for these two agents. The intruder sends this message now to `x401`, but claiming to be `x27`, i.e., a different person than who the server generated the key for. To `x401` this message looks like a perfectly correct step 3 of the protocol, so it will believe that `KAB(1)` is a shared key with `x27`. So the intruder has not found out any secret, but he managed to break the authentication: server `s` and recipient `x401` disagree on who is playing role `A` of the protocol in this session, i.e., who the key is shared with, thus it is a violation of the goal `B authenticates s on A,KAB`.

We will define the authentication goals formally later of course, but it can sometimes help understanding the point of view of an agent by looking at the witness/request facts (in comments of the attack output): the fact `witness(s,x29,pAsKABB,KAB(1),x401)` means that the server `s` intends to run the protocol with agent `x29` in role `B`, using `KAB(1)` as a key (for variable `KAB` of the protocol) and `x401` for role `A` of the protocol. (The identifier `pAsKABB` is just a technicality to distinguish several similar authentication goals.) In contrast, the fact `request(x401,s,pBsKABA,KAB(1),x27,1)` shows what `x401` is thinking: he thinks `x27` is playing role `B`. (The number 1 is for strong authentication, as explained later.)

More generally, the violation of weak authentication is given if there is a request fact without a matching witness fact. For a goal of the form `B authenticates A on M`, the witness fact reflects the point of view of `A` while the request fact reflects the point of view of `B`. This goal should thus be used if the protocol is supposed to ensure the authentic communication of a message `M` from `A` to `B`. It then counts as an attack, if `B` finishes the protocol believing that `A` has sent message `M` for him; this includes the case that `A` has meant the message for somebody else (as in the example attack) or it is somebody else than `A` sending this message, even somebody honest. Additionally, in contrast to weak authentication, the standard strong authentication includes also a freshness aspect that we discuss later.

The problem of the second-attempt protocol could be described as follows. Whenever the server produces an encryption $\{|KAB|\}sk(A,s)$ then this indicates to `A` that the key has been produced by the server `s` for use between `A` and some other agent that is not mentioned in that

message. Similarly, the message $\{|KAB|\}sk(B,s)$ only indicates to B that KAB is a shared key for B and somebody else. Since everything outside the encryption can freely be manipulated by the intruder, he can easily confuse the agents and break authentication goals. One may wonder why such a confusion is such a big deal since the intruder apparently does not benefit much from it. For that, consider that the established key may later be used for the transmission of sensitive information like banking transactions or medical data; it is very undesirable that such information are directed to a wrong party because of an authentication problem in the key-exchange.

Actually, this authentication problem can be used to break secrecy – to that end the user may just comment out the two authentication goals and observe that OFMC finds then a secrecy violation.

2.3 Third Attempt

From the previous example we have learned that the encrypted messages by the server should explicitly mention the other agent that the key is meant for, i.e. in the encryption for A the name of B should be mentioned and vice-versa. The exchange then looks as follows:

```
A->s: A,B
s->A: {| KAB,B |}sk(A,s), {| KAB,A |}sk(B,s)
A->B: {| KAB,A |}sk(B,s)
```

This time we get an attack that is not described in the book by Colin Boyd and Anish Mathuria [6] whose development we were following so far:

```
GOAL:
weak_auth
...
ATTACK TRACE:
i -> (s,1): x401,x27
(s,1) -> i: {|KAB(1),x27|}_(sk(x401,s)),{|KAB(1),x401|}_(sk(x27,s))
i -> (x401,1): {|KAB(1),x27|}_(sk(x401,s))

% Reached State:
%
% request(x401,s,pBsKABA,KAB(1),x27,1)
...
% witness(s,x401,pAsKABB,KAB(1),x27)
```

This is indeed a very subtle attack, and one may even argue that this should not be considered an attack. In fact, OFMC has—in the present version—a more sensitive notion of authentication. In contrast to other definitions of authentication, we do not only require that the parties agree on some data, e.g., here the agent $x401$ and the server s on the key $KAB(1)$; rather, we also require that they agree on the which roles they play. In fact, the agent $x401$ believes to play role B here, while the server thinks that $x401$ plays role A . This may be considered less important, mainly because the intruder did not learn the key, and nobody got confused about the names of the partners they are talking with; however, it can in general lead to problems when there is confusion in which role the different participants are acting. (In fact, this is the first version to satisfy the secrecy goal.)

We therefore slightly divert from the development in [6] and change the message format to take this into account. In order to keep consistency with the book, we call this version “3b”. Basically we need to prevent that the messages that the server sends to role A and role B could be confused. There are in fact many ways to do this, e.g., introducing new constants. Instead, we simply mention both the agent A and the agent B in the messages. This is good practice since now both encrypted messages from the server have exactly the same meaning: the first field is the key, the second field is the agent playing role A , and the third field is the agent playing role B :

```

A->s: A,B
# s creates key KAB
s->A: {| KAB,A,B |}sk(A,s), {| KAB,A,B |}sk(B,s)
A->B: {| KAB,A,B |}sk(B,s)

```

Strong Authentication/Replay In this case, we get a violation of the strong authentication aspect of the goal:

Verified for 1 sessions

```

SUMMARY
ATTACK_FOUND
GOAL
  strong_auth
...
ATTACK TRACE
i -> (s,1): x34,x501
(s,1) -> i: {|KAB(1),x34,x501|}_(sk(x34,s)),{|KAB(1),x34,x501|}_(sk(x501,s))
i -> (x501,1): {|KAB(1),x34,x501|}_(sk(x501,s))
i -> (x501,2): {|KAB(1),x34,x501|}_(sk(x501,s))

% Reached State:
%
% request(x501,s,pBsKABA,KAB(1),x34,2)
% request(x501,s,pBsKABA,KAB(1),x34,1)
...

```

The attack trace starts like the previous ones with the intruder sending a message to the server choosing two agent names, now called $x34$ and $x501$. The server answers with the corresponding message mentioning everywhere the agent names. (This produces again the corresponding witness facts.) Now the intruder sends this message to agent $x501$ which is actually as the protocol intends it. $x501$ generates a request term and this request term actually matches the second of the two witness terms; so authentication is fine here (for every request there is a matching witness). Now in the final step the intruder just sends the same message a second time—a *replay*. Note that the receiver is now $(x501, 2)$ while in the previous it was $(x501, 1)$. This means that in both cases it is the same agent $x501$, but it is playing in two different sessions of the protocol. Imagine that the last step of the attack happens much later, say a week, than the first three. That would mean $x501$ accepts a quite old key for communication again. This can be bad for several reasons. First, think of a banking transaction: if one can make the bank perform a transaction several times that was actually issued only once this is clearly a problem. Also it is in many contexts important that a message is recent and not a replay of an old message, e.g., think of electronic stock-market applications. Finally, in many scenarios such as wireless communication, shared keys may be of very limited length, allowing an intruder to find them in a brute-force attack that takes a few hours or days. Establishing a new key frequently can still provide security against such an intruder—but only if the key exchange protocol is protected against replay of course, so the intruder cannot re-introduce an old key that he has broken.

A replay attack (and thus a violation of strong authentication without violating weak authentication) is characterized by two identical request terms with different session numbers, i.e., an agent is made to accept the exact same message more than once.

In fact Lowe's definition of injective agreement [19] is more complicated: it requires basically (in our terminology) that there is an injective mapping from request facts to corresponding witness facts. If we assume, however, that the message being authenticated upon contains at least one part that is supposedly fresh (like the key KAB in this case), then we will never have two times

exactly the same witness fact and two times exactly the same request fact occurs iff there is a replay attack.

Timestamps A very simple and natural way to ensure freshness is the use of timestamps in messages. Assuming we manage to have computers' clocks synchronized up to a few seconds, we can safely require that agents never accept messages bearing a timestamp that is more than a few minutes old. This already ensures that only recent messages are accepted. Additionally, we can prevent any replay even within the validity of the timestamp, if all messages are stored as long as their timestamp is valid and newly incoming messages are checked against this store.

AnB (and OFMC) have no precise model of timestamps; the reason is that talking about concrete timing would require assigning also concrete times to all the normal operations and we would need to formalize also the speed at which the intruder can send messages and similar things.

However, the above sketched methods with timestamps effectively prevent old messages or replays. So if the protocol has these mechanisms in place, one may simply drop the check for replay in our model. In our example that would mean to write the authentication goals as:

```
A weakly authenticates s on KAB,B
B weakly authenticates s on KAB,A
```

With this, the protocol can actually be verified. In fact, looking closely at the attack trace against `strong_auth`, we see that the first line says (with slight grammatical problems):

```
Verified for 1 sessions
```

This means that looking at only one single session, OFMC found no attack. This is not surprising as a replay attack requires at least two sessions of some agent. Using now the weak authentication we see after some time also that it is verified for 2 sessions and so on.

2.4 Fourth Attempt

The described buffering of messages for a limited amount of time can still be an impractical solution in many scenarios, especially when dealing with large amounts of data or a distributed system. (Nonetheless the use of timestamps in electronic transaction is generally a good idea.)

Nonces An alternative way to ensure recentness is the use of *challenge-response* protocols. The challenge is a random number chosen by one party; this number is often called a *nonce*. It abbreviates *number once*, indicating it should be used only one time. The point is that if another party has to include the nonce in a response, then the creator of the nonce can be sure that that response is no older than the nonce it contains. The value of these guarantees of course depends on the cryptographic operations in which the nonce is used.

Since in our case, we want to protect *B* against a replay of the key, we add two steps to the protocol, namely one where *B* generates a nonce *NB* and sends it encrypted with the new shared key *KAB* to *A*, and then *A* has to respond with *NB - 1* encrypted with *KAB*. (The subtraction of 1 is so that the response is actually a different message than the challenge.) The protocol looks as follows:

```
...
Number NB;
Function sk,pre
Knowledge:
A: A,B,s,sk(A,s),pre;
B: A,B,s,sk(B,s),pre;
s: A,B,s,sk(A,s),sk(B,s),pre
Actions:
A->s: A,B
s->A: {| KAB,A,B |}sk(A,s), {| KAB,A,B |}sk(B,s)
```

```

A->B: {|| KAB ,A ,B ||}sk(B,s)
B->A: {|| NB ||}KAB
A->B: {|| pre(NB) ||}KAB

```

Public Functions To model the function ‘ -1 ’ in our abstract model as simple as possible, we have declared a new function symbol pre and given it to the knowledge of every agent. As a consequence, every agent is able to produce $pre(M)$ for a message M that it knows. We do not model more aspects of arithmetic, because that is not really necessary for this model.

This version has a similar attack as the following famous variant:

Needham-Schroeder Shared Key We have now arrived at a protocol very similar to a classic protocol, the Needham-Schroeder Shared Key (NSSK) protocol [27]. That protocol also had a nonce NA from A that is included in the server’s message for A and here the two encryptions are nested, i.e. the server sends the message for B as part of the encrypted message for A :

```

A->s: A,B,NA
s->A: {|| KAB ,B ,NA , {|| KAB ,A ||}sk(B,s) ||}sk(A,s)
A->B: {|| KAB ,A ||}sk(B,s)
B->A: {|| NB ||}KAB
A->B: {|| pre(NB) ||}KAB

```

Denning-Sacco attack on NSSK Both our fourth protocol and the NSSK are vulnerable for very similar attacks, first reported by Denning and Sacco [9]. For NSSK we obtain:

```

SUMMARY
  ATTACK_FOUND
GOAL
  strong_auth
...
ATTACK TRACE
i -> (s,1): i.x701.x206
(s,1) -> i: {||KAB(1).x701.x206.{||KAB(1).i||}_(sk(x701.s))||}_(sk(i.s))
i -> (x701,1): {||KAB(1).i||}_(sk(x701.s))
(x701,1) -> i: {||NB(2)||}_KAB(1)
i -> (x701,1): {||pre(NB(2))||}_KAB(1)
i -> (x701,2): {||KAB(1).i||}_(sk(x701.s))
(x701,2) -> i: {||NB(4)||}_KAB(1)
i -> (x701,2): {||pre(NB(4))||}_KAB(1)

```

The Intruder Acting Under His Real Name This is again a replay attack. As in the previous attacks, we begin with the intruder sending a message to the server s . Here for the first time, we see that the intruder chose a concrete name as a sender: his own name. The reason is that this particular attack only works if the intruder can decrypt the outermost encryption of the reply by the server, which is with the key $sk(A, s)$. The intruder does not know any shared key of an honest agent with the server, but he knows his own shared key with the server: $sk(i, s)$. So for the concrete choice $A = i$, he is actually able to decrypt the answer from the server.

The reader may wonder where it is specified that the intruder knows $sk(i, s)$. It is actually specified both by the knowledge of role A and role B , since both roles can be played by the intruder:

In general, for the initial knowledge specification $A : m_1, \dots, m_n$ (where A is a variable), then the intruder obtains for his initial knowledge all messages m_1, \dots, m_n where all occurrences of A are substituted by i .

The first 5 steps of the attack trace are in fact a perfectly normal protocol run: the intruder acts just like an honest agent would behave in role A . The variables $x701$ and $x206$ are again choices of the intruder, namely of the agent playing role B and the value of the nonce NA , that do not matter for the attack. The actual attack now happens in the last three steps. Here the intruder talks to a second session of the agent $x701$ (in role B) using the old message from the server and then responding to the challenge from $x701$. Note that $x701$ actually generates a fresh nonce $NB(4)$ for this second session.

Meaning of the Attack With this attack, the intruder makes an honest B accept an old session key a second time, violating the strong authentication goal between B and the server. In this form, the attack is actually not that interesting because the intruder needs to play under his real name to achieve it, so it is a session key for secure communication between i and B which is not very attractive to attack. The attack becomes more interesting if we think of KAB as a short session key (that can be broken with brute force within some hours) and $sk(A, s)$ and $sk(B, s)$ as long-term keys that have more length and cannot be broken by brute force. In this case, the attack would also work for an honest A because the intruder just needs to replay an old message of step 3 of the protocol for which he has cracked the contained session key.

AnB currently does not have a method to specify the loss of short-term secrets, although this can be done on the IF level. However, the fact that we get a very similar attack by the normal specification (although it is less interesting) is often indicative that there may be other, related, problems.

2.5 Fifth Attempt

Denning and Sacco suggest to rearrange the protocol a bit and to let B start with sending a nonce NB to A , so that the server can include the nonces of both agents in its messages, and thus provide freshness guarantees to both agents. This protocol now looks as follows:

```
B->A: A, B, NB
A->s: A, B, NA, NB
s->A: {| KAB, B, NA |}sk(A, s), {| KAB, A, NB |}sk(B, s)
A->B: {| KAB, A, NB |}sk(B, s)
```

This protocol is considered secure by many (including [6]). However, OFMC still finds an attack! First, we get again the role confusion problem of the 3rd attempt. So the stricter goals of authentication that OFMC is using are still not satisfied. Let us fix that the same way we did before changing the protocol into:

```
B->A: A, B, NB
A->s: A, B, NA, NB
s->A: {| KAB, A, B, NA |}sk(A, s), {| KAB, A, B, NB |}sk(B, s)
A->B: {| KAB, A, B, NB |}sk(B, s)
```

Anyway we *still* get an attack! This time it is a replay attack:

```
GOAL
  strong_auth
  ...
ATTACK TRACE:
(x701,1) -> i: x701,x701,NB(1)
(x701,2) -> i: x701,x701,NB(2)
i -> (s,2): x701,x701,NB(2),NB(1)
(s,2) -> i: {|KAB(3),x701,x701,NB(2)|}_(sk(x701,s)),{|KAB(3),x701,x701,NB(1)|}_(sk(x701,s))
i -> (x701,1): {|KAB(3),x701,x701,NB(1)|}_(sk(x701,s))
i -> (x701,2): {|KAB(3),x701,x701,NB(2)|}_(sk(x701,s))
```

In fact, the attack is more easy to understand if we reorder the messages in there (the slightly confusing ordering is due to partial-order reduction techniques used in OFMC [26]):

```
ATTACK TRACE
(x701,1) -> i: x701,x701,NB(1)
i -> (s,2): x701,x701,NB(2),NB(1)
(s,2) -> i: {|KAB(3),x701,x701,NB(2)|}_(sk(x701,s)),{|KAB(3),x701,x701,NB(1)|}_(sk(x701,s))
i -> (x701,1): {|KAB(3),x701,x701,NB(1)|}_(sk(x701,s))
(x701,2) -> i: x701,x701,NB(2)
i -> (x701,2): {|KAB(3),x701,x701,NB(2)|}_(sk(x701,s))
```

Talking to Oneself In the attack trace, we see a strange thing: the agent $x701$ who starts (playing role B) intends to talk to — $x701$. We see here that if the roles A and B can be instantiated by two agents, this does not exclude $A = B$. Some people have argued that such scenarios should be considered since a user may work on different physical machines and on all machines, the user may have the same long-term keys. Then, when a user (like $x701$ in this example) tries to establish a secure connection between the two machines (using Denning-Sacco in this case) he would instantiate both roles A and B and thus both shared keys with the server are the same, namely $sk(x701, s)$. If such a scenario is possible, i.e. if the protocol does not explicitly require that the logical name of the two endpoints are different, then the above attack is possible. Note here with *logical name* we mean the identity to which the keys are bound. This is usually *not* the concrete IP-address of the machine and could thus be completely independent from addressing mechanisms. We therefore recommend to make protocols even safe for agents “talking to themselves” and interpret attacks as being related to different machines the agent is working on.

2.6 Final Version

A simple way to fix this last attack is to simply add a constraint to the knowledge section:

```
where A != B
```

This prevents all instantiations of the roles where A and B are played by the same agent. This should of course be noted when implementing the protocol: the implementation should always check whether the identity of the other partner claims to be the same agent identifier (that would have the same key).

3 Example: TLS

We now look at a more interesting example both since it is more complex and since it is one of the most widely used protocols in the Internet. Our model is inspired by the one of Paulson [28].

```
1 Protocol: TLS
2 Types: Agent A,B,s;
3           Number NA,NB,Sid,PA,PB,PMS;
4           Function pk,hash,clientK,serverK,prf
5 Knowledge: A: A,pk(A),pk(s),inv(pk(A)),{A,pk(A)}inv(pk(s)),B,
6           hash,clientK,serverK,prf;
7           B: B,pk(B),pk(s),inv(pk(B)),{B,pk(B)}inv(pk(s)),
8           hash,clientK,serverK,prf
9 Actions:
10 A->B: A,NA,Sid,PA
11 B->A: NB,Sid,PB,
12           {B,pk(B)}inv(pk(s))
13 A->B: {A,pk(A)}inv(pk(s)),
```

```

14      {PMS}pk(B),
15      {hash(NB,B,PMS)}inv(pk(A)),
16      {|hash(prf(PMS,NA,NB),A,B,NA,NB,Sid,PA,PB,PMS)|}
17          clientK(NA,NB,prf(PMS,NA,NB))
18 B->A:  {|hash(prf(PMS,NA,NB),A,B,NA,NB,Sid,PA,PB,PMS)|}
19          serverK(NA,NB,prf(PMS,NA,NB))
20 Goals:
21   B authenticates A on prf(PMS,NA,NB)
22   A authenticates B on prf(PMS,NA,NB)
23   prf(PMS,NA,NB) secret between A,B

```

Walkthrough We discuss the messages step by step:

Client hello (line 10) A client A first contacts the server B that she wants to connect to. This includes a fresh nonce NA and session identifier Sid , as well as the security preferences PA . The security preferences cannot really be modeled here, and we replace them with a nonce (to not change the message format).

Server hello (line 11) The server replies with his own nonce NB and his own preferences PB (again represented as a nonce).

Server certificate (line 12) The server sends a certificate of his public key. This is essentially a digital signature by some trusted certificate authority s , signing for B 's public key. Of course, the real certificates may contain more fields, in particular expiry dates, but we do not model that.

In our model, every agent A has a long-term public key $pk(A)$ and a corresponding private key $inv(pk(A))$. Note that pk is a function symbol that we declare similar to sk in previous examples to represent a given (static) key infrastructure. In contrast, inv is a built-in symbol that maps public keys to corresponding private keys. The general rule is that public-key encrypted messages can only be decrypted with the corresponding private key and vice-versa. Encryption with a private key thus means *signing* a message (because only the owner of the private key can have done that). We distinguish asymmetric (public/private-key) encryption from symmetric encryption by using the notation $\{M\}K$ for encryption of message M with key K .

The initial knowledge of role A contains her public and private key as well as a certificate for her public key by the server and the server's public key. The knowledge of B is analogous. They both do not in advance know each other's public keys, modeling that they only learn them through the exchange of the certificates. In order to verify the certificates, they need the public key of the server. As a consequence, in the translation from AnB to IF, the first exchange looks like this on the side of A :

```
send(A,NA,Sid,PA).receive(NB,Sid,PB,{B,PKB}inv(pk(s)))
```

Here, the public key of B is learned by A as PKB from the certificate. A has no means to check $PKB = pk(B)$ as it is supposed to be, although this will always be the case since in this model nobody has the key $inv(pk(s))$, so nobody can forge certificates.

Client certificate (line 13) Similar to the server's certificate. Note this is optional in TLS: if omitted (which is usually the case if the client is a normal web-browser) then the client is not authenticated. The authentication and secrecy goals we state do not hold then. We discuss this interesting case of a unilaterally authenticated TLS channel below.

Client key exchange (line 14) The client generates the *pre-master secret* PMS, which is just another fresh random number. This number is encrypted with the public key of the server.

Certificate verify (line 15) This signature is present iff the client certificate (line 13) is present.

It then authenticates the PMS and links it with the nonce NB and the name of B .

What is signed is actually a *cryptographic hash* of NB, B, PMS . Recall that a cryptographic hash provides a cryptographic check function in the sense that for two random messages M and M' , it is very unlikely that $h(M) = h(M')$ (low chance of collisions); it is difficult to obtain M from knowing only $h(M)$ (hard to invert); and for given M (or $h(M)$) it is hard to find M' such that $h(M) = h(M')$ (collision-resistant). We simply model this in AnB again as a new function symbol `hash` and give this function to initial knowledge of all roles, so everybody can compute $h(M)$ for given M ; the hardness of finding collisions and inverses is modeled by the absence of intruder rules in the algebraic theory of OFMC.

Client finished (line 16-17) The next message for the first time contains the basis of the shared keys that A and B will obtain. This basis is $K = \text{prf}(PMS, NA, NB)$ where `prf` stands for *perfect random function* and is just another cryptographic hash (like in line 15). This basis K is used to create a message authentication code, i.e. a hash-function with a symmetric key. The original TLS specification tells us to MAC all messages that have been exchanged so far with K . To simplify this a bit, we use just the variables that occur so far. This hash is called the “*Finished*-Message”. It is transmitted encrypted with the client’s shared key $\text{clientK}(NA, NB, K)$ where `clientK` is yet another hash-function.

Server finished (line 18-19) The server B answers with the same finished message encrypted with his shared key $\text{serverK}(NA, NB, K)$ where `serverK` is the last of the hash-functions we introduce. Note that both A and B can compute both client and server keys. The distinction is made so that messages from A to B can not be mistaken as messages from B to A .

3.1 Unilateral TLS

The most commonly used form of TLS is without the optional client authentication (i.e. lines 13 and 15 in the above AnB specification), because the user does not have a certificate. These connections have strictly weaker security guarantees: the server cannot be sure about the identity of the client he is talking with (while the client can, thanks to the server’s certificate). Still, this client and server have a secure connection in the sense that confidentiality and integrity are preserved. We may think of a client acting under a pseudonym and being authenticated with respect to that pseudonym as proposed in [25]. We will come back to this when discussing channels in Section 15.

3.2 Diffie-Hellman

Diffie-Hellman [10] is a cryptographic primitive that can be regarded as the beginning of public key cryptography (as it pre-dates RSA); it has also an interactive/protocol aspect and we shall give an short introduction, since it is usually a good idea to create fresh keys using Diffie-Hellman.

To go slightly into cryptography, the idea is to pick first a large prime number p (which is also publicly known) and make computations in \mathbb{Z}_p^* , that is the group of numbers $\{1, \dots, p-1\}$ with multiplication modulo p . For instance let $p = 7$ (to have a small prime number of the example) and write \equiv_p for equivalence modulo p , then $3^3 \equiv_7 3 \cdot 3 \cdot 3 \equiv_7 9 \cdot 3 \equiv_7 2 \cdot 3 \equiv_7 6$. Thus in all multiplications (and thus exponentiations) we “stay” within \mathbb{Z}_p . Note also that any of the intermediate results can be taken modulo p (so also intermediate results do not get larger than $p-1$). Next, we fix also a *generator* $g \in \mathbb{Z}_p$: a generator is an element so that every for every $z \in \mathbb{Z}_p^*$, there is an $x \in \mathbb{Z}_p^*$ such that $g^x \equiv_p z$. Both g and p are fixed and publicly known. An important property is that given x , it is easy to compute $g^x(\text{mod}p)$, but the opposite direction it is believed to be a hard problem, i.e., the best known algorithm that given z as input finds an x such that $g^x \equiv_p z$ is exponential (in the size of p).

Now Diffie-Hellman between two agents A and B is essentially the following: both A and B generate random numbers X and Y , respectively, and make the following exchange:

```
A->B: exp(g,X)
B->A: exp(g,Y)
```

where for simplicity we omit the fixed prime modulus p and write \exp for modular exponentiation. After this exchange, the numbers X and Y are still secret to the participant who created them. Now A can exponentiate the value $\exp(g, Y)$ from B with her secret X , i.e., $\exp(\exp(g, Y), X)$. Similarly B can exponentiate the value $\exp(g, X)$ from A with his secret Y , i.e., $\exp(\exp(g, X), Y)$. By the laws of exponentiation (that also hold modulo p) that is the same: $\exp(\exp(g, Y), X) = \exp(\exp(g, X), Y)$. Thus they have a secret shared key that can only be constructed by the creators of X and Y .

So in a way, we have created here a secret “out of thin air”, i.e., without A or B having to have any prior relationship with each other and without the help of a trusted third party. Of course, that is not entirely true, we need to be precise here: we do not have a secret between A and B necessarily, but between whoever created X and Y . Any attacker could have generated his own secret X and send $\exp(g, X)$, claiming to be A (or similar impersonating B). There is no relationship here between A and $\exp(g, X)$ or between B and $\exp(g, Y)$. However we can make such a relationship using any approach to *authenticate* the exchange. Here is a simple Diffie-Hellman based protocol that uses digital signatures to authenticate the exchange:

```
Protocol: DH
# A simple protocol based on Diffie-Hellman

Types: Agent A,B;
          Number X,Y,g,MsgA,MsgB;
          Function pk;

Knowledge: A: A,B,pk(A),pk(B),inv(pk(A)),g;
              B: A,B,pk(A),pk(B),inv(pk(B)),g
where A!=B

Actions:

A -> B: {A,B, exp(g,X)}inv(pk(A))
B -> A: {A,B, exp(g,Y)}inv(pk(B))
A -> B: {|A,B,MsgA|}exp(exp(g,X),Y)
B -> A: {|B,A,MsgB|}exp(exp(g,X),Y)

Goals:

A authenticates B on exp(exp(g,X),Y),MsgB
B authenticates A on exp(exp(g,X),Y),MsgA
exp(exp(g,X),Y) secret between A,B
MsgA secret between A,B
MsgB secret between A,B
```

Here we use the Diffie-Hellman key to exchange some “payload” messages $MsgA$ and $MsgB$. It is actually left implicit how A and B create the key, i.e., that A rather has to construct $\exp(\exp(g, Y)X)$ to be able to encrypt here payload message to B and decrypt B 's payload message to her. OFMC does this automatically, since the property that $\exp(\exp(g, X), Y) = \exp(\exp(g, Y), X)$ is understood by the compiler (and the protocol analysis). How this is actually computed is discussed in Section 10.

What was actually revolutionary when Diffie-Hellman was introduced becomes clear if we compare it to the key exchange protocols that can be built with symmetric cryptography only (like the first example of this section). Suppose two parties who already have a shared secret want to update it to a new secret. We could have a simple update protocol like this:

```
A->B: {|update,K'|}K
```

where K is the current shared secret key of A and B and K' is a fresh key that A just created. One may make a more complicated protocol to also make key confirmation etc., but let us focus on this. Basically the only choice to exchange a new secret like K' is to already have a shared secret K . Diffie-Hellman requires a bit less: we do not have to have a secret, it suffices to be able to authenticate each other, while confidentiality is not important: the intruder may well see the exchanged exponents. This is why we consider this actually like the beginning of public key cryptography: one can regard X and Y as private keys and $\exp(g, X)$ and $\exp(g, Y)$ as public keys. It suffices to authentically distribute the public keys, then we can build any secure channels out of it.

Another advantage is built in: both parties contributed to the Diffie-Hellman key, so both have an implicit guarantee of freshness. (However, this can also be achieved with symmetric cryptography.) Finally Diffie-Hellman has perfect forward secrecy: consider the following Diffie-Hellman-based key update protocol where A and B currently shared a symmetric key K (may be constructed earlier using Diffie-Hellman) and they update as follows:

```
A -> B: {| update ,A ,exp(g ,X) |}K
B -> A: {| update ,B ,exp(g ,Y) |}K
```

and suppose the agents from now on use the resulting $K' = \exp(\exp(g, X), Y)$ as new shared key. Suppose now, the key K becomes known to the intruder (e.g., by a brute force attack or some mistake) *after* this exchange. Then still, he cannot construct K' (except by a separate brute force attack). In contrast, in the conventional key update before, if the intruder has logged all the traffic and finds out just one key, all the following keys fall like domino stones.

In a very similar way, consider our key exchange protocol with the trusted third party from the beginning of this section. If the intruder manages to hack the trusted third party at any time and obtain the long-term keys of some agents, then he can decrypt every conversation where he has logged the key-exchange messages. That is not the case if we exchange the protocol to use Diffie-Hellman for creating the shared keys.

For all these reasons, it is usually a good idea to use Diffie-Hellman to construct shared keys – and why \exp has “the full support” of AnB/OFMC.

4 The Syntax of AnB

After the examples we have seen so far, we want to first define precisely the syntax of Alice and Bob notation, and then define formally, what it actually means – the semantics. To that end, let us consider a full specification, the Needham-Schroeder Shared-Key protocol, which is one of the protocols we have “walked into” in our development in Section 2.4.

```
Protocol: NSSK # Needham Schroeder Shared Key
Types: Agent A,B,s;
       Number NA,NB;
       Symmetric_key KAB;
       Function sk,pre
Knowledge: A: A,B,s,sk(A,s),pre;
            B: A,B,s,sk(B,s),pre;
            s: A,B,s,sk(A,s),sk(B,s),pre
Actions:
A->s: A,B,NA
s->A: {| KAB,B,NA, {| KAB,A |}sk(B,s) |}sk(A,s)
A->B: {| KAB,A |}sk(B,s)
B->A: {| NB |}KAB
A->B: {| pre(NB) |}KAB
Goals:
A authenticates s on KAB,B
B authenticates s on KAB,A
```

```
KAB secret between A,B,s
```

Obviously the first item is to give the protocol a name and comments can be made using the # sign. The we have to declare the types of all identifiers that we use in the protocol (actually OFMC allows omitting type declarations, but this may lead to unexpected results). As possible types we have:

- **Agent**: the type of all agents (participants) in the protocol, i.e., who can send or receive messages.
- **Number**: random nonces and the like.
- **Symmetric_key**: similar to nonces (in fact either can be used as keys in encryptions).
- **Public_key**: for declaring a public key, i.e. asymmetric encryption); if P is a public key then $\text{inv}(P)$ is the corresponding private key, so there is no type for specifying private keys.
- **Function**: modeling a function (of arbitrary arguments). These functions will by default *not* be available to the intruder, so one can, like in the example, use it to define a key-infrastructure.

Note that the types **Symmetric_key** and **Public_key** are only used for *freshly created keys*, i.e., that are newly created during the protocol run. For long-term keys (that exist before a particular protocol run), one must use functions like **sk** in the example.

Variables are identifiers that start with an uppercase character, while constants and functions start with a lower-case character.

We can now build *terms*, or *messages*, using the functions and constants symbols that have been declared including the following *built-in* functions:

- For any term p (that represents a public key) the corresponding private key is $\text{inv}(p)$.
- Public key encryption is denoted $\{m\}p$ where p is the public key and m is the message encrypted. (In order to decrypt one needs the corresponding private key $\text{inv}(p)$.) The same symbol is also used for signatures: $\{m\}\text{inv}(p)$ is a signature on the message m with private key $\text{inv}(p)$. (To verify the signature, one needs the corresponding public key p .)
- Symmetric encryption is denoted $\{|m|\}k$ where k can be an arbitrary term representing the encryption key, and m is the encrypted message. In the text, where we are not limited by ASCII, we may rather write more nicely $\{m\}_k$.
- A concatenation of messages is simply written with commas between the messages, e.g., KAB,B. Note that “,” is thus an infix operator, and it binds less than any other, e.g. $\{c\}a,b$ is equivalent to $(\{c\}a),b$.⁶ Also, the operator is right-associative, i.e., A,B,C is understood like A,(B,C).
- The function $\text{exp}(b,e)$ represents modular exponentiation (where the modulus is not explicitly written) of basis b with exponent e . This is for use in Diffie-Hellman based protocols, and we cover it more precisely below in Section 3.2.
- There is also a partial support for bitwise exclusive or (XOR), but it is actually not recommended to use XOR: such a function should only occur in the implementation of cryptographic primitives, but not in the protocols itself, and often it is indicative of poor protocol design.⁷

⁶To express a key that is a concatenation, one uses parentheses, e.g., $\{c\}(a,b)$. However, usually one would not have such a raw concatenation, but some key derivation function like in the TLS example.

⁷There are some exceptions. First, distance-bounding protocols, but here so many other assumptions are needed that standard protocol models and verification tools cannot help. Second, protocols for devices with very low power consumption; but these cannot protect against any intruder with at least modest computational power. Third, one-time pads; however these are best modeled by a confidential channel.

Note that there are no hash or MAC functions built in; one simply declares them as user-defined functions and makes them public (i.e., add to the knowledge of every protocol role).

Further, there are no decryption functions; this is because they are not used to construct any messages (but only to extract information from messages) and thus, they are not part of any term in AnB.

In fact, we later see that even in modeling the actions of honest agents and intruder, we do not need function symbols for decryption.

The next part of the AnB specification is the initial knowledge. The knowledge should be specified for any *role* of the protocol, i.e., for every variable and constant of type agent (except for ones that never send or receive any messages like the server s in the TLS example).

The knowledge is a list of messages that the role initially knows. All variables in the knowledge **must** be of type **Agent**. All variables that do not occur in the knowledge section represent messages freshly created during the protocol run by the agent who first uses them. The knowledge of a role must be sufficient to *execute* the protocol (as defined below).

Intuitively, the requirement of “executable” means that at any step of the protocol agents must have sufficient knowledge (from initial knowledge and received messages) to construct the next message they are supposed to send. In fact, this is at the very core of understanding the *meaning* of AnB.

The core of an AnB specification is the action section, where we have steps of the form $A \rightarrow B : m$ for two roles A and B and a message m . The roles can be constants or variables of type agent. Later in Section 15 we will also introduce here a channel notation to allow for communication channels with some built-in security properties. The channel $A \rightarrow B$ in contrast means that there are no guarantees on the security of the communication medium, e.g. the Internet or a wireless transmission where potentially an attacker could listen, send, or even intercept messages.

Finally in the goals section we can specify three kinds of goals (and their formal meaning will be defined in Section 9):

- **m secret between A, B, C** where m is a message and A, B, C is a list of protocol roles that are allowed to know the message.
- **B weakly authenticates A on m** where m is a message and A and B are roles of the protocol. The intuitive meaning is that when B finishes the protocol, then A has at least started the protocol and agrees with B on each others name and the message m .
- **B authenticates A on m** . Like the **weakly** variant, but additionally we require that there is no replay, i.e., that B did not complete more sessions than A started.

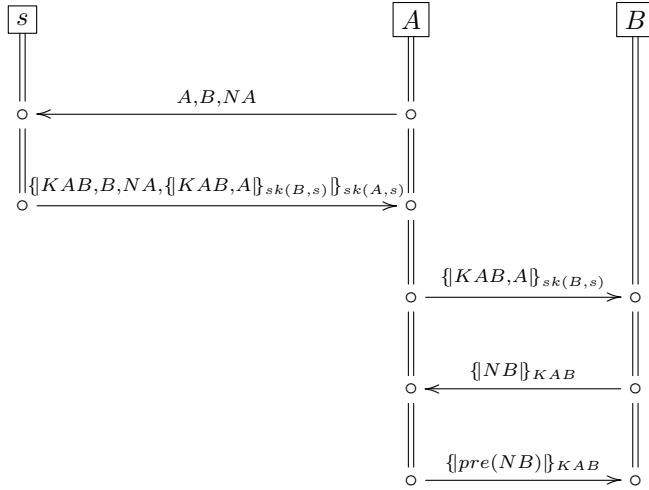
The “direction” (from A to B) that is implicit in the authentication goals may be confusing at first. Suppose m is a message generated by A in the protocol and then somehow transmitted (even indirectly) to B . Then it is clear that authentication is a goal from B ’s point of view, i.e., B wants to be sure that this message indeed comes from A . In contrast, A does not have any guarantees from this goal: it does guarantee neither that the message will only reach B (that would have to be a secrecy goal) nor that the message will at all reach B (since the intruder can disrupt communication). Consider again the example of the authentication goals for key KAB in the NSSK example: here we declare the goal that both agents A and B authenticate the server s on the key, who is actually the one generating the key. We could however also formulate a goal that A and B should authenticate each other on the key. It is instructive to experiment with such goals and try to understand the attacks that result in some cases.

5 From Alice and Bob to Strands – Intuition

Even though Alice and Bob notation seems fairly simple and intuitive, it is surprisingly difficult to define it precisely. Essentially it describes how honest agents would behave, i.e., those that faithfully execute the protocol without trying to cheat. The intruder in contrast can do anything he likes, limited only by his knowledge and his cryptographic abilities. It turns out that, in order to define the behavior of the honest agents and the notion of executability, we also need a kind of intruder model: no agent can for instance perform encryptions or decryptions to which they do not possess the necessary keys. In other words, in order to execute a protocol, one should not need to break the cryptography.

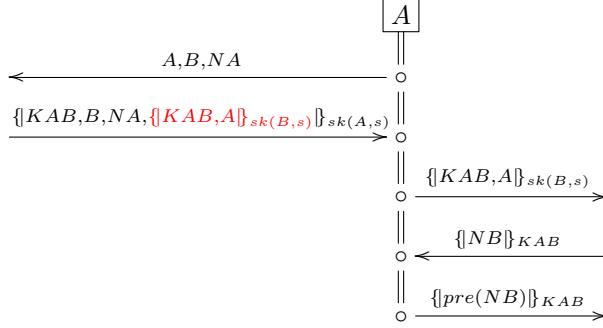
We will now define the meaning of Alice and Bob notation by first a translation to strands – one for each role – that describes the behavior of each honest agent playing in this role. Together with a model of the intruder, this will then give rise to a *state transition system*, where each state is a simple model of the protocol world, and both the intruder and the honest agents can perform actions that lead to new states. The security goals will then be checks on states whether any goal has been violated (e.g., has the intruder found out a secret that he was not supposed to see); we call a state where a goal is violated, an *attack state*. The security of a protocol means then: no *reachable* state is an attack state. In fact there will be, in general, infinitely many states so that an exhaustive search directly will not work, but we will see some methods that can often “cope” with the infinity.

A first step in the translation is to write the action section of a protocol as a message sequence chart. For the NSSK example, we obtain the following MSC:



The main idea is now to split this into several strands, one for each role. For the NSSK example, for the role A this would be simply the messages that A is involved in (with some red

highlighting explained below):



Suppose we choose concrete values for the variables A and B – some agent names – and for the variable NA – a fresh nonce, then this can be regarded as a fully specified program how this agent should behave:

1. It first sends out the message A, B, NA
2. It then waits for receiving any message that *matches* the *pattern* of the first incoming arrow. That is, it would have to be encrypted with the key $sk(A, s)$ (the shared key of A and s), so A can decrypt this. The result of the decryption is a list of messages:
 - The first field is an arbitrary key KAB . That means, this variable is *bound* by whatever is received here – any value is accepted.
 - The second field must be the name of B sent in step 1.
 - The third field must be the random number NA sent in step 1.
 - The fourth one, highlighted in red is actually a bit strange, let us get back to that a little later.

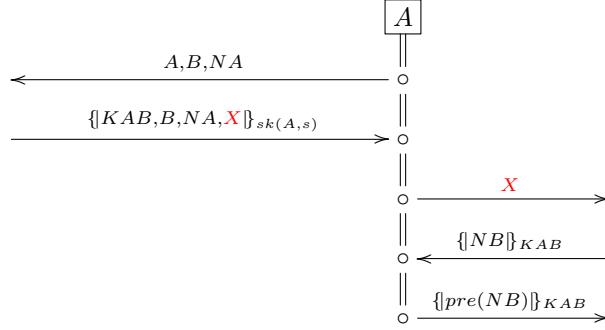
If any of the checks on the fields (or the decryption) fails, then this message is not accepted. It could then either be that the agent terminates the session, or keeps on waiting for another message that meets the expectations.

3. If everything went fine in step 2, the agent proceeds to step 3, sending out the “red” message.
4. The agent waits now to receive any message that is encrypted with the novel key KAB (that was learned in step 2). Again, the value NB is *bound* here – the agent accepts any value and remembers it as NB .
5. In the final step the agent applies the pre function to NB (representing $NB - 1$), encrypts it with KAB and sends it.

More generally, we are thus first instantiating with concrete terms all variables of type agent and the values that the agent in question freshly generates (note that A generates only NA , while KAB and NB are generated by others). For incoming messages, we only accept concrete terms that match the pattern we have, and this instantiates the remaining variables of that message. We will define this precisely below as a transition system.

But there is a strange thing about the highlighted red part in the example: this is a message encrypted with the shared key of B and s that agent A does not know. Therefore, there is no way for A to decrypt this message and check that it is really encrypted with that key, that it contains the same value KAB that was received in the first part of the message, and that it also contains the name A . The red part represents a couple of checks that A actually cannot perform! In fact, A should be willing to accept any message in this position. Of course, A sends out as step

3 whatever she received here. Thus, we get an accurate representation by replacing the red part with another variable X in both the step 2 and step 3:



Thus, the translation of AnB to strands must take care of

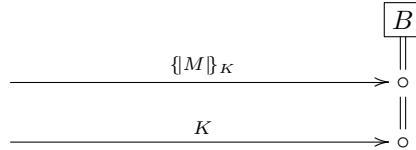
- what agents can actually check on incoming messages and
- how they can generate outgoing messages.

It may seem fairly simple: we just have to figure out subterms (like the red one in the above example) that an agent cannot decrypt and check, and that has therefore to be replaced by a variable. The following example shows that it is not so easy:

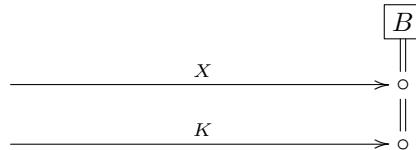
```

A->B: {| M |}K
# some time passes
A->B: K
  
```

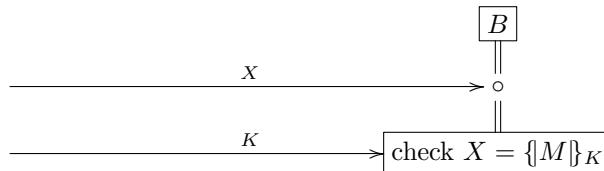
where K is a freshly generated key and M is some message that A wants to send to B but so that B cannot open it yet. At a later point she can now send K and open the message. How would the strand for B look like? The simple solution



is not correct, since B does not know K yet, and thus cannot check that he really received an encrypted message. Replacing it simply with X :



is also not correct, because we now lost any the crucial relation between the two received messages: in particular B would now accept any message X and then any message K (even if it does not decrypt the first one). For this one we would have to add a check in form of an equation:



In fact, this check would bind the variable M to whatever had been encrypted in the message X . Note that a protocol may well now contain steps that involve M , since B has learned it by decryption.

To precisely define the translation from Alice and Bob to strands, there is a sequence of increasingly more complicated papers, see for instance [16]. Few of these papers deal with another complication: that operators may have algebraic properties like

$$\exp(\exp(A, X), Y) = \exp(\exp(A, Y), X)$$

which is required for the translation of any Diffie-Hellman based protocols (without this property, these protocols would not be executable).

An interesting question was therefore: how can we define the meaning of Alice and Bob notation for an *arbitrary* given algebraic theory? This generalization of the question to arbitrary properties had a surprising effect: it lead to a much simpler definition than any of the previous specialized efforts [23, 7, 1].

This concludes for now our first overview of the Alice and Bob notation and its meaning. We now need to first look a bit at terms, algebraic properties, and intruder deduction, and then get back to Alice and Bob and strands.

6 Term Algebra and All That

We introduce now the basic notions of terms and algebras that are needed in the rest of this tutorial. For a more thorough account we suggest [3, 17].

6.1 Signatures

The basic logical term *signature* (not to be confused with for instance digital signatures) means the symbols from which we can build terms, and it is thus sometimes also called an alphabet:

Definition 1 (Signature). *A signature Σ is a set of function symbols where every function symbol has an arity (number of arguments). We write f/n for a function symbol f of arity n . Constants are a special case of function symbols with arity 0.*

Example 1. $\Sigma_{\mathbb{Z}} = \{\text{add}/2, \text{mult}/2, \text{minus}/1, \text{zero}/0, \text{one}/0\}$. While the names of these function symbols somehow suggest a meaning, there is none attached to them.

Table 1 shows the function symbols we usually need in protocol verification, together with their intuitive meaning and whether or not they are public, i.e., whether the intruder can apply them. For instance, the function $\text{inv}(\cdot)$ is not public because it shall later be used to assign a private key to a given public key; if $\text{inv}(\cdot)$ were public, then the intruder would know all private keys that he knows the public key of. Let us denote with Σ_p in the following the subset of symbols from Σ that are declared *public*.

We also use the “mixfix” notation for encryption from AnB, i.e., we write $\{m\}_k$ instead of introducing a prefix function symbol like $\text{crypt}(k, m)$. We allow the user to declare their own constants and function symbols, so we cannot really give one fixed signature Σ – it all depends on what the user specifies in an AnB file. Similarly, we will in other parts of the document need to introduce new function symbols. That is why all the following definitions are given for an *arbitrary* Σ , i.e., the definitions are *parameterized* over Σ .

Definition 2 (Terms). *Let Σ be a signature and $V = \{X, Y, Z, \dots\}$ be a set of variable symbols disjoint from Σ . Define $\mathcal{T}_{\Sigma}(V)$ to be the set of terms (over Σ and V) as follows:*

- All variables of V are terms.
- If t_1, \dots, t_n are terms and $f/n \in \Sigma$, then also $f(t_1, \dots, t_n)$ is a term.

Terms without variables are called ground terms. We write \mathcal{T}_{Σ} for the set of all ground terms.

Symbol	Arity	Meaning (informal)	Public
i	0	name of the intruder	yes
$\text{inv}(\cdot)$	1	private key of a given public key	no
$\{\cdot\}$	2	asymmetric encryption	yes
$\{\cdot\beta\}$	2	symmetric encryption	yes
$\langle \cdot, \cdot \rangle$	2	pairing/concatenation	yes
$\text{exp}(\cdot, \cdot)$	2	exponentiation modulo fixed prime p	yes
a, b, c, \dots	0	User-defined constants	User-def.
$f(\cdot)$	\star	User-defined function symbol f	User-def.

Table 1: Standard function symbols for protocol verification.

Example 2. $\mathcal{T}_{\Sigma_{\mathbb{Z}}}(\{X, Y\}) = \{X, Y, \text{zero}, \text{one}, \text{minus}(X), \text{minus}(Y), \text{minus}(\text{zero}), \text{minus}(\text{one}), \text{add}(X, X), \dots\}$. Again, these terms have no meaning attached to them yet.

A central concept is that of a substitution: it expresses that some variables should be substituted (replaced) by some terms:

Definition 3 (Substitution). Let X_1, \dots, X_n be variables and let s_1, \dots, s_n be terms. A substitution σ is written as $\sigma = [X_1 \mapsto s_1, \dots, X_n \mapsto s_n]$ and means the following function from terms to terms:

$$\sigma(t) = \begin{cases} s_i & \text{if } t = X_i \text{ for any } i \in \{1, \dots, n\} \\ f(\sigma(t_1), \dots, \sigma(t_n)) & \text{if } t = f(t_1, \dots, t_n) \\ t & \text{otherwise} \end{cases}$$

Thus, a substitution replaces every X_i with s_i , leaves all other variables untouched, and for a composed term $f(t_1, \dots, t_n)$, the substitution is recursively applied to the subterms t_i .

Example 3. For $\sigma = [X \mapsto f(Z), Y \mapsto Z]$ we have $\sigma(g(Z, Y, f(X))) = g(Z, Z, f(f(Z)))$.

6.2 Algebra

While signatures and terms are pure syntax, an algebra gives symbols and terms a *meaning*. The idea is that we have to have a “universe” (or “domain”) in which we want to interpret terms, and then interpret every function symbol as a function in that universe:

Definition 4 (Σ -Algebra). Given a signature Σ , a Σ -Algebra \mathcal{A} consists of

- a non-empty set U , called the Universe, and
- for every $f/n \in \Sigma$ a function $f^{\mathcal{A}} : U^n \rightarrow U$.

For a ground term $t \in \mathcal{T}_{\Sigma}$ and a Σ -Algebra \mathcal{A} , we denote with $t^{\mathcal{A}}$ the meaning of t in \mathcal{A} which is:

$$f(t_1, \dots, t_n)^{\mathcal{A}} = f^{\mathcal{A}}(t_1^{\mathcal{A}}, \dots, t_n^{\mathcal{A}})$$

Example 4. One possible algebra \mathcal{Z} for our example signature $\Sigma_{\mathbb{Z}} = \{\text{add}/2, \text{mult}/2, \text{minus}/1, \text{zero}/0, \text{one}/0\}$ is to interpret all terms as integer numbers:

- Universe: $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$
- $\text{add}^{\mathcal{Z}}(a, b) = a + b$
- $\text{mult}^{\mathcal{Z}}(a, b) = a \cdot b$
- $\text{minus}^{\mathcal{Z}}(a) = -a$
- $\text{zero}^{\mathcal{Z}} = 0$

- $\text{one}^{\mathcal{Z}} = 1$

We can evaluate any ground term in \mathcal{Z} , for instance:

$$\text{add}(\text{mult}(\text{one}, \text{minus}(\text{one})), \text{one})^{\mathcal{Z}} = 0 .$$

Another choice is the algebra \mathcal{B} that interprets terms as Booleans:

- Universe: $\mathbb{B} = \{\text{true}, \text{false}\}$
- $\text{add}^{\mathcal{B}}(a, b) = a \vee b$ (logical or)
- $\text{mult}^{\mathcal{B}}(a, b) = a \wedge b$ (logical and)
- $\text{minus}^{\mathcal{B}}(a) = \neg a$ (logical not)
- $\text{zero}^{\mathcal{B}} = \text{false}$
- $\text{one}^{\mathcal{B}} = \text{true}$

It holds that

$$\text{add}(\text{mult}(\text{one}, \text{minus}(\text{one})), \text{one})^{\mathcal{B}} = \text{true} .$$

Compare the result with the one for \mathcal{Z} ; it is normal that in two different algebras we get different results for terms.

A possible algebra \mathcal{C} for the signature of Table 1:

- Universe: \mathbb{B}^* (all bit-strings)
- $\{\cdot\}_k^{\mathcal{C}}$ is AES 128 bit with MAC (returning an error message if k is not 128 bit long).
- ...

Observe that the algebras \mathcal{Z} and \mathcal{B} of this example share some properties, for instance both add and mult are associative and commutative and mult is distributive over add . More formally, let us write $s \approx_{\mathcal{A}} t$ if two (ground) terms s and t are equal in algebra \mathcal{A} , i.e., if $s^{\mathcal{A}} = t^{\mathcal{A}}$. Then we have for every ground terms s, t, u that

$$\text{mult}(s, \text{add}(t, u)) \approx_{\mathcal{Z}} \text{add}(\text{mult}(s, t), \text{mult}(s, u))$$

and the same holds for $\approx_{\mathcal{B}}$.

An important idea of mathematical logic is that, instead of studying a concrete algebra like \mathcal{Z} and \mathcal{B} and see what properties they have, we could rather take the opposite direction and start with a set of equations like distributivity above and ask: what is the class of algebras in which this property holds? This allows to *abstract* from the concrete algebra and prove statements that hold in *every* algebra that satisfies a number of equations. In that spirit, in cryptographers have studied cryptographic operators on an abstract level, i.e., considering only what (algebraic) properties the cryptographic building-blocks need to satisfy, allowing for the study of entire classes of cryptographic implementations [21].

6.3 The Free Algebra

In logic programming and theoretical computer science, one extreme form of algebra plays a central role: if we do *not assume any algebraic properties*. This is in some sense the most abstract and also somehow the most easy algebra to work with. (Hence the popularity in computer science...) It is often called the initial or free algebra because it is “freely generated” by its terms. Even though we have already used the notation \mathcal{T}_{Σ} for the set of all ground terms, we will “overload” it and also use it to denote the free algebra:

Definition 5 (Free Algebra). *Given a signature Σ , the free algebra \mathcal{T}_{Σ} is defined as follows:*

- The universe is the set of all ground terms \mathcal{T}_Σ .
- For every $f/n \in \Sigma$, define $f^{\mathcal{T}_\Sigma}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$.

Lemma 1. For all ground terms s and t , we have $t^{\mathcal{T}_\Sigma} = t$ and $s \approx_{\mathcal{T}_\Sigma} t$ iff $s = t$. ⁸

Thus, in the free algebra, the meaning of a term t in \mathcal{T}_Σ is just the term t itself and two terms are equal in the free algebra, if they are syntactically equal (there are no syntactic equations). In other words, terms do not have a deeper meaning, they just are what they syntactically are. (No wonder this resonates so well with computer science...)

In Example 4, we have sketched how a algebra \mathcal{C} for the symbols of Table 1 could look like. This algebra would be a concrete cryptographic interpretation where every message is a bit string and most function symbols are actual cryptographic algorithms. We will now go for the other extreme and interpret messages in the most abstract way – in the free algebra.

This has some interesting consequences. In \mathcal{C} , we may for instance interpret a function $h(\cdot)$ by a cryptographic hash function and there will be collisions, i.e., different messages n and m such that $h(n) \approx_{\mathcal{C}} h(m)$. In contrast, in the free algebra it holds that $h(n) \approx_{\mathcal{T}_\Sigma} h(m)$ iff $h(n) = h(m)$ iff $n = m$. So the free algebra models hash functions as *collision free* – which is actually absurd in reality, since no function $h : A \rightarrow B$ with infinite A and finite B can be injective.

To see that it makes sense to interpret terms in the free algebra anyway, consider that the example of the collision is a (necessary) imperfection of the concrete hash function used in \mathcal{C} , and a different concrete hash function may have collisions for different pairs of inputs. The free algebra is thus abstracting from the (undesirable) properties of the concrete implementation, and we will thus arrive at an intruder who can only perform attacks that work in *any* implementation of the cryptographic functions.

One could thus say that we model an intruder who is oblivious to the actual cryptography, and will not attempt any attacks on the crypto-level. Thus, the security results that we formally prove in such a model by default tell us only about the security against an intruder who does not perform crypto-analytic attacks. We come back to this issue when we define the intruder deduction relation below, i.e., the intruder’s ability to analyze terms.

6.4 ★ Quotient Algebra

A section title in blue and marked with a ★ signals an advanced topic for the readers who would like to know more. We always give a short *executive summary paragraph* that suffices for following the rest of the tutorial.

Summary 1. For some advanced protocols and primitives we simply need a few algebraic properties in the model. We then define a set of equations E (like $s + t = t + s$) and define an algebra where the equations of E hold but that is “as close as possible” to the free algebra, i.e., two terms are equal iff that is a consequence of E .

Protocols that use Diffie-Hellman are only executable if the model supports at least the following property:

$$\exp(\exp(g, X), Y) \approx \exp(\exp(g, Y), X)$$

This is because A knows X and $\exp(g, Y)$ and can thus construct the term on the right, while B knows Y and $\exp(g, X)$ and can thus construct the term of the left. That they arrive at the same shared secret thus requires the equality to hold.

Definition 6. Let E be a set of equations $s \approx t$ where s and t are terms with variables. Let \approx_E be the smallest congruence relation that includes all ground instances of E :

- (Equations of E .) If $s \approx t$ is an equation of E , and σ is a substitution that maps every variable of s and t to a ground term, then $\sigma(s) \approx_E \sigma(t)$ holds.

⁸“iff” abbreviates “if and only if”

- (Reflexivity.) $s \approx_E s$ for every term s .
- (Transitivity.) If $s \approx_E t$ and $t \approx_E u$ then also $s \approx_E u$.
- (Symmetry.) If $s \approx_E t$ then also $t \approx_E s$.
- (Structure.) If $s_1 \approx_E t_1, \dots, s_n \approx_E t_n$ and f has arity n , then $f(s_1, \dots, s_n) \approx_E f(t_1, \dots, t_n)$.

The equivalence class $[t]_E$ of a term t is the set of equivalent terms in \approx_E :

$$[t]_E = \{s \mid s \approx_E t\}.$$

In general, a relation that is reflexive, transitive, and symmetric is called an *equivalence relation*, and if it satisfies also the (Structure.) condition, it is called a *congruence relation*. We have defined \approx_E to be the *smallest* relation that satisfies the above properties, i.e., $s \not\approx_E t$ unless it follows from the properties that $s \approx_E t$. Therefore \approx_E is “the closest we can get to the free algebra” while still respecting the equations of E .

Example 5. Let E be just the equation $\exp(\exp(B, X), Y) \approx \exp(\exp(B, Y), X)$. Then

$$\begin{aligned} [\exp(\exp(\exp(a, b), c), d)]_E = \{ & \exp(\exp(\exp(a, b), c), d)), \\ & \exp(\exp(\exp(a, b), d), c)), \\ & \exp(\exp(\exp(a, c), b), d)), \\ & \exp(\exp(\exp(a, c), d), b)), \\ & \exp(\exp(\exp(a, d), b), c)), \\ & \exp(\exp(\exp(a, d), c), b)) \} \end{aligned}$$

The remaining question is how to define an algebra \mathcal{A} such that $\approx_{\mathcal{A}}$ is the relation \approx_E : the free algebra is too abstract (no equations hold), several other algebras are too concrete (too many equations hold). Recall that the free algebra is constructed by using the set of all terms as the universe, and every term is interpreted by itself. The idea is now to interpret every term t by its equivalence class $[t]_E$ – automatically if $s \approx_E t$ we then have $[s]_E = [t]_E$, thus two terms are interpreted as equal iff they are equal according to \approx_E . This means that the universe is now a set of sets of terms, namely for every term, the equivalence class of that term is an element of the universe:

Definition 7 (Quotient Algebra). Given a signature Σ , and a set of equations E , the quotient algebra $\mathcal{T}_{\Sigma/\approx_E}$ is defined as follows:

- The universe is the set $\{[t]_E \mid t \in \mathcal{T}_\Sigma\}$ of equivalence classes of ground terms.
- For every $f/n \in \Sigma$, define $f^{\mathcal{T}_{\Sigma/\approx_E}}([t_1]_E, \dots, [t_n]_E) = [f(t_1, \dots, t_n)]_E$.

The second item of this definition requires some more explanation: it defines how a function symbol f can be applied to n elements of the universe. Since these elements are equivalence classes, we can write them in the form $[t_i]_E$ where the t_i are some element of the equivalence class. The definition does not tell how this t_i should be picked from the equivalence class, e.g., what happens if we choose some other terms s_1, \dots, s_n with the property that $s_i \approx_E t_i$? The answer is that this does not matter for the result since $f(t_1, \dots, t_n) \approx_E f(s_1, \dots, s_n)$ since \approx_E is a congruence relation. (Thus, also $[f(t_1, \dots, t_n)]_E = [f(s_1, \dots, s_n)]_E$.) This definition tells us to arbitrarily pick any “representatives” t_i from the given equivalence classes, and this is not ill-defined, since result is independent of that choice.

As a closing remark, most algorithms that work in \approx_E are more complicated than their counterparts for the free algebra. In general \approx_E is not even a decidable relation, i.e., for some E there is no algorithm that can – for every pair of terms s and t as input – correctly answer whether or not $s \approx_E t$.

7 The Dolev-Yao Intruder Model

One of the most cited papers of protocol verification is one by Danny Dolev and Andrew Yao [11] because they suggested a simple but comprehensive intruder model that has become the de-facto standard for modeling an intruder if one does not consider the cryptographic level. The original paper considers only public-key encryption as a cryptographic primitive, but it is common to treat other primitives in the same spirit. Here are the key points:

- Every user has a public/private key pair.
- Every user knows the public key of every other user.
- The intruder is also a user with his own key pair.
- The intruder can decrypt only messages that are “meant” for him, i.e., that are encrypted with his public key.
- The intruder controls the network: he can read messages, block them, divert them to a different recipient, and insert new messages.

It may seem excessive to assume the intruder controls the entire network (e.g., the entire Internet and also the Intranet of a company). However, this expresses simply that we do not rely the network to offer any protection, and we should not if a message may travel over any point that could be insecure. We will later see how to integrate a notion of channels on which the intruder has no, or limited, control.

The insecure network is the classical view of secure communication: Alice wants to send to Bob some messages, they are honest people, but between them is a hostile world that tries to read, manipulate, or even forge messages between them. One may think of a spy novel where Alice is a secret agent operating in a foreign country and Bob is the home base of the secret service. Dolev and Yao make an important change to this classical view: that the participants of a protocol are not necessarily honest people (who stick to the rules, in particular the protocol). One may think for instance of an e-Banking protocol where Alice is a customer and Bob is a bank: it should not be a requirement of this protocol that all customers are honest, some clients may be trying to cheat. Maybe even a bank may be dishonest, e.g., due to dishonest employees trying to manipulate transactions. In fact, several of the most surprising attacks involve a dishonest participant (while the protocol is secure when considering only honest agents and the intruder can only control the network) e.g. [18]. Thus, we recommend to consider by default that every role of the protocol may be played by a dishonest agent. In some cases like the keyserver example from before, we see that the protocol is (trivially) broken if the keyserver is dishonest. We have thus explicitly made the keyserver a trusted party by using a constant s that cannot be instantiated by the intruder, while all other roles of the protocol (where we have variables) can be instantiated by the intruder i. ⁹

7.1 Intruder Deduction

At the core of the Dolev-Yao intruder model is the question of the cryptographic abilities. While Dolev and Yao only consider public-key cryptography, the general idea is that the intruder “knows” all cryptographic algorithms – these algorithms should not be considered a secret themselves, but only the secret keys. This is sometimes called *Kerckhoff’s principle*. It is particularly important if you have dishonest participants, because they obviously need to know the algorithms. Note that only functions like $\text{inv}(\cdot)$ are exempt from this, because they do not represent a cryptographic operation, but a function in our model that assigns to every public key its corresponding private key.

⁹One may wonder what happens if there is more than one intruder: in the worst case, they all collaborate, therefore we see that as a special case of one intruder and the dishonest agents are bots under the intruder’s control. It is a bit of a simplification that there is only one dishonest agent called i, but as long as no inequalities on agent names are required, this is sound.

Another key idea of Dolev-Yao is now: the intruder can use all (public) cryptographic operations – but that is it, there is no other possibility to analyze messages, like cryptanalysis. Thus we model an intruder who has access to a *Crypto API*, i.e., a library of encryption and decryption algorithms, but all he ever does cryptographically are function calls to that library. All the security results we prove in this model thus are against an intruder who is oblivious to cryptography and may no longer hold against an attacker with crypto-analytic abilities.

One could of course instead perform “cryptographic security proofs”, i.e., prove that a protocol is secure in an algebra like \mathcal{C} in Example 4. This gets extremely complex as one has to then define bounds on the intruder’s resources, consider probabilities (since the intruder may make guesses) and use assumptions of the hardness of certain mathematical problems like prime factorization. If we think of a developer in industry whose goal is to design complex distributed systems then security against such a full-fledged cryptographic model may be infeasible: we should not require that all developers of systems that use cryptography are also cryptographers themselves.

Our suggestion is to clearly distribute “responsibility” and a reasonable distribution can be: the goal of a crypto API should be, roughly speaking, that the intruder cannot derive any message that he cannot obtain from an API call. This is not trivial to formalize and prove, but there are several results that show the soundness of a Dolev-Yao model with respect to a concrete cryptographic implementation, e.g. [5].

We formalize now the cryptographic abilities of the intruder according in the style of Dolev and Yao for our set of operators from Table 1. This is a relation of the form $M \vdash m$ where M is a finite set of messages, called the intruder knowledge, and m is a message that is derivable from that knowledge:

Definition 8. We define \vdash as the least relation that satisfies the following rules:

$$\begin{array}{c} \frac{}{M \vdash m} \text{ if } m \in M \text{ (Axiom)} \quad \frac{M \vdash m_1 \dots M \vdash m_n \text{ if } f/n \in \Sigma_p}{M \vdash f(m_1, \dots, m_n)} \text{ (Compose)} \\ \frac{M \vdash \langle m_1, m_2 \rangle}{M \vdash m_i} \text{ (Proj}_i\text{)} \quad \frac{M \vdash \{m\}_k \quad M \vdash k}{M \vdash m} \text{ (DecSym)} \\ \frac{M \vdash \{m\}_k \quad M \vdash \text{inv}(k)}{M \vdash m} \text{ (DecAsym)} \quad \frac{M \vdash \{m\}_{\text{inv}(k)}}{M \vdash m} \text{ (OpenSig)} \\ \frac{M \vdash s \text{ if } s \approx_E t}{M \vdash t} \text{ (Algebra)} \end{array}$$

(Axiom) This rule just says that the intruder can derive any term m that is already in his knowledge M .

(Compose) This rule says that for any public function symbol f of n arguments he can apply f to any terms m_1, \dots, m_n that he can already derive.

(Proj_i) If the intruder knows the pair $\langle m_1, m_2 \rangle$, then he also knows its components m_1 and m_2 .

(DecSym) If the intruder knows a symmetrically encrypted message $\{m\}_k$ and also knows the key k , then he can derive m .

(DecAsym) Similarly for asymmetric encryption, only here he has to know the private key $\text{inv}(k)$.

(OpenSig) If the intruder knows a signature $\{m\}_{\text{inv}(k)}$, then he also knows the signed message m .¹⁰

(Algebra) For the case that one wants to analyze a protocol under some algebraic properties E (e.g. for exponentiation), this rule closes the deduction under \approx_E .

¹⁰We assume here a signature scheme where the message m being signed is actually included in clear-text and the actual signature is applied only to a hash of that message. To obtain m , the intruder does not need to know the public key, but only for verifying signatures. Verifying signatures however is not a message deduction problem (he does not learn a new message from that).

The fact that the intruder cannot do anything else than these rules is captured by saying that \vdash is the *least* relation satisfying the rules: if $M \vdash t$ does not follow from these rules (by any number of steps), then $M \not\vdash t$.

Example 6. Let $M = \{k_1, \{m_1\}_{k_1}, m_2, \{m_3\}_{k_2}\}$. Then for instance we have:

- $M \vdash m_1$
- $M \not\vdash m_3$
- $M \vdash \{\langle m_1, m_2 \rangle\}_{k_1}$
- ...

We can actually write derivations as a proof tree where the leaves of the tree are (Axiom) steps and the root of the tree is the result we want to prove:

$$\frac{\frac{\frac{\overline{M \vdash \{m_1\}_{k_1}} \text{ (Axiom)} \quad \overline{M \vdash k_1} \text{ (Axiom)}}{M \vdash m_1} \text{ (DecSym)} \quad \frac{\overline{M \vdash m_2} \text{ (Axiom)}}{M \vdash \langle m_1, m_2 \rangle} \text{ (Compose)}}{M \vdash \langle m_1, m_2 \rangle} \text{ (Compose)}}{M \vdash \{\langle m_1, m_2 \rangle\}_{k_1}}$$

Example 7. As an example for reasoning with algebraic properties consider again the property $\exp(\exp(B, X), Y) \approx \exp(\exp(B, Y), X)$.

Let the intruder knowledge be $M = \{x, \{b, \exp(g, y)\}_k, k, m\}$. Observe that x, y are constants (lower-case letters) i.e., concrete exponents that the intruder and another participant have randomly chosen.

We show that the intruder can derive from M for instance the message $\{m\}_{\exp(\exp(g, x), y)}$:

$$\frac{\frac{\frac{\overline{M \vdash \{b, \exp(g, y)\}_k} \text{ (Axiom)} \quad \overline{M \vdash k} \text{ (Axiom)}}{M \vdash \langle b, \exp(g, y) \rangle} \text{ (DecSym)}}{\frac{\frac{\overline{M \vdash \exp(g, y)}}{M \vdash \exp(\exp(g, y), x)} \text{ (Proj}_2\text{)}}{\frac{\overline{M \vdash x} \text{ (Axiom)}}{M \vdash \exp(\exp(g, y), x)}} \text{ (Compose)}}}{\frac{\frac{\overline{M \vdash \exp(\exp(g, x), y)} \text{ (Algebra)}}{\overline{M \vdash m} \text{ (Axiom)}} \text{ (Compose)}}{M \vdash \{m\}_{\exp(\exp(g, x), y)}}}$$

7.2 Automating Dolev-Yao

Let us quickly think about implementing an algorithm that, given M and m as input should tell us whether $M \vdash m$, and consider the free algebra (i.e., $E = \emptyset$, i.e., we can omit the (x Algebra) rule). A problem is that there are infinitely many things the intruder can do, and thus there are infinitely many proofs. How can we limit the search for a proof of $M \vdash m$, so that the algorithm does not run into an infinite loop?

The idea is to solve first an easier problem: let $M \vdash_c m$ denote derivations that only use (Axiom) and (Composition), i.e., an intruder who does not analyze any terms. This can be solved by a simple backwards search: if the goal is to derive $t = f(t_1, \dots, t_n)$, then check if t is contained in M , and if not, and if $f \in \Sigma_p$, try to recursively derive every t_i . Otherwise the answer is no. This algorithm terminates since in every recursive call, the terms get smaller.

Next we solve the problem to *analyze* the intruder knowledge: if the intruder knows a message of the form $\{m\}_k$, use the above algorithm for checking $M \vdash_c k$, i.e., whether the key can be obtained by composition steps only. If so, add m to the intruder knowledge M . This modification of M does not change the set of terms that the intruder can derive from M via \vdash , the intruder is just explicitly remembering what he can derive (in this case using (DecSym)). Repeat this for all of the decomposition rules (DecSym), (DecAsym), (Proj $_i$), and (OpenSig) until no new terms can be obtained. This algorithm terminates because it can only add sub-terms of the original intruder knowledge M (and since M is finite, there are only finitely many subterms).

Example 8. The complete analysis of the intruder knowledge M in Example 7, would add the message b and $\exp(g, y)$.

Now that the intruder knowledge is completely analyzed, for any term m to derive it suffices to use only composition steps:

Theorem 1 ([1]). For an analyzed knowledge M , $M \vdash m$ iff $M \vdash_c m$.

This algorithm can also be extended to handle many equational theories E (e.g. the exponentiation example), but in general \approx_E is undecidable, and thus, so is \vdash .

8 Transition Systems

We can now put it all together and define a world of honest agents, an intruder, and an intruder-controlled network. We proceed as follows:

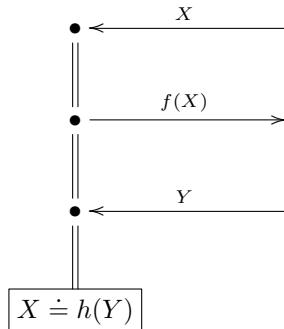
- We start with an AnB description of a protocol. As explained in Section 5 we can first see such a description as a message sequence chart, and then split it into several strands, one for each role of the protocol.
- We have pointed out also in Section 5 there are some cases where this does not yet give an accurate description of the protocol execution, but let us postpone this problem at first and come back to it in section 10.
- We define how to instantiate the protocol for concrete sessions of the protocol, yielding an infinite set of “closed” strands.
- We then define a state transition system where both the honest agents (represented by the strands) and the intruder can make transitions.

Let us start by giving a formal definition of strands:

Definition 9. A strand is a sequence of steps where each step is either

- $\text{Snd}(t)$ for sending a message t .
- $\text{Rcv}(t)$ for receiving a message t .
- $s \doteq t$ for checking whether two terms s and t are equal. (We write \doteq to indicate that this is an operation performed during protocol execution.)
- $\text{Evt}(t)$ generates a special event t (that the intruder cannot see and that we use only for formulating the goals).

For a strand of zero steps we write 0. The graphical notation of strands is straightforward, e.g. the strand $\text{Rcv}(X).\text{Snd}(f(X)).\text{Rcv}(Y).X \doteq h(Y)$ would be represented as



Let $S = S_1.\text{Rcv}(t).S_2$ be a strand, and let X be a variable that occurs in t but not in S_1 . Then we say X is bound in S by the receive step $\text{Rcv}(t)$. We say that all variables that are not bound by such a receive step are free variables of S . We say that a strand is closed if it does not have free variables.

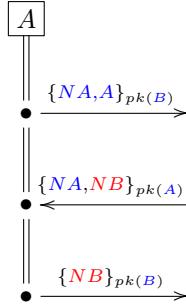
Example 9. Consider the following AnB specification, called Needham-Schroeder Public-Key protocol:

```

Protocol: NSPK
Types: Agent A,B;
       Number NA,NB;
       Function pk
Knowledge: A: A,pk,inv(pk(A)),B;
            B: B,pk,inv(pk(B))
Actions:
A->B: {NA,A}(pk(B))
B->A: {NA,NB,B}(pk(A))
A->B: {NB}(pk(B))
Goals:...

```

Extracting the strand for role A yields (when there is no danger of confusion, we sometimes write a pair $\langle s, t \rangle$ of messages simply as “ s, t ” without the angle brackets):



The variable NB is bound by a receive step, while the blue variables are free.

8.1 ★ Instantiation

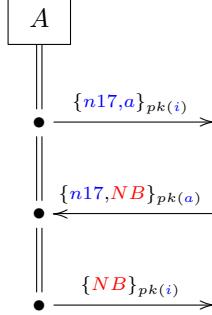
Summary 2. The roles that come out of AnB have free variables that represent either agent names (and can be instantiated with any concrete agent name, including the intruder i), and freshly generated constants (that are instantiated with constants that are not used anywhere else). This gives an (in general infinite) set S_0 of closed strands. Also this induces an initial intruder knowledge M_0 (derived from the knowledge specification in AnB).

We consider the free variables of a role as its *parameters*, i.e., when we see a role as a program, the parameters could be inputs to the program. In fact, recall that in the syntax of AnB we had required that

- only variables of type *Agent* can occur in the initial knowledge of any agent – and they can be instantiated with any concrete agent name; and
- all other variables (that do not occur in the initial knowledge) are freshly created by the agent who first uses them.

Example 10. Continuing Example 9, the variables A and B are agent names and shall be instantiated with agent names, e.g. $\sigma(A) = a, \sigma(B) = i$. The variable NA is freshly created by A since it is not in the initial knowledge and first occurs in a message sent by A . Thus we can

instantiate it with a constant that is unique to this strand – to model a random number generator, e.g., $\sigma(NA) = n17$. Under substitution σ we obtain the following closed strand:



The precise semantics of AnB includes an executability check that can only succeed if all free variables are either agent names of the initial knowledge of that role or freshly created by that role. For instance in the example, if we remove NB from the receive step, but leave it in the following send step, then this protocol is unexecutable (since NB is created by B and thus a priori not known to A).

Thus, the instantiation of agent names and fresh constants will always yield *closed* strands. We can thus create an arbitrary large supply of protocol *sessions*:

Definition 10. Let R_1, \dots, R_k be the roles of a protocol (as strands). For each role R_i let I_{R_i} be a set of substitutions that instantiate the free variables of R_i with agent names and freshly generated constants. Further let K_i be the knowledge of role R_i and A_i be the agent name of role R_i . We define the initial set of closed strands as:

$$S_0 = \bigcup_{i=1}^k \{\sigma(R_i) \mid \sigma \in I_{R_i} \wedge \sigma(A_i) \neq i\}$$

and the initial intruder knowledge as

$$M_0 = \bigcup_{i=1}^k \{\sigma(K_i) \mid \sigma \in I_{R_i} \wedge \sigma(A_i) = i\}.$$

Thus, for every instantiation σ that does not instantiate the role name with the intruder, we create a closed strand as part of S_0 , and otherwise, we give the instantiated knowledge to the intruder.

Example 11. In the previous example, we have the instance $\sigma = [A \mapsto a, B \mapsto i, NA \mapsto na_{17}]$ for role A , producing the already depicted closed strand. Let us have a matching instance for role B : $\sigma' = [A \mapsto a, B \mapsto i, NB \mapsto nb_{18}]$. Since this role B is now played by i , this instance does not produce a strand but gives the intruder knowledge $\{b, pk, inv(pk(i))\}$, i.e., the knowledge that was specified for role B under substitution σ' . Thus the intruder is supplied with all messages he needs to know in order to faithfully execute the protocol.

Note that in general there is no bound on the agent names or on the number of sessions that can execute a protocol. Also between the same set of agents, we can have an unbounded number of sessions.

8.2 States and Transitions

We define now our abstract protocol world by first describing what a state of this world is:

Definition 11 (State). A state consists of three things:

- a set of closed strands representing the honest agents,
- a set of ground messages M that the intruder currently knows,
- and a set E of events that have occurred so far (for later analyzing the security goals).

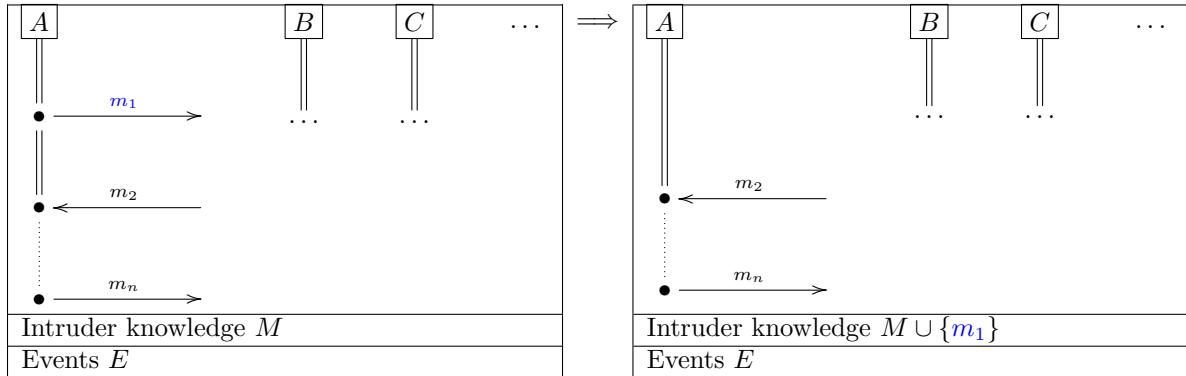
The *initial state* of this world consists of the initial set S_0 of closed strands and the initial intruder knowledge M_0 that were produced by the instantiation. The set E of events is initially empty.

We now define how this world can evolve by giving possible *transitions* from one state to another caused by actions of honest agents and the intruder. In every state there can be several possible transitions, yielding different possible successor states of the world. We are then interested in all states that our world can reach, the *reachable* states. Later we define the security goals by declaring which reachable states we consider an *attack state* and then security is defined by: no attack state is reachable.

We now define the possible transitions, based on the current state and a possible next state:

8.2.1 Transition: Sending

If the current state contains an honest agent whose next step is to send a message m_1 , then a possible next state is obtained by removing the message m_1 and adding it to the intruder knowledge:

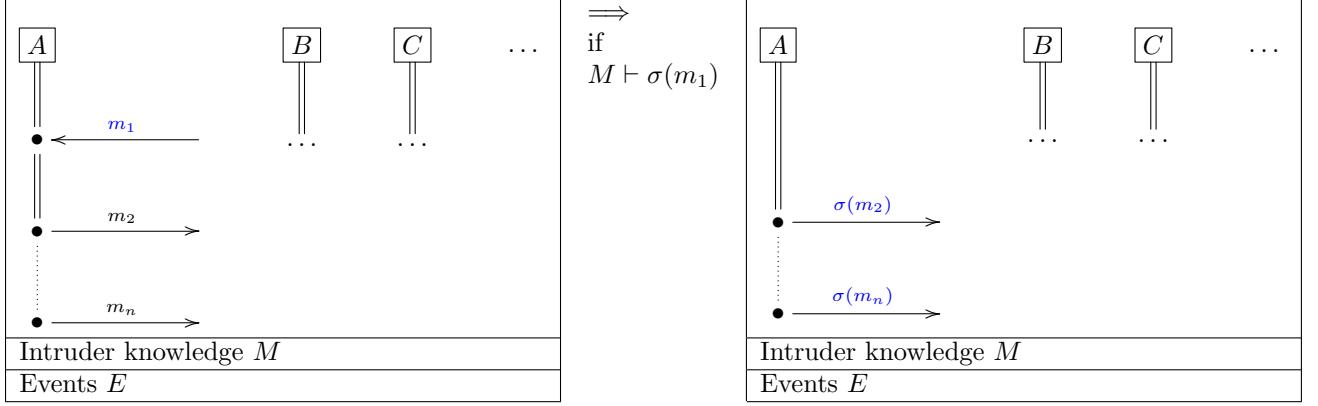


Since the intruder controls the entire network, in this model all messages are simply “received” by the intruder – observe that the strand notation does not even write who the intended recipient is (it does not matter anyway in the presence of this intruder).

8.2.2 Transition: Receiving

The next kind of transition is for a state that contains an honest agent whose next step is to receive a message m_1 . The situation is more complicated, because m_1 may contain variables that are bound by this receive step. The question is whether the intruder can generate any instance of m_1 from his current knowledge M , i.e., whether $M \vdash \sigma(m_1)$ for some substitution σ . Then the

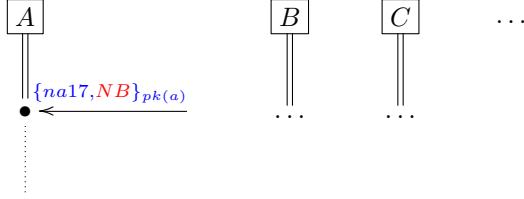
following transition is possible:



The substitution σ is applied to the rest of the strand since all variables that have been bound by the receive step are bound to that value for the rest of the strand. Again this reflects the fact that the intruder controls the network: all messages that an honest agent receives come from the intruder. Observe that the strands do not carry any information about who the claimed sender is, since the intruder could anyway insert any name he knows.

Observe that with this strong intruder all communications are subsumed, e.g., that a message from honest a to be received by honest b : by the send event of a , the intruder learns (and intercepts) the message. If he pleases, he sends it to b in the next step, so the possibility is in the model, but he could choose to send a different (modified) message to b (that matches what b is waiting for) and so on.

Example 12. Consider the state



with intruder knowledge $M = \{a, b, i, pk(a), na5, na17, \{na17, nb3\}_{pk(a)}\}$.

Then there are infinitely many substitutions σ under which the receive step can work, i.e., so that $M \vdash \sigma(\{na17, NB\}_{pk(a)})$:

$$\sigma(NB) =$$

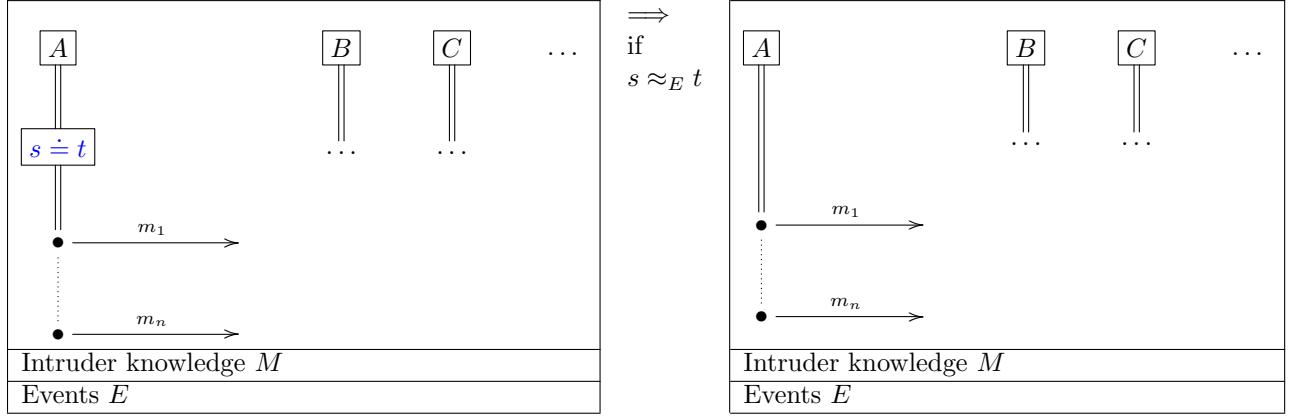
- $nb3$ (using the encrypted message)
- $na5$ or $nb17$ (construct himself)
- $a, b, i, pk(a), \{na17, nb3\}_{pk(a)}, \dots$, i.e. “ill-typed” messages: the intruder can actually use any message he can construct.

This demonstrates that a single honest strand can induce an infinite set of reachable states, since the intruder has an infinite choice of terms to construct.

8.2.3 Transition: Checking

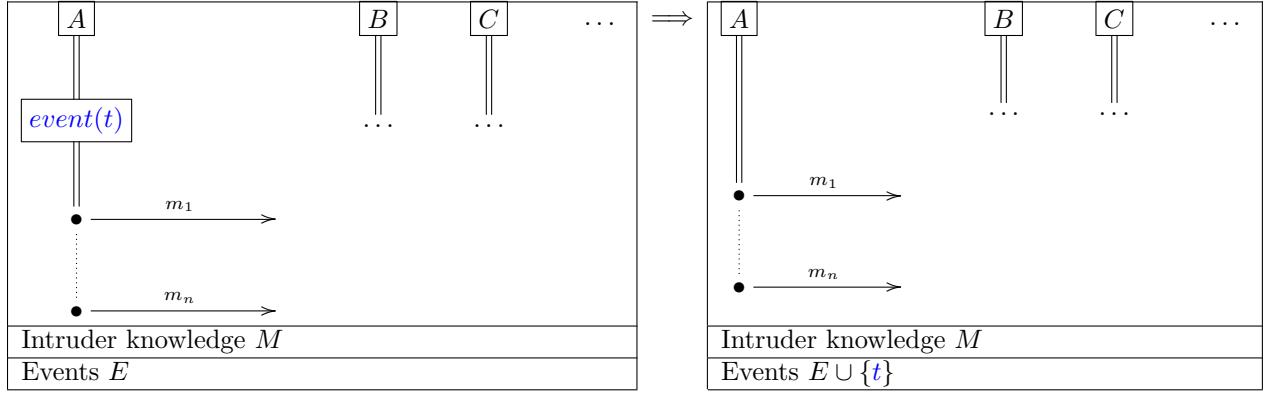
The next kind of transition is pretty easy: when the next step of an agent is an equality $s \doteq t$ then it can proceed only if $s \approx_E t$ (i.e., $s = t$ in the free algebra); otherwise this strand is simply

stuck:

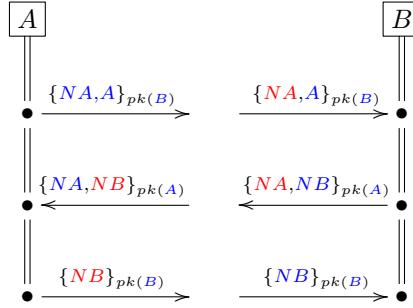


8.2.4 Transition: Events

Finally, if an agent creates an event, this is simply added to the set of events:

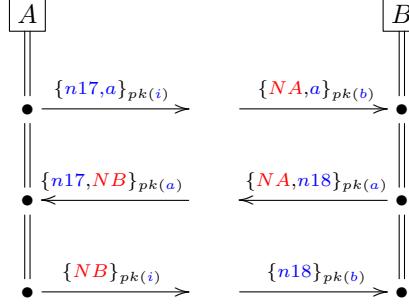


Example 13. Consider again the NSPK protocol from example 9 (where we have marked the free variables blue and the bound variables red):



Let us consider for each role only one instance with an honest player: $\sigma_A = [A \mapsto a, B \mapsto i, NA \mapsto n17]$, so agent a would like to talk to (the dishonest) agent i . Remember this is because our model does not rely on all parties to be necessarily honest. For role B we consider instance $\sigma_B =$

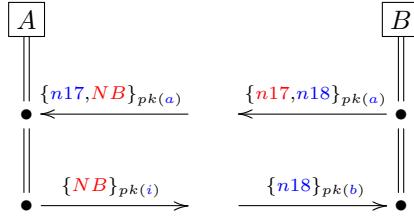
$[B \mapsto b, A \mapsto a, NB \mapsto n18]$:



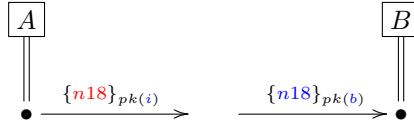
For the initial intruder knowledge we have $M_0 = \{a, b, i, pk(a), pk(b), pk(i), \text{inv}(pk(i))\}$, i.e., he knows the agents, all public keys and his own private key.

Let us look at just one trace, i.e., sequence of transitions, that this world can take:

- First a sends out her first message $\{n17, a\}_{pk(i)}$ that is added to the intruder knowledge M . Note that $M \vdash n17$ since the message is encrypted with the public key of the intruder.
- The intruder has now many choices, he could reply to a , basically making a normal run with her, or he could try to talk to b and impersonate a . Also for that there are many choices, since he can take any term he knows to be NA and compose a message that b is waiting for. He could use the value $n17$ that he just learned, giving this state:



- Now b can send out his reply $\{n17, n18\}_{pk(a)}$ which is again added to the intruder knowledge. Note that the intruder cannot decrypt this message, and thus he cannot learn the secret $n18$ yet. But this message matches what a is waiting for:
- Let the intruder simply forward the message he learned from b to a we have the state:



- When a sends out her next message, $\{n18\}_{pk(i)}$, the intruder learns the secret $n18$.
- The intruder is now also able to complete the run with b , because he can produce $\{n18\}_{pk(b)}$.

In fact, with the standard definition of secrecy and authentication goals – that we more formally discuss next – it is not a secrecy problem that the intruder learned $n17$, because that was a secret meant for i , but it is a violation of secrecy that he learned $n18$ since it was meant for a . So at the next to last step we had already reached an attack state. That a finished her protocol run does not violate any authentication goals, since a did indeed talk with i like she intended, but that b finished his run is a violation of authentication since b believed to be talking with a while a has never intended to talk to b .

This protocol is the “canonical” example why one should consider dishonest agents: it is secure when all participants are honest. This is maybe a reason that after NSPK was published in 1978 it took 18 years until this attack was discovered by Lowe [18].

9 Security Goals

We now define formally secrecy and authentication goals for protocols. There are of course many other interesting goals, such as sender invariance, anonymity, privacy, non-repudiation, and availability. Some of these we will actually discuss later, but they require additional measures and infrastructure, so for now we only focus on the basic goals.

9.1 Secrecy

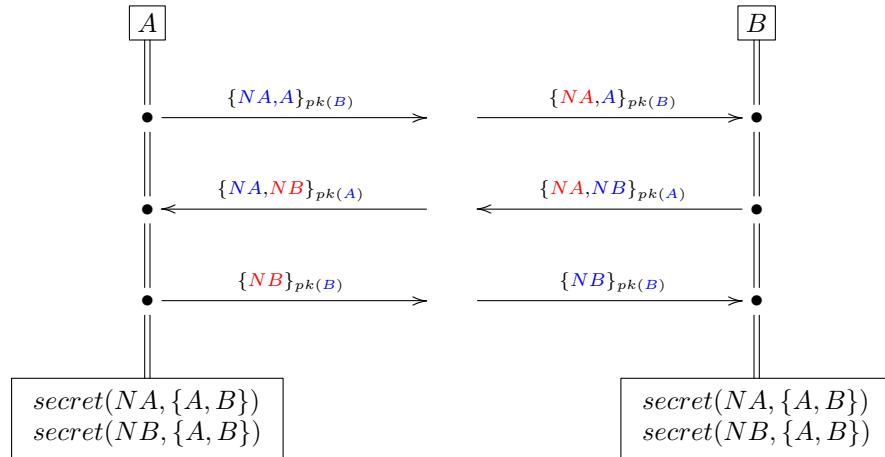
For a protocol like the NSPK example above, one could formulate the following secrecy goals in AnB:

```
NA secret between A, B
NB secret between A, B
```

It is thus important to specify *what* should be a secret, and *who* is cleared to know it. The “who” is important at least if we have dishonest participants, because it is not a violation of secrecy if a dishonest participant learns a secret that is meant for him (like *n17* in the NSPK attack).

A convenient way to define goals is to insert special events into the protocol execution that indicate what an agent “thinks”. For secrecy we use an event $\text{secret}(m, s)$ where m is the secret message and s is a set of agents that may know it. This event is inserted at the *end* because some protocols may have a “candidate secret” that only becomes a secret when the party completed some steps. It is inserted in every role that participates to the secret, because secrecy should also be violated if only one of the parties completed the protocol run.

Example 14. For NSPK with the given secrecy goals we have:



Now we define an attack on secrecy as any state where the intruder finds out a secret he is not supposed to:

Definition 12 (Secrecy). *A state with intruder knowledge M violates secrecy, if it contains an event $\text{secret}(m, s)$ and $M \vdash s$ and $i \notin s$.*

9.2 Authentication

Authentication, also called *agreement*, mainly means that communication partners agree on who they are talking to. It however also entails a property sometimes called *integrity*: that the intruder did not manipulate the exchanged information. This is part of authentication by simply requiring that the agents also agree on the information. Finally, authentication can also entail freshness: that the intruder cannot make one agent agree to something that truly happened, but a long time ago.

For the NSPK protocol we could have the following two goals:

```
B weakly authenticates A on NA
A weakly authenticates B on NB
```

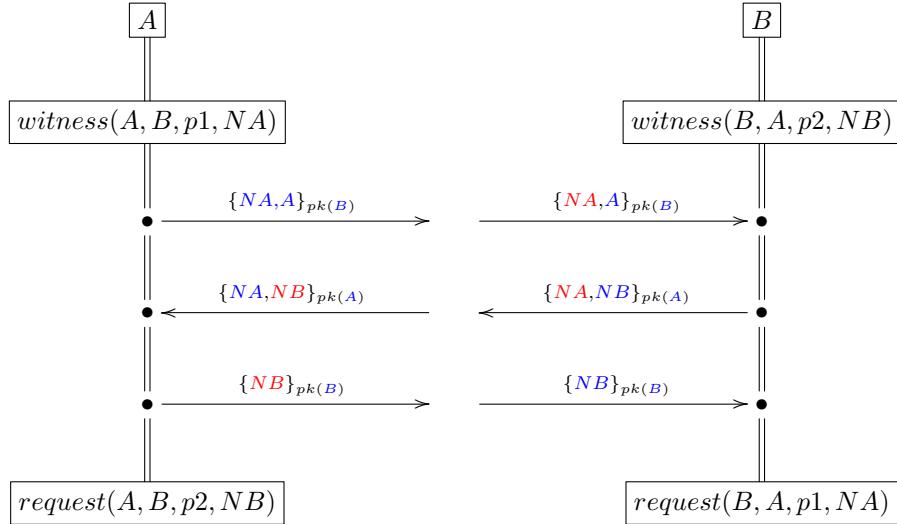
Here “weakly” denotes the weaker variant of authentication – we come to the strong variant below. Authentication goals have a direction as already explained in Section 4: one could see the protocol as authentically (and secretly, but that is another goal) transmit NA from A to B and NB from B to A . We thus need to talk about the intention of one party to send information, and talk about a party apparently receiving information. Like for secrecy goals we define special events for this.

For assign to every authentication goal a unique identifier, so that events of different authentication goals are not confused. Then for any authentication goal $B(\text{weakly})\text{authenticates}A\text{on }M$ with identifier p :

- we insert the event $witness(A, B, p, M)$ into role A at the earliest point where A can compose M .
- we insert the event $request(B, A, p, M, ID)$ at the end of role B . Here ID is a new free variable, representing another fresh identifier to be created; this is for the strong authentication later.

Intuitively $witness(A, B, p, M)$ means that A intends to run the protocol with B and agree on message M ; and $request(B, A, p, M, ID)$ is the counter-part meaning that B now believes to have been talking to A and has received message M in this session.

Example 15. For NSPK with the given authentication goals we have:



Now we define authentication as a state where a request without corresponding witness has occurred, i.e., one party accepts a communication that did not happen that way:

Definition 13 (Weak Authentication). *An attack on weak authentication (aka non-injective agreement [19]) is any state where*

- the event $request(B, A, p, M, ID)$ has occurred for some $A \neq i$ and
- the event $witness(A, B, p, M)$ has not occurred.

We have the side condition $A \neq i$ because in a session with a dishonest participant the agreement is pointless. We could also require $B \neq i$, but that does not change anything since the intruder cannot issue any events, so an event $request(i, \dots)$ cannot occur by construction.

We can see how different aspects are subsumed by this goal: if B believes that A has sent M it is an attack

- (Wrong Sender) if M comes from somebody else than A ;
- (Wrong Receiver) if M comes from A , but was meant for C ; or
- (Wrong Content) if A actually meant to talk to B , but said something different than M .

Finally, we add the aspect of freshness to achieve the (strong) authentication goal. For this we assume however that the message M that we authenticate upon contains something fresh (like NA and NB in the NSPK example are created freshly):

Definition 14 (Authentication). A state violates strong authentication (aka injective agreement [IS]) if it violates weak authentication or:

- two distinct events $\text{request}(B, A, p, M, ID)$ and $\text{request}(B, A, p, M, ID')$ with $ID \neq ID'$ and $A \neq i$.

Thus, we use the freshly created identifier (ID and ID') to distinguish two events that are otherwise equal in every argument. Thus B has accepted two times exactly the same message M to come from A . Since we required that M contains something fresh, no honest A would have sent this M twice, and it is thus a replay by the intruder.

10 ★ The Algebraic AnB Semantics

Summary 3. In Section 5, we had remarked that in general we cannot simply split the message sequence chart into roles, because this often will not reflect how the protocol is really executed, i.e., what incoming messages an agent can accept and how outgoing messages are constructed. Here we give the precise definition for this. A key idea is that a normal protocol execution is based on using only the standard cryptographic operations just like the Dolev-Yao intruder. From this follows almost immediately what an agent can check about a received message and how an agent can compose an outgoing message. However, one must distinguish between the knowledge that an honest agent has about an actual message and how a message supposedly looks like according to the AnB specification. Finally, we can define the semantics of AnB by a translation from AnB to a set of strands, one for each role. We can also generate actual protocol implementations from this.

This section contains several excerpts from [I] where more details and examples can be found.

10.1 Message model

We now define the AnB semantics precisely for any given set Σ of operators, whereof a subset $\Sigma_p \subseteq \Sigma$ is public, and where \approx is an arbitrary congruence relation on terms.

In the previous definition of the Dolev-Yao reduction relation \vdash , we have used some analysis rules like (DecSym) that allow the intruder to analyze a term $\{\{m\}_k\}$ if he knows k and obtain m . An alternative way to model this is to use an explicit decryption operator $\{\cdot\}^{-1}/2 \in \Sigma_p$ with the algebraic property:

$$\{\{\{m\}_k\}\}_k^{-1} \approx m$$

With this property the (DecSym) rule is redundant since for every terms k and m we have:

$$\frac{\begin{array}{c} M \vdash k \quad M \vdash \{\{m\}_k\} \\ \hline M \vdash \{\{\{m\}_k\}\}_k^{-1} \end{array}}{M \vdash m} \text{ (Algebra)}$$

In a similar way, we can replace all analysis rules with explicit decryption functions. One can show that (under some reasonable restrictions) this model with explicit decryption functions is equivalent to our free algebra model with analysis rules. The advantage of the free algebra model

is that it is algorithmically much easier (no algebraic reasoning) while the algebraic model has the advantage that we can handle decryption in a uniform way and more easily talk about steps that honest agents have performed.

We now distinguish two kinds of messages:

- (1) the *protocol messages* that appear in an AnB specification and
- (2) *recipes* that are the messages in the strands the semantics translates to.

It is necessary to make this distinction as the AnB specification reflects the ideal protocol run, while the semantics reflects the actual actions and checks that an honest agent performs in the run of the protocol. For the same reason, we will also distinguish between two kinds of variables:

- *protocol variables* that appear in the AnB specification like A, B, NA and the like and
- *label variables* that we introduce now: $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \dots$

Intuitively, the label variables represent “memory locations” of an honest agent: given the knowledge K_R of role R in an AnB specification, we give every message in K_R one unique label:

Definition 15. A labeled knowledge M is a substitution of the form $M = [\mathcal{X}_1 \mapsto t_1, \dots, \mathcal{X}_n \mapsto t_n]$ where the \mathcal{X}_i are label variables and the t_i are protocol messages. We then say $|M| = n$ is the size of M . The set $\mathcal{T}_{\Sigma_p}(\{\mathcal{X}_1, \dots, \mathcal{X}_n\})$ of terms built from the label variables of M and public operators is called the set of recipes over M . (This is intuitively the set of all terms that an agent with knowledge M could build.) For a recipe t over M , we say the corresponding protocol message is $M(t)$, i.e., applying the substitution M to the recipe, replacing its label variables with protocol terms.

Example 16. During the execution of a Diffie-Hellman based protocol (see sec 3.2), an agent may reach the following knowledge (where X, Y, K, N are variables from the protocol description in AnB):

$$M = [\mathcal{X}_1 \mapsto X, \mathcal{X}_2 \mapsto \{\exp(g, Y)\}_K, \mathcal{X}_3 \mapsto K, \mathcal{X}_4 \mapsto N].$$

The agent is supposed to generate as a next step the message

$$\{\{N\}\}_{\exp(\exp(g, X), Y)}.$$

A recipe for that term is $t = \{\mathcal{X}_4\}_{\exp(\{\mathcal{X}_2\}_{\mathcal{X}_3^{-1}}, \mathcal{X}_1)}$ since:

$$M(t) = \{\{N\}\}_{\exp(\{\{\exp(g, Y)\}_K\}_{\mathcal{X}_3^{-1}}, X)} \approx \{\{N\}\}_{\exp(\exp(g, Y), X)} \approx \{\{N\}\}_{\exp(\exp(g, X), Y)}.$$

This already takes care of the question how agents can generate messages. It remains to define what they can check about messages they receive. The idea is simply: does the reached knowledge M allow for any pair of recipes s, t such that $M(s) \approx M(t)$. That means there are two ways to derive a term that should give the same result if the messages are correct, i.e., follow the AnB specification:

Definition 16. For a knowledge M , we say a formula ϕ is a complete set of checks (for M) if it implies every equation $s \doteq t$ where s and t are recipes over M such that $M(s) \approx M(t)$.

Note that there are trivially always infinitely many equations, since if $s \doteq t$ is one and $f/1 \in \Sigma_p$, then also $f(s) \doteq f(t)$ is one. We are therefore happy with any finite collection ϕ that implies all the others.

Example 17. Suppose according to the protocol an agent shall first receive $h(N)$, but does not know N , so cannot check the received message at first. In a later step N is revealed. The knowledge is then: $M = [\mathcal{X}_1 \mapsto h(N), \mathcal{X}_2 \mapsto N]$. Then $\phi \equiv \mathcal{X}_1 \doteq h(\mathcal{X}_2)$ is a complete set of checks.

The entire translation/semantics from AnB to roles is now as follows:

- First split the message sequence chart from AnB into a set of strands, one for each role. We call it the plain strands.
- Label for each strand the initial knowledge M_0 of that role, which is the messages from the knowledge section labeled with $\mathcal{X}_1, \dots, \mathcal{X}_n$.
- Go step by step through the roles:
 - For sending a protocol message m , find a recipe t over the current knowledge M such that $M(t) \approx m$. If no such recipe exists the protocol is refuted as *unexecutable*. If there is more than one such label, one can choose any: they are all equivalent due to the checks we perform on received messages. The step $\text{Snd}(m)$ is replaced by $\text{Snd}(t)$ for the chosen label.
 - For receiving a protocol message m , extend the current knowledge M by $\mathcal{X}_k \mapsto m$ for a new label variable \mathcal{X}_k (that does not occur in M). Find a complete set ϕ of checks for the augmented M . Replace the receive step $\text{Rcv}(m)$ by $\text{Rcv}(\mathcal{X}_k)$ followed by the equations ϕ .

Note that this now contains explicit decryption functions that our standard model from the previous sections does not support. It is however easy to get rid of them: For instance consider the strand $\text{Rcv}(\mathcal{X}_1).[\{s\}]_{(}^{-1}, \mathcal{X}_1) \doteq t.S$ for some terms s, t and a rest strand S . Here we can extract the substitution $\sigma = [\mathcal{X}_1 \mapsto \{s\}_{(}, t)]$ (otherwise the decryption would not work), and apply σ to the strand, yielding $\text{Rcv}(\{s\}_{(}, t)).\sigma(S)$. In this way all decryption functions can be turned into pattern matching.

Part II

Automated Verification

11 Introduction

It would be great to have a general verification procedure for computer programs. Such a procedure would receive as input a program P (choose your favorite programming language¹¹) and a *specification* what the program should compute. A specification should in some way describe a function from inputs to desired outputs of the program. For instance a program for sorting integers should as input receive a list of integers and as output return a *permutation* of the input list that is in ascending order. In general, a specification give the programmer even more freedom and specify a set S of functions and it is fine if the program computes one of the functions of S . We do not discuss here how a language or logic for describing S could look like, because it will be irrelevant for our point. The question that we want to solve is the following:

Definition 17. *Given a set S of computable functions, then an S -verifier is an algorithm that gets a program P as input and returns yes if P computes a function in S , and no otherwise.*

The following theorem tells us that we cannot even conceive such an algorithm for any S – with two trivial exceptions.

Theorem 2 (Rice). *Let S be any set of computable functions, except for the emptyset and the set of all computable functions. Then there is no S -verifier.*

It is important to keep this principle limitation in mind, because it appears in similar shape again and again, and one can save a lot of time if one recognizes earlier that the problem one tries to solve is actually undecidable. Many questions of logic and mathematics fall into this: it would be nice to have procedures to tell us whether a claimed statement is actually true (i.e., prove that it is a theorem) or not (and give a counter-example).

The principle limitation does not mean, however, that one cannot solve practically relevant parts of the problem, in particular

- identifying restrictions under which the problem becomes decidable, i.e., deciding a *fragment* of the problem;
- procedures that on some inputs give the answer *Inconclusive* (instead of yes or no) or do not terminate. In particular
 - Procedures may be focused on finding counter-examples (attacks, in our case) and be inconclusive or non-terminating on correct inputs.
 - Procedures may be focused on proving correctness and use some over-approximation. They may then fail to find a correctness proof or even present a counter-example that could be a *false positive*, i.e., a counter-example that arises from the over-approximation.

11.1 The Sources of Infinity

The reasons for undecidability in verification is that some aspect of the described system is infinite. Infinity does not necessarily lead to undecidability, for instance the set of even integers is infinite but obviously decidable. We therefore would like to first distinguish several aspects of infinity that arise from the security protocol model we have introduced in the first part and see how it relates to decidability. In fact that restriction of some aspects yields decision procedures that are the most successful ones in practice.

We will in general see our security protocol models as *tree* of states:

¹¹We just have to require that the language is Turing complete, which holds for all standard programming languages like C/C++, Java, Haskell, ...

- we have an initial state that is the root node, and
- we can make state transitions from one state to another that gives a *child* relation between nodes, i.e., if one can reach from state S in one transition a set S' , then S' is a child of S .
- The security goals are described as a check on states, i.e., we should later check if any node of the tree violates this check.

In general this tree has several aspects of infinity:

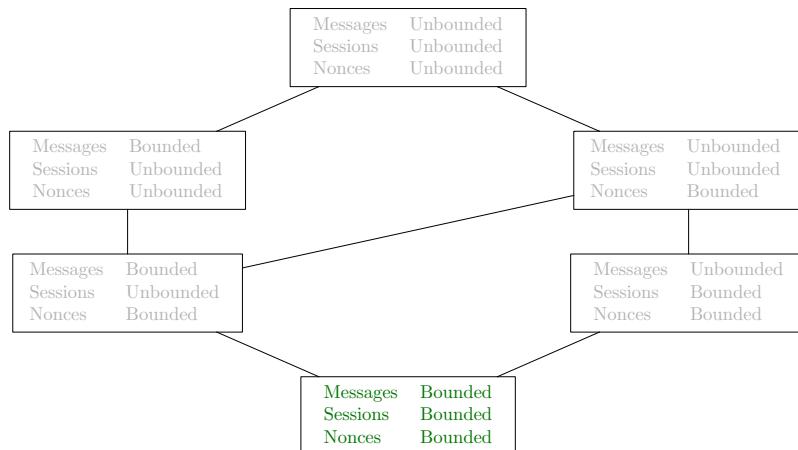
Unbounded Messages Recall the transitions for an honest agent receiving a message m (where m may contain variables). If the current intruder knowledge is M , then for every substitution σ such that $M \vdash \sigma(m)$, we can have a transition. In general this is infinite, i.e., we have a tree that can be *infinitely branching*.

Unbounded Sessions By default, the set of strands for honest agents that we start with is infinite. That is because there we do not want to limit how many people can use the protocol in parallel, and how many sessions between the same people can be open at the same time. An infinite set of sessions (no matter how represented) means both infinite branching of the tree and infinite depth (i.e., infinitely long paths).

Unbounded Nonces Agents can in any session generate fresh nonces, so in an unbounded number of sessions, also the number of nonces may be unbounded. It may seem that this automatically follows from infinite sessions, but we later want to ask what happens if we have unbounded sessions but without fresh nonces.

Algebraic Properties Algebraic properties alone can lead to undecidable problems, we want to limit this here, and just mention that the algorithms we present here can be at least be extended to work with some standard algebraic properties like Diffie-Hellman.

Leaving out algebraic properties and having unbounded nonces only with unbounded sessions, we get a lattice of six possible variants how we could restrict the “infinities” of the problem, from restricting everything to restricting nothing:



Here we have noted the most restricted model already in green, because restricting everything gives a finite tree and then a decision procedure obviously exists.

11.2 * An Undecidability Result

We show that if protocol security *were* decidable, then we could construct a decision procedure for a classic problem: Post’s Correspondence Problem (PCP). Since PCP is already known to be undecidable, so must be protocol security. See also: the updated decidability lattice at the end of this section.

To show the undecidability of a problem, i.e., that it is *impossible* to construct an algorithm that decides that problem, it is often helpful to relate it to other undecidable problems for which undecidability has already been proved. Some people like to use the classical Halting problem of Turing machines, but that can often lead to cumbersome encodings. A much “cleaner” undecidability problem to work with is the correspondence problem proposed by Emil Post:

Definition 18 (Post’s Correspondence Problem (PCP)). *Consider the following function:*

Input *A finite sequence of pairs of strings $(s_1, t_1), \dots, (s_n, t_n)$.*

Output *Yes if there is a finite sequence of indices $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $s_{i_1} \dots s_{i_k} = t_{i_1} \dots t_{i_k}$;
and No otherwise.* \square

Example 18. *The correspondence problem*

$$\begin{array}{lll} s_1 = 1 & s_2 = 10 & s_3 = 011 \\ t_1 = 101 & t_2 = 00 & t_3 = 11 \end{array}$$

has a solution: $s_1 s_3 s_2 s_3 = 101110011 = t_1 t_3 t_2 t_3$.

The infinite aspect that makes this problem undecidable is that there is no bound on the length of the sequence of indices: if we have checked that there is no correspondence for any sequence up to length, say, 100, there is no guarantee that there is no sequence at all that has a correspondence.

We now give a reduction: we “encode” PCP into protocols by giving a computable translation f from PCP to protocols, so that the a PCP problem p has a correspondence (i.e. the answer should be *Yes*) iff the protocol $f(p)$ has an attack. It follows, if there *were* any decision procedure for protocol security, then we could build one for PCP using this translation f . Since PCP is undecidable, it follows that protocol security is undecidable.

Definition 19 (Reduction from PCP to Security Protocols inspired by [14]). *For the given PCP problem $((s_1, t_1), \dots, (s_n, t_n))$ define the following strands:*

- An initialization strand: $\text{Rcv}(\langle\langle X, Xs \rangle, \langle \$, \$ \rangle\rangle).\text{Snd}(\{\langle\langle X, Xs \rangle, \langle \$, \$ \rangle\rangle\}_k)$. Here $\$$ is a public constant. This takes (from the intruder) a sequence of indices that cannot be empty (there is at least one element X , but any number), and encrypts it with a secret key k . Assume also all characters of the strings s_i and t_i and the indices $1, \dots, n$ are public constants.
- For each of the (s_j, t_j) (where $j \in \{1, \dots, n\}$), we have an infinite number of strands of the form^[12]

$$\text{Rcv}(\{\langle\langle j, Xs \rangle, \langle Y_l, Y_r \rangle\rangle\}_k).\text{Snd}(\{\langle\langle Xs, \langle s_j \cdot Y_l, t_j \cdot Y_r \rangle\rangle\}_k)$$

The notation $s \cdot X$ is not a new operator but a notation for the following: let $s = c_1 c_2 \dots c_n$, then $s \cdot Y$ shall denote the term $\langle c_1, \langle c_2, \langle \dots, \langle c_n, Y \rangle \rangle \rangle \rangle$. Intuitively these strands step by step construct the strings that are induces by the indices that the intruder had chosen originally.

- Finally the strand: $\text{Rcv}(\{\langle \$, \langle X, X \rangle \rangle\}_k).\text{Snd}(\text{secret})$. This means that all indices by the intruder have been transferred into a string (i.e., we have reached the end marker $\$$) and the obtained strings are identical (both X) then the intruder has indeed found a solution to the PCP. In this case he gets the secret constant secret .

The goal is that the intruder never obtains secret .

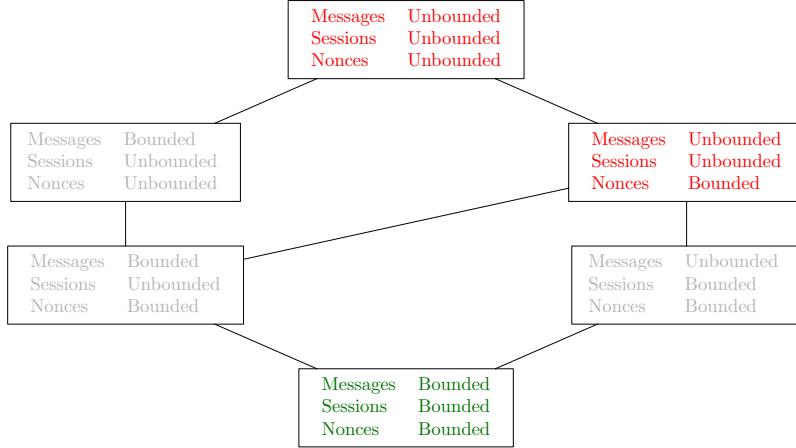
Theorem 3. *The intruder finds out secret iff the given PCP has a correspondence. Thus protocol security is undecidable.*

Observe that this reduction relies on two infinities:

¹²We want infinitely many copies of each strands, so that this input-output behavior can be performed any number of times. Since we have a *set* of strands, we achieve this by a renaming of the strands bound variables.

- The intruder must be allowed to compose arbitrary large messages (i.e., sequences of indices).
- There must be an unbounded number of sessions so that the honest agents can check the solution that the intruder has submitted, no matter how large it is.
- There are however no fresh nonces used in the protocol. So even without fresh nonces we have undecidability.

We update the decidability lattice by two marking the proved undecidable settings in :



In the upcoming sections, we will further fill this lattice.

12 Symbolic Transition Systems

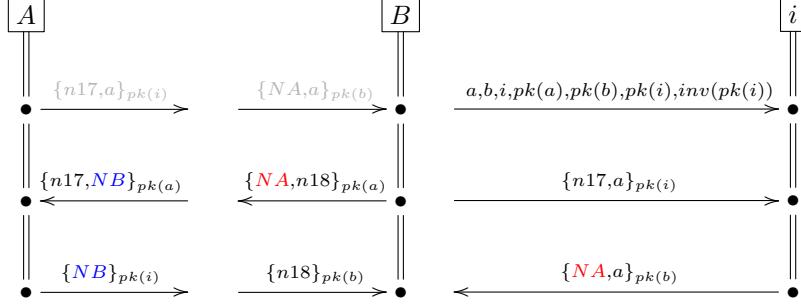
Let us now first bound the number of sessions (and thus of the nonces). Recall that even for a finite number of sessions we can get an infinite-state transition system if we do not bound the messages:

Example 19. In Example 13, there is an infinite choice of messages that the intruder can send to b: b expects a message of the form $\{NA, a\}_{pk(b)}$ and the intruder knows both $pk(b)$ and a. So he can choose an arbitrary term t from his knowledge and construct $\{t, a\}_{pk(b)}$.

This gives at this state infinitely many successor (so, as a tree, an infinite branching degree). Actually, even under some reasonable bounds on the intruder messages this is the point where search for an attack becomes really infeasible. The key idea is now that it is not really productive to list all kinds of terms t that the intruder could use as NA. Let us be *lazy* and *procrastinate* this choice of NA for now.

This means that we get a *symbolic state* that has a free variable NA and we must remember that, whatever NA is, it is something that the intruder can generate from his current knowledge. To store this information what the intruder has generated and at what state of this knowledge, we introduce the concept of an *intruder strand* – the messages the intruder has sent and received so far. We also will call this an (intruder) *constraint*, because we are interested for which values of the variables it can be fulfilled. Let us first look at the NSPK example how that looks like:

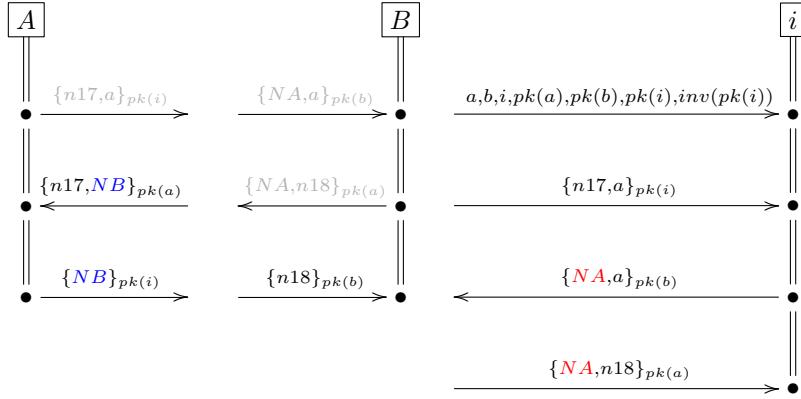
Example 20.



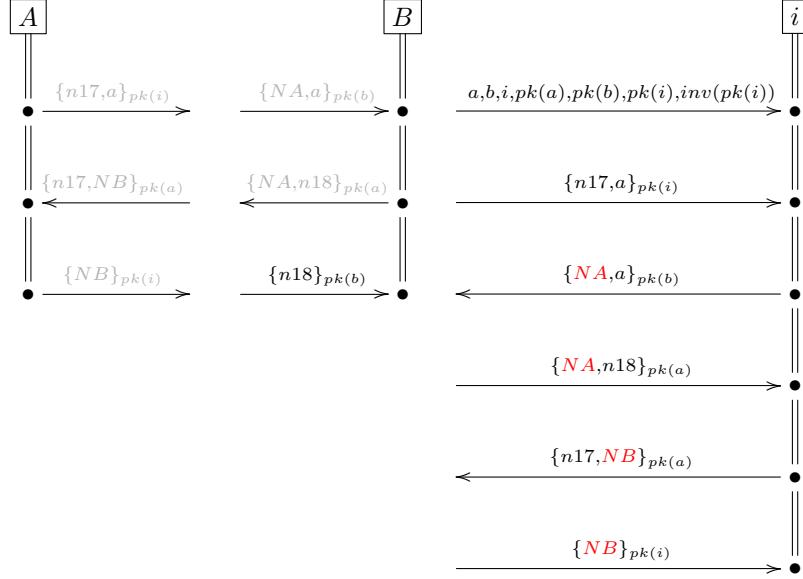
We here have an intruder strand with the following (intuitive) meaning:

- Incoming messages are messages that the intruder has learned. (In the first line we have the initial intruder knowledge, in the second line we have the message that a has sent to i first.)
- Outgoing messages are messages that the intruder has produced. (In the third line we have the message that the intruder has sent to b .)
- This should represent all **solutions** σ of the **free variables** such that the intruder can generate every outgoing message (under σ) when knowing all previous incoming messages (under σ). We are just **lazily** procrastinating that choice of σ !

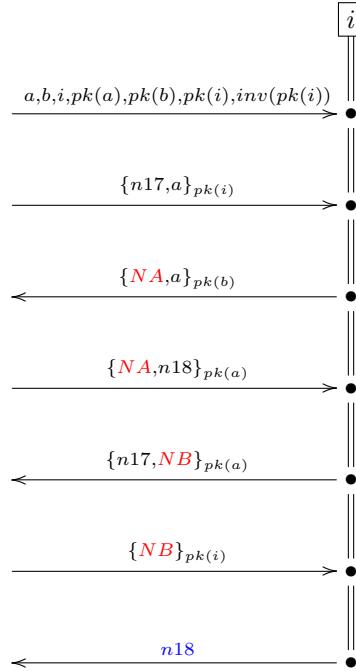
The next step could be that b answers the message; note that b 's answer depends on the free variable NA – we have not instantiated in the previous step as of our laziness, so now also honest agents are sending around messages with variables:



Let us say that the intruder next talks to a and gets an answer:



Let us finally consider the secrecy goal and check whether the intruder can produce the secret `n18` right now. To that end, we can just add put it as a message he has to derive on the intruder strand. Then secrecy is violated, if we can find a solution for this constraint:



In section 13 we give a decision procedure for such constraints. In fact there is a solution for this example (and only one): $\sigma(NA) = n17$ and $\sigma(NB) = n18$; that's the secrecy attack from Example 13. \square

Before we have a closer look at the constraints, let us define a symbolic transition and how that induces constraints:

Definition 20 (Symbolic Transition System). A symbolic state *consists of*

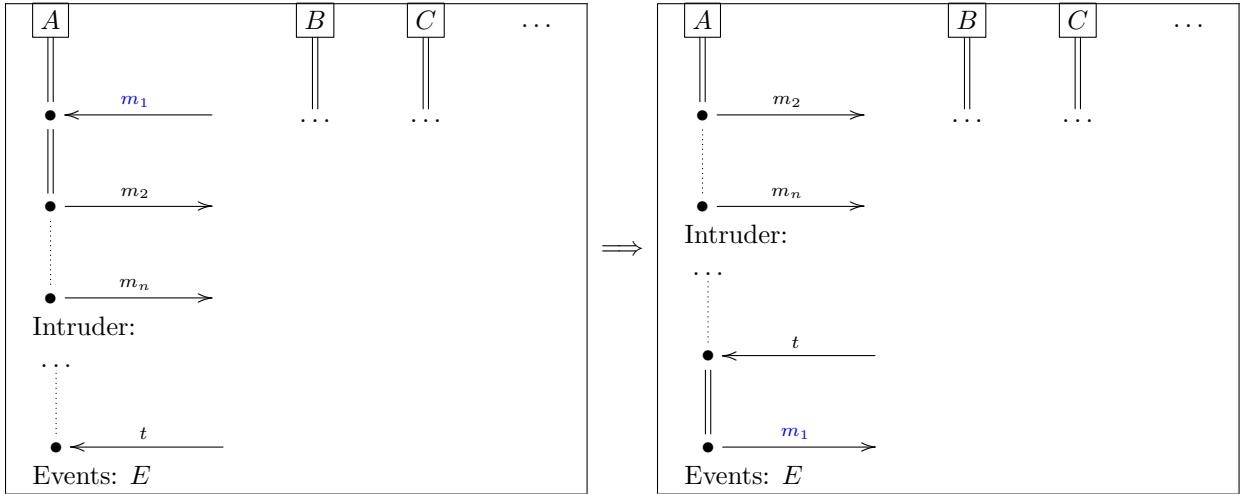
- a set of honest strands
- an intruder strand
- a set of events that have occurred.

The initial state has a set of closed strands for the honest agents, a receive step for the intruder strand containing the initial intruder knowledge, and an empty set of events.

We have transition rules for honest agents sending and receiving messages, checking equations and emitting events; these are similar to the ones in the original (concrete) transition system, and we define them in the following subsections.

12.1 Transition: Receiving

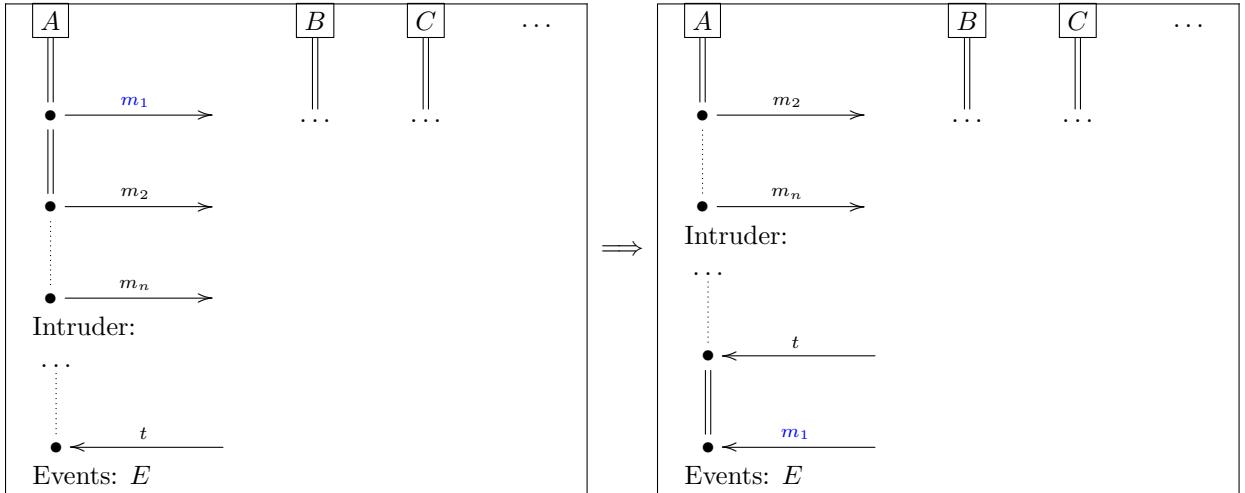
Messages that honest agents are receiving are simply moved (with inverse direction) to the intruder knowledge:



Note that this rule is actually much simpler than the corresponding ground rule, because it does not care for expressing the solutions of the intruder constraint.

12.2 Transition: Sending

Messages that honest agents are sending are also simply moved (with inverse direction) to the intruder knowledge:



12.3 Events and Equations

Events can just be carried over as before. Equations we leave to the reader as an exercise.

13 The Lazy Intruder

We first define the meaning of lazy intruder constraints and then show how to solve them.

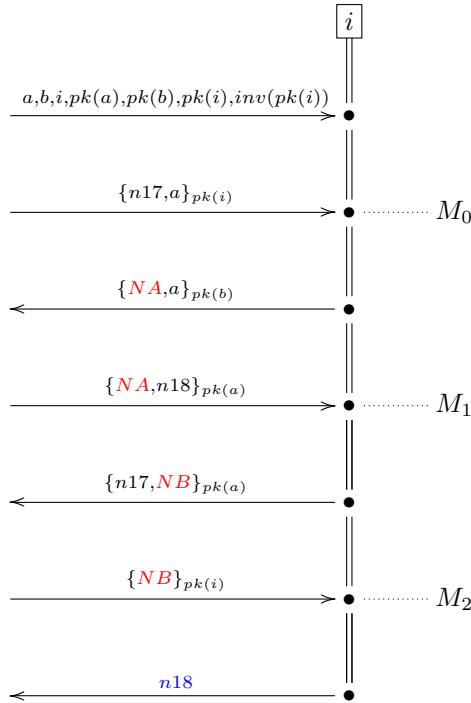
Definition 21. An *intruder constraint* is a strand where all variables are free, i.e., all variables first occur in an outgoing message.

At any point \bullet in the constraint, the *intruder knowledge* at that point is the set of all messages received so far.

Given an intruder constraint C , and σ a substitution of all its free variables with ground terms. Then σ is called a *solution* of C iff:

- for every outgoing message m of C , it holds that $\sigma(M) \vdash \sigma(m)$ where M is the intruder knowledge at that point.

Example 21. Consider again the constraint in from the NSPK example. Let us label the intruder knowledge at different points M_0, \dots, M_2 :



The meaning of this constraint is any substitution σ such that the following Dolev-Yao deductions hold:

$$\begin{aligned}
 M_0 &:= \{a, b, i, pk(a), pk(b), pk(i), \\
 &\quad inv(pk(i)), \{n17, a\}_{pk(i)}\} \\
 \sigma(M_0) &\vdash \sigma(\{NA, a\}_{pk(b)}) \\
 M_1 &:= M_0 \cup \{\{NA, n18\}_{pk(a)}\} \\
 \sigma(M_1) &\vdash \sigma(\{n17, NB\}_{pk(a)}) \\
 M_2 &:= M_1 \cup \{\{NB\}_{pk(i)}\} \\
 \sigma(M_2) &\vdash \sigma(n18)
 \end{aligned}$$

□

13.1 Solving Constraints

We now give a procedure for solving constraints. This will be done with rules of the form:

$$[S] \rightsquigarrow [S']$$

The meaning of such a rule is that to solve S , one way is to try to solve S' . Put another way, if we can solve S' , then also we can also solve S .

For any given constraint S_0 , the relation \rightsquigarrow induces a search tree for solutions of the constraints as follows:

- The root node is the constraint S_0
- Every node S has as children the strands that are reachable in one step with \rightsquigarrow :
 - i.e., if $S \rightsquigarrow S'$ then S' is a child of S .

Important feature we prove about this constraint tree are:

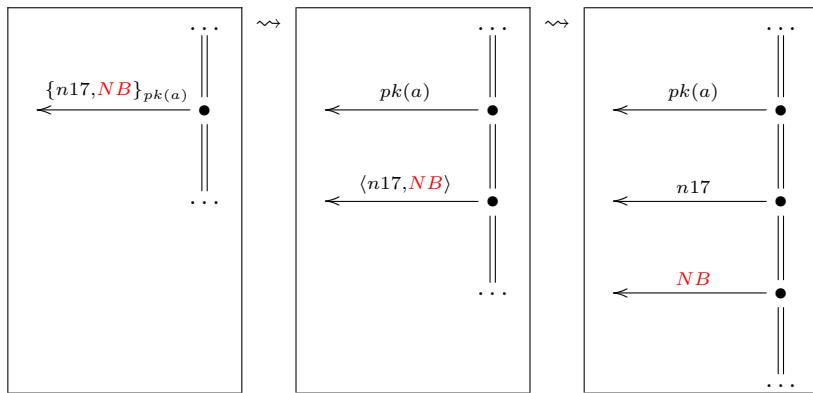
- Termination: the tree has finitely many nodes (we do not run into an infinite search of solutions)
- Soundness: we only find correct solutions.
- Completeness: we do not miss solutions.

This means in particular that every leaf of the tree is either *simple* or unsolvable (Explained below, easy to check). The root has a solution iff any leaf has a solution.

13.2 Composition

Idea: to construct an outgoing message of the form $f(t_1, \dots, t_n)$ it is sufficient that f is a public symbol and the intruder can construct the submessages t_i :

Example 22.



More general we define the intruder composition rule as follows:

Definition 22 (Lazy Intruder Composition Rule).

$$S_1.\text{Snd}(f(t_1, \dots, t_n)).S_2 \rightsquigarrow S_1.\text{Snd}(t_1) \dots \text{Snd}(t_n).S_2$$

if $f/n \in \Sigma_p$, i.e., f is a public function symbol.

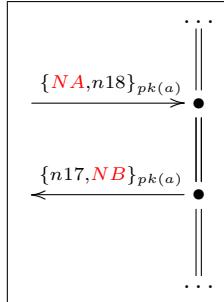
The lazy intruder composition rule corresponds to the (Compose) rule of the Dolev-Yao model: the intruder can compose terms he knows by applying a public function symbol. One

may consider the lazy intruder as a backward search: rather than blindly composing terms (that may be useless) in a forward exploration of terms we can generate, we rather start at the terms we want to obtain, and if the toplevel symbol is public, we backwards-apply composition, i.e., try if we can compose the subterms.

13.3 Unification

Idea: another way to construct an outgoing message is to use a previously received message that has the right “shape”:

Example 23.



These messages can be unified under unifier $\sigma = [NA \mapsto n17, NB \mapsto n18]$. This unifier has to be applied to the entire intruder constraint.

13.3.1 Computing the Most General Unifier – mgu

At this point we need a classic algorithm: computing the most general unifier, or mgu for short.

Definition 23 (Unification). A *unification problem* is a set $\{(s_1 \doteq t_1), \dots, (s_n \doteq t_n)\}$ of equations of terms. (We use the symbol \doteq again to distinguish from the normal equality symbol.)

A *unifier* σ for this unification problem is a substitution such that

$$\sigma(s_1) = \sigma(t_1) \text{ and } \dots \text{ and } \sigma(s_n) = \sigma(t_n).$$

In general, there are infinitely many unifiers, for instance the problem $\{x \doteq f(y)\}$ has infinitely many unifiers like $\tau = [x \mapsto f(c), y \mapsto c]$ (if you replace c by any other term, then you get another unifier). However, there is in some sense a *most general* unifier for this problem: $\sigma = [x \mapsto f(y)]$ since all other unifiers are a special case of σ . This notion of generality can be defined as follows:

Definition 24. We say that a substitution σ is at least as general as substitution τ , and write $\sigma \preceq \tau$,¹³ if there is a substitution θ such that $\theta \circ \sigma = \tau$.

Intuitively, this means that we can obtain the more special substitution τ from the more general σ by composing σ with another substitution θ . In the above example $\{x \doteq f(y)\}$, we have with $\theta = [y \mapsto c]$ that $\theta \circ \sigma = \tau$, and thus $\sigma \preceq \tau$. In the free algebra we have the nice property that a unification problem has either no unifier or there is a most general unifier (that is at least as general as any other unifier). There is a fairly simple recursive algorithm that either computes the most general unifier if it exists, or returns *failure* otherwise:

Definition 25 (Algorithm $mgu(U, \sigma)$).

Input: a unification problem U , a substitution σ (initially the identity $[]$).

Output: A substitution or answer *failure*.

If $U = \emptyset$ then return σ . Otherwise pick any equation $(s \doteq t)$ in U and

- if $s = t$, continue to $mgu(U \setminus \{s \doteq t\}, \sigma)$.

- if s is a variable:

- if $s \in vars(t)$: return *failure*

¹³The direction \preceq may seem counter-intuitive since it calls the more general substitution “smaller”: this convention becomes intuitive if you see the sizes of terms as the size of the substitution, and then the smallest (and thus most general) element in this partial order is the identity $[]$ that does not substitute anything.

- otherwise: update σ with $[s \mapsto t]$
and continue to $mgu(\sigma(U \setminus \{s \doteq t\}), \sigma)$
- if t is a variable: symmetric to previous case
- otherwise, i.e., $s = f(s_1, \dots, s_n)$ and $t = g(t_1, \dots, t_m)$:
 - if $f \neq g$: return *failure*.
 - if $f = g$ (and thus $n = m$): continue with $mgu((U \setminus \{s \doteq t\}) \cup \{s_1 \doteq t_1, \dots, s_n \doteq t_n\}, \sigma)$.

We sometimes just write $mgu(s \doteq t)$ for $mgu(\{s \doteq t\}, []])$

□

Example 24.

$$\begin{aligned}
 & mgu(\{ \{NA, n18\}_{pk(a)} \doteq \{n17, NB\}_{pk(a)} \}, []) \\
 &= mgu(\{ pk(a) \doteq pk(a), \langle NA, n18 \rangle \doteq \langle n17, NB \rangle \}, []) \\
 &= mgu(\{ \langle NA, n18 \rangle \doteq \langle n17, NB \rangle \}, []) \\
 &= mgu(\{ NA \doteq n17, n18 \doteq NB \}, []) \\
 &= mgu(\{ n18 \doteq NB \}, [NA \mapsto n17]) \\
 &= mgu(\emptyset, [NA \mapsto n17, NB \mapsto n18]) \\
 &= [NA \mapsto n17, NB \mapsto n18]
 \end{aligned}$$

Theorem 4. Consider any unification problem U . If $mgu(U) = \text{failure}$, then U has no unifier, otherwise, if $mgu(U) = \sigma$ then σ is the most general unifier of U , i.e., for every unifier τ of U , we have $\sigma \preceq \tau$.

13.3.2 The Lazy Intruder Unification Rule

Using the mgu function, we can now finally define the lazy intruder unification rule. If the intruder received a term s and needs to send a term t , and s and t are unifiable, and σ is the most general unifier of s and t , then we can apply σ to the entire constraint and consider the sending of t as “done”:

Definition 26 (Lazy Intruder Unification Rule).

$$S_1.\text{Rcv}(s).S_2.\text{Snd}(t).S_3 \rightsquigarrow \sigma(S_1.\text{Rcv}(s).S_2.S_3)$$

if s and t are not variables, and $\sigma = mgu(s \doteq t)$.

The lazy intruder unification rule corresponds to the (Axiom) rule of the Dolev-Yao intruder. A crucial condition of the unification rule is that neither s nor t can be a variable. This is exactly what makes the intruder lazy: when the term to generate is a variable, we do not bother to select a value for it (at this point).

One may wonder why we do not allow at least the received term s to be a variable. This is because this variable then must have originated in an earlier Send-step (since all variables in an intruder constraint must first occur in a Send-step); if that Send-step is a composed term, we should solve that first; otherwise, i.e., if we have a constraint of the form

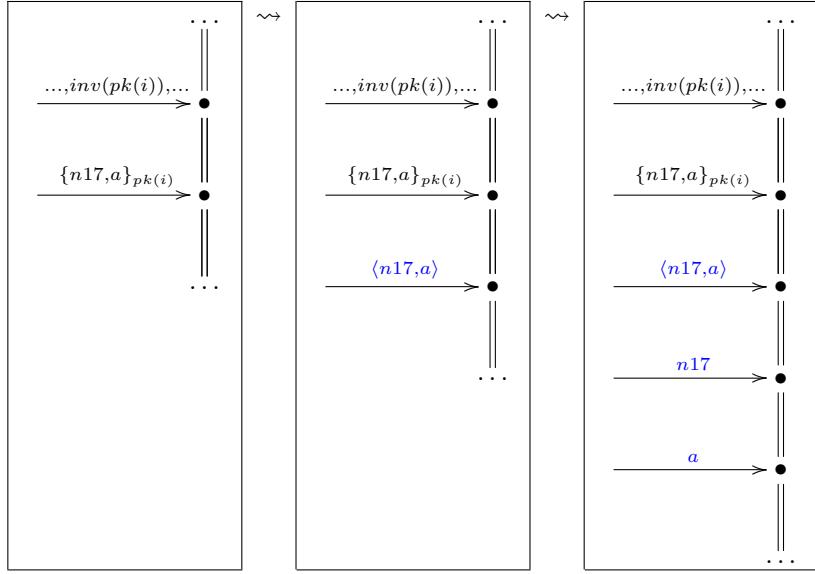
$$\dots \text{Snd}(x) \dots \text{Rcv}(x) \dots \text{Snd}(t) .$$

Thus the intruder has sent a message x that he then received back. Thus unifying t with x would not be wrong, but redundant, and go against the philosophy of laziness here. For this reason we also forbid unification steps where the received message is a variable.

13.4 Simple Analysis

Idea: if the intruder received an encrypted message and has the decryption key in his knowledge, then we can also add the decrypted message. (Similar for pairs, he can obtain the components immediately.)

Example 25.



Definition 27 (Lazy Intruder Simple Analysis Rules).

$$\begin{aligned}
 S_1.\text{Rcv}(\text{inv}(k)).S_2.\text{Rcv}(\{m\}_k).S_3 &\rightsquigarrow S_1.\text{Rcv}(\text{inv}(k)).S_2.\text{Rcv}(\{m\}_k).\text{Rcv}(m).S_3 \\
 S_1.\text{Rcv}(k).S_2.\text{Rcv}(\{m\}_k).S_3 &\rightsquigarrow S_1.\text{Rcv}(k).S_2.\text{Rcv}(\{m\}_k).\text{Rcv}(m).S_3 \\
 S_1.\text{Rcv}(\langle m_1, m_2 \rangle).S_2 &\rightsquigarrow S_1.\text{Rcv}(\langle m_1, m_2 \rangle).\text{Rcv}(m_1).\text{Rcv}(m_2).S_2 \\
 S_1.\text{Rcv}(\{m\}_{\text{inv}(k)}).S_2 &\rightsquigarrow S_1.\text{Rcv}(\{m\}_{\text{inv}k}).\text{Rcv}(m).S_2
 \end{aligned}$$

Note: this now can lead to non-termination, if one repeatedly applies a rule to the same term. But since this is redundant (not adding new knowledge), we can exclude repeated application to the same term.

13.5 ★ Full Analysis

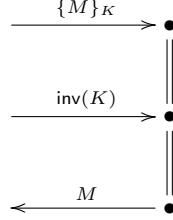
Analysis in the symbolic model is tricky and we can avoid it, if we “out-source” analysis into special strands, representing honest agents that perform analysis steps for the intruder.

The analysis rules given so far are for the simple case that the intruder receives a composed message and already has the decryption key for it in his knowledge. In general, analysis is more difficult for the following reasons:

- The key-term may contain variables, like $\{m\}_{pk(A)}$. Suppose the only private key that the intruder knows is $\text{inv}(pk(a))$. Then we actually get a case split: if $A = i$ then he can decrypt it, otherwise he cannot.
- The key-term may be composed, like $\{m\}_{h(n1,n2)}$. If the intruder knows n_1 and n_2 and h is a public function, he can compose the decryption key, but the simple analysis rule would miss this.

- The intruder may receive a decrypted message that he cannot decrypt at first, but learn the decryption key in a later step.

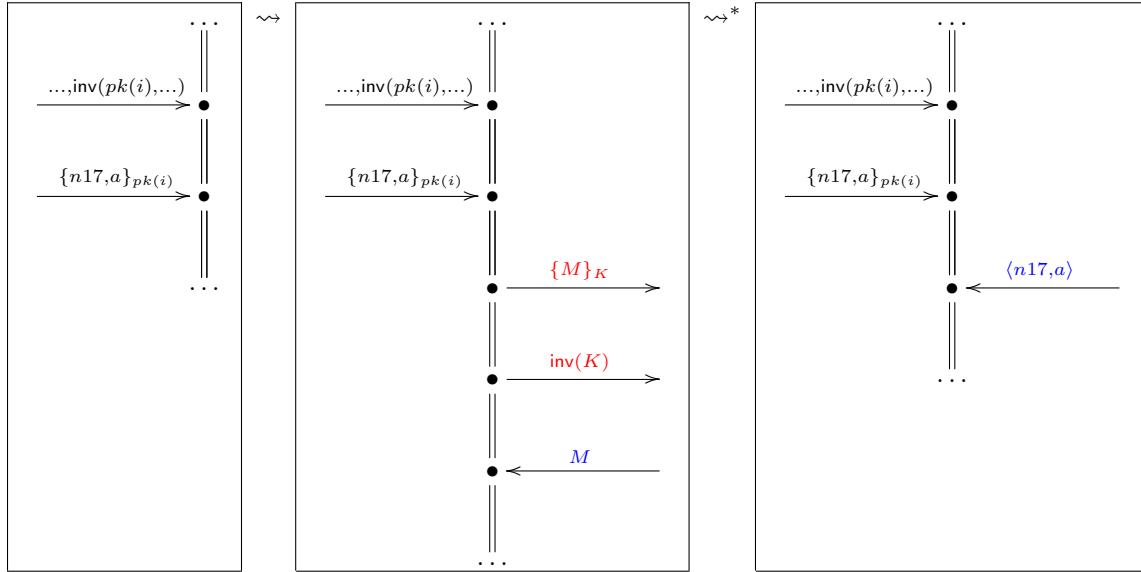
An idea is to “out-source” analysis in the following sense: suppose we add special strands to the protocol like this one:



This means an agent “offering as a service” the same functionality that the analysis rule for asymmetric encryption provides: give me a public-key encrypted message and the private key, then you get the result. Since all agents but the intruder are honest, we do not have to worry that the intruder transmits his private key to this service – nobody in our model will attack the intruder. Also this would not insert any new attacks since the intruder only obtains something that the normal Dolev-Yao model gives him anyway. If we do this for all decryption rules, we can consider an intruder who does not decrypt himself. In fact, we would then obtain a transition system where the intruder for any decomposition would ask one of the special agents.

This corresponds to the intruder to be able to insert an analysis step at any point into the constraints. For instance, we would then have for NSPK:

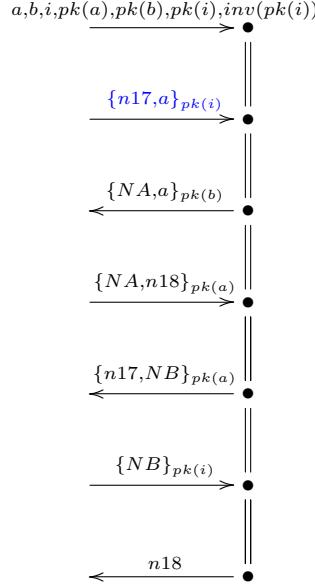
Example 26.



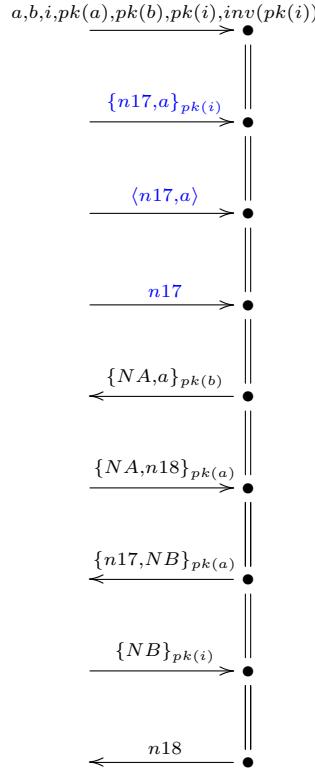
Without care, this can easily lead to non-termination. However, if we are focusing on the problem of a bounded number of sessions (i.e., infinitely many strands without the special analysis strands) then we can also limit the number of analysis strands: for every constructor in the honest strands that permits analysis, we shall have one corresponding analysis strand.

13.6 Constraint Solving Complete Example

We now give the complete example of solving the intruder constraint from Example 21. The initial constraint to solve is the following:

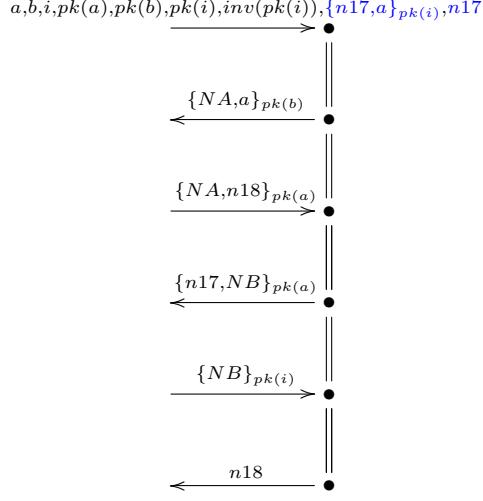


Here we can decrypt $\{n17, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$. We can also decompose the resulting pair $\langle n17, a \rangle$. Since a is already known, we only really learn $n17$ from this. With this we obtain the constraint:

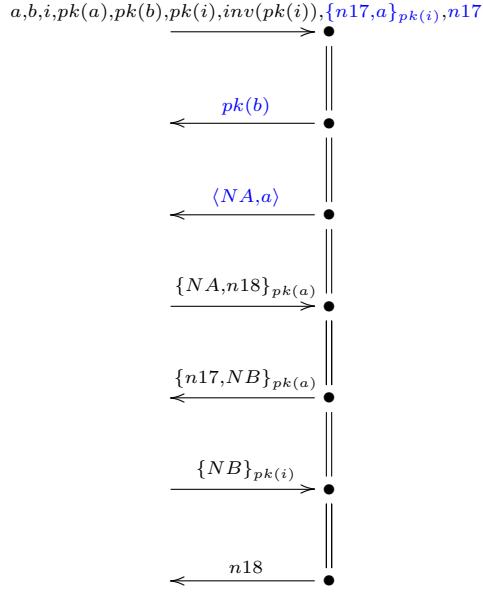


Let us simplify notation to put these messages into the initial intruder knowledge. Also note

that it is not a restriction to remove now the pair $\langle n17, a \rangle$ since we have the components and the intruder can always compose the pair again. (In fact this later reduces a few redundant cases.)

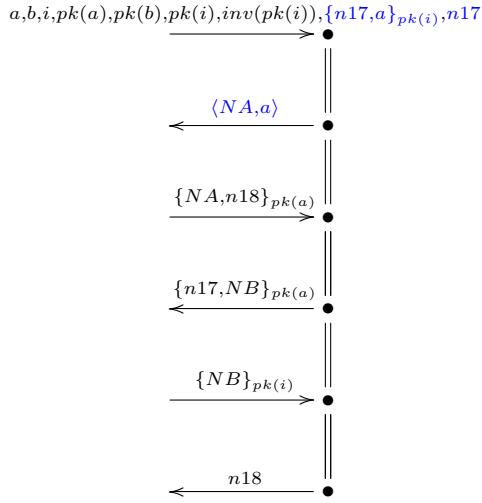


Consider the first outgoing message $\{NA, a\}_{pk(b)}$. For each outgoing message, there are always basically two possibilities: composition or unification. Unification does not work here since the intruder has nothing fitting in his knowledge. So let us go for composition:

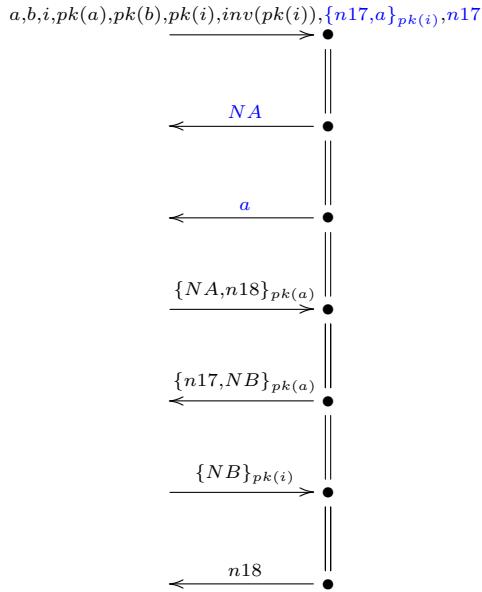


The intruder is now required to produce the key $pk(b)$ which is directly in the knowledge, so

we can remove it using unification:

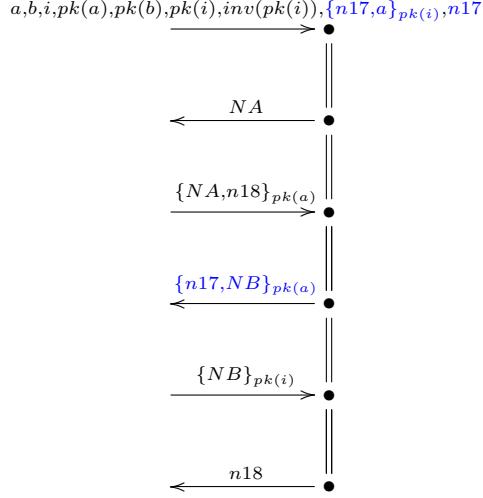


For $\langle NA, a \rangle$ we can only use composition (in fact, had we not removed the incoming pair before, we would have to deal with it here as an extra case):

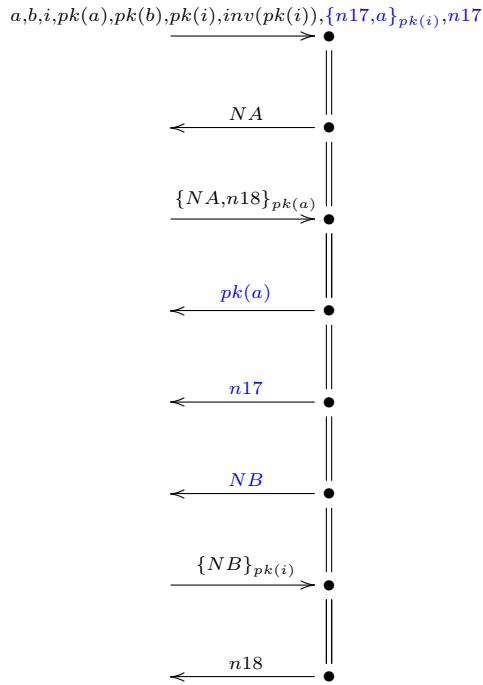


Now an important point is that we have to generate NA which is a variable. The composition rule cannot be applied to it, because it is not of the form $f(\dots)$ for a public function f . The unification rule can only be applied between two terms s and t that are **not variables**. Thus, no rule can be applied – we leave NA for now. This is why the intruder is **lazy** – any value for NA will do, so why bother!

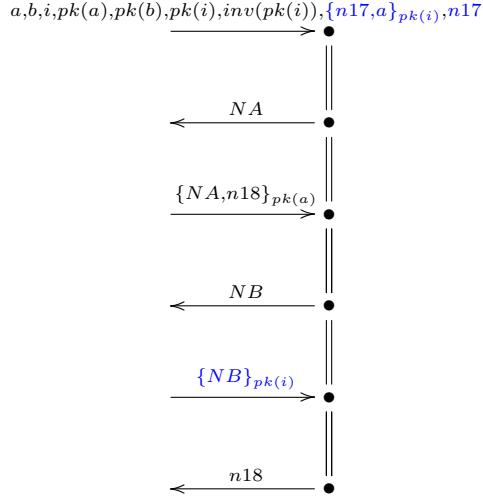
The next item to generate is a : it is already known and can be removed with unification:



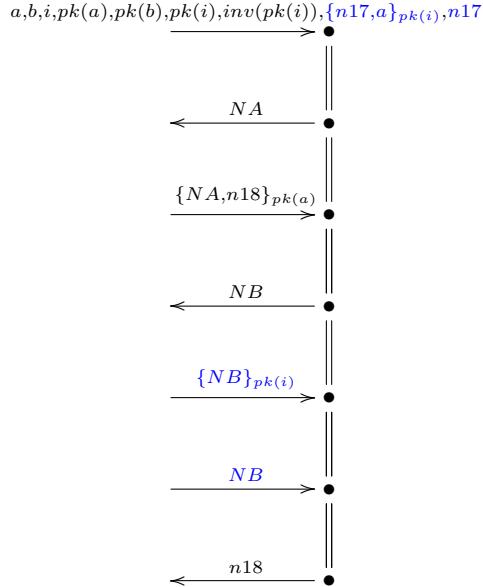
Consider now the first outgoing message that is not a variable: $\{n17, NB\}_{pk(a)}$. Again two general possibilities: unification or composition. Both cases are possible here, and to build the \rightsquigarrow tree, we have to follow both. Let us follow composition first (both for the encryption and the pair):



$pk(a)$ and $n17$ are known and can be done with unification. NB is a variable and we are lazy again here, just leave it for now:



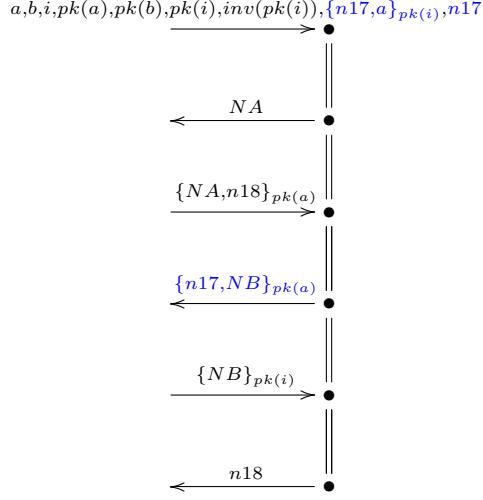
Looking at the next (incoming message), we can apply decryption here, giving us NB :



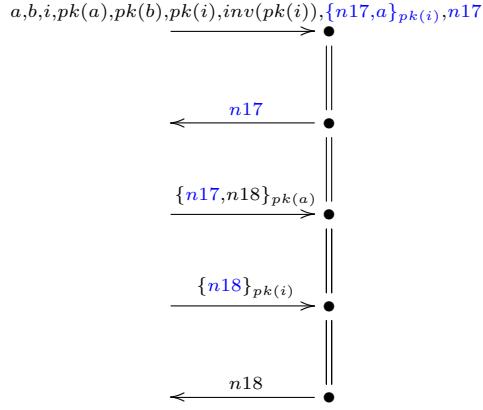
This is pretty useless: whatever NB is, we have constructed that ourselves earlier! Actually – this is the normal protocol execution where the intruder has generated some value NB and now received it back from Alice.

It remains to construct $n18$. That's impossible, because we cannot apply composition (since $n18$ is not a public function), and unification is impossible: we do not have $n18$ in our knowledge.

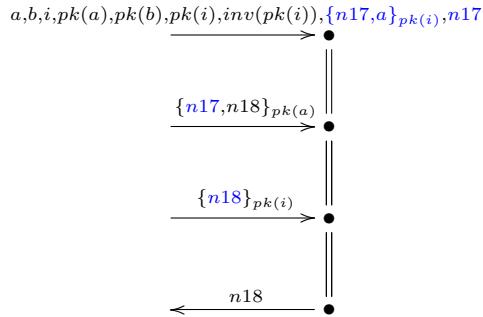
So this branch of the \rightsquigarrow tree fails and we have to backtrack to the branching point from before:



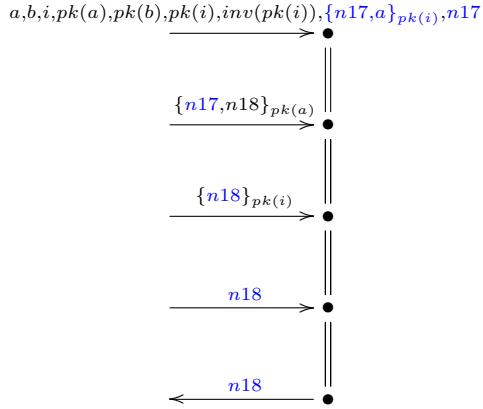
We can apply unification between the incoming message $\{n17, NB\}_{pk(a)}$ and the outgoing message $\{NA, n18\}_{pk(a)}$. The unifier (according to the mgu algorithm) is $\sigma = [NA \mapsto n17, NB \mapsto n18]$. We apply it to entire constraint and remove the outgoing message we have just “done” by unification:



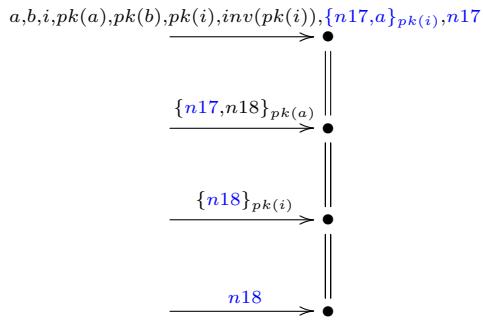
This unification is actually the central point in finding the solution. We have “retro-actively” decided that the intruder used $n17$ as nonce NA . This of course requires that the intruder knows $n17$. He does – so one unification step:



The message $\{n18\}_{pk(i)}$ can be decrypted, so we get $n18$:



The remaining outgoing message $n18$ is one simple unification step:



Solved!

13.7 Summary

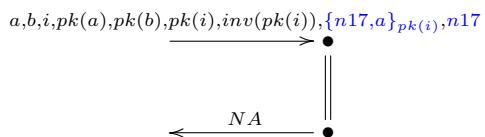
The procedure for solving constraints (with simple analysis) can be summarized as follows:

- Always start with the first step that has not been considered yet.
- If incoming: can simple decryption be applied?
- If outgoing:
 - If it is a variable, leave it for now and continue with the next step.
 - If it is not a variable, consider independently (with backtracking!) the following cases:
 - * Composition (if it is a public operator)
 - * Unification – with any incoming message that is not a variable.
- Whenever a unification is done where variables are substituted, apply to the entire constraint and go back to the first message that was affected!
- When all remaining outgoing messages are variables, the constraint is solved. We call that a **simple** constraint.

13.8 Simple Constraints

Definition 28. An intruder constraint is called simple if all outgoing messages are variables.

For instance, the constraint we obtained at the end of the NSPK example is simple since it has no more outgoing messages. Moreover, in the intermediate steps, the strands have a simple *prefix* like this one:



This is simple since the only outgoing message NA is a variable. Note that we cannot apply composition or unification steps to such a message – and it would be pointless, because it requires the intruder to only send *some* message. Since the intruder can always construct *some* message we have:

Lemma 2. *Every simple constraint has a solution.*

Thus, when we arrive at a simple constraint, we have found a solution. If we arrive at a constraint that is not simple and no further intruder rules can be applied, then the constraint is not satisfiable – as the following lazy intruder correctness theorems show.

13.9 * Correctness of the Lazy Intruder

Important properties of the lazy intruder:

- Termination: every unification and composition step makes the constraint simpler, this cannot go on forever. The analysis steps can only produce subterms of terms we already have.
 - Soundness: the lazy intruder procedure finds only correct solutions (covered by the Dolev-Yao model)
 - Completeness: if a constraint has a solution, the lazy intruder will find it:
 - Consider any solution of a constraint.
 - Then the constraint is either already simple or one of the lazy intruder steps gets us to a new constraint that still supports that solution.
 - By termination, we eventually arrive at a simple constraint that supports the considered solution.

Finally, one can show that the problem of protocol security with bounded sessions (and nonces) but unbounded messages is co-NP-complete.

Theorem 5 (Termination). *The lazy intruder terminates, i.e., for a given constraint S , there are only finitely many S' such that $S \rightsquigarrow^* S'$.*

Proof. For a constraint S let us say its *weight* is a triple (l_1, l_2, l_3) of positive integers where

- l_1 is the number of variables in S .
 - l_2 is the number of terms in S that have not been analyzed (i.e., where the intruder does not have the subterms).
 - l_3 is the size of all terms to be sent (number of characters) together.

We order the components lexicographically, i.e., the weight l_1 is the most significant, l_2 second, and l_3 least. For instance $(3, 2, 10) < (3, 3, 3)$. Then every rule of the lazy intruder reduces the weight, and it is positive in all three components, so there cannot be an infinite chain of lazy intruder steps. Also, for every constraint S there are only finitely many constraints S' reachable in one step. \square

Theorem 6 (Soundness). *The lazy intruder is sound: if $S \rightsquigarrow^* S'$ and S' has a solution, then also S has a solution.*

Proof. Every reduction rule can be justified as sound by the Dolev-Yao rules. \square

Theorem 7 (Completeness). *The lazy intruder is complete: if S has a solution, then there is a simple S' with $S \rightsquigarrow^* S'$.*

Proof. It is sufficient to show that for every non-simple S that has a solution, we have $S \rightarrow S'$ for some S' that has a solution. From this plus termination then follows completeness.

Let S be a non-simple constraint so that σ is a solution of S , i.e., every outgoing message $\text{Snd}(t)$ in S with M being the set of messages received before $\text{Snd}(t)$, it holds that $\sigma(M) \vdash \sigma(t)$ (where \vdash is the Dolev-Yao intruder deduction). Consider the first such step where t is not a variable (if there is none, the constraint is already simple). We have the following cases:

- If the last step in the deduction $\sigma(M) \vdash \sigma(t)$ is (Axiom): then there is a term $s \in M$ such that $\sigma(s) = \sigma(t)$. If s is not a variable, then this covered by the unify rule. Otherwise, if s is a variable, we have a constraint of the form

$$\dots \text{Snd}(s) \dots \text{Rcv}(s) \dots$$

and thus there must be solution that does not rely on $\text{Rcv}(s)$, because the intruder sent that earlier himself. We can thus proceed with that “simpler” solution.

- If the last step in the deduction $\sigma(M) \vdash \sigma(t)$ is (Compose): then we can similarly apply compose.
- If the last step in the deduction $\sigma(M) \vdash \sigma(t)$ is an analysis step. Let s_0 be the term being analyzed here. We distinguish two cases.
 - Let us first suppose that s_0 is obtained by an axiom rule. If there is a non-variable term $s \in M$ such that $s_0 = \sigma(s)$, then an out-sourced analysis steps is applicable right before $\text{Snd}(t)$ and lead to the desired analysis result. Otherwise, if there is a variable $x \in M$ such that $s_0 = \sigma(x)$ then the constraint has the form:

$$\dots \text{Snd}(x) \dots \text{Rcv}(x) \dots$$

then again there must be a simpler solution that does not rely on $\text{Rcv}(x)$ and that we can proceed with.

- If s_0 is obtained from an analysis step, then proceed with that analysis step first.
- If s_0 is obtained from a composition step, then the intruder has first composed a message that he then analyzed. We can simplify this solution to eliminate this composition-analysis-pair, and proceed with that solution.

Thus in all cases, we get closer to a solved constraint. \square

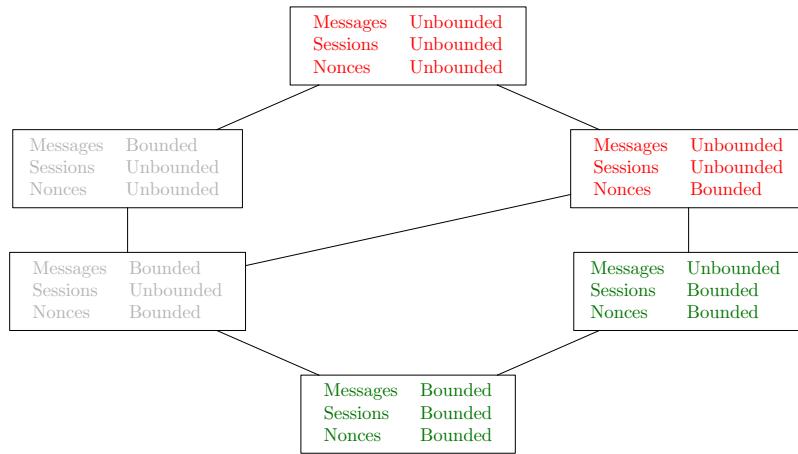
Theorem 8.

Theorem 9 (NP-Completeness). *The problem of protocol security with bounded sessions (and nonces) but unbounded messages is co-NP-complete.*

Proof. First, one can show that for every Boolean formula, we can generate a protocol of a size that is polynomial in the size of the formula (and this generation takes polynomial time) and so that the formula is satisfiable iff the protocol has an attack. Thus insecurity is NP-hard.

Second, we can show that there is a non-deterministic algorithm in polynomial time (in the size of the initial state of the protocol) to decide protocol insecurity (without bounding messages). To that end, consider that in the symbolic transition system a trace cannot be longer than the total number of steps in all strands; for each constraint the number of lazy intruder reductions is also polynomially bounded by the weight of the constraint.¹⁴ □

This gives us one more entry in the decidability lattice: we can handle unbounded messages if sessions (and thus nonces) are bounded:



The remaining two items will be covered in the next chapter.

14 Abstract Interpretation

While the technique of the lazy intruder is actually very good for quickly finding attacks, it can only verify for a bounded number of sessions: when we have verified a protocol for 3 sessions, it does not tell us if there is maybe an attack for 4 sessions. Since of protocols like TLS millions of sessions exist worldwide at the same time, and the verification is growing exponentially with the number of sessions, this is not really feasible. One often refers to this as the *state-explosion problem* of model-checking. Moreover this way we never obtain a statement about *any* number of sessions.

There is however another approach that can side-step the state-explosion problem and verify for infinitely many sessions: abstract interpretation. This approach is used in static program analysis. A simple example is a compiler that shall check that every variable of a program is initialized before it is used. In general we cannot tell that for sure because of the halting problem: if an uninitialized variable is used first after a while loop, then the answer depends on whether this while loop will ever terminate (which is undecidable in general). The simple solution is: let us assume the while loop could terminate, then the compiler should flag this as a use of an uninitialized variable. We are thus considering an *over-approximation* (of what can happen), potentially ruling out a few good programs, but we get an analysis that is much simpler, that will never accept a bad program, and that always terminates.

In protocol verification this idea has been also very successful using logical formulas to give an over-approximation of what can ever happen in a protocol, in particular what the intruder can ever know. Since this is literally all a bit *abstract*, let us consider a concrete example step by step.

¹⁴A naïve implementation of substitutions can actually lead to an exponential runtime here, but there is an implementation that avoids this blow-up.

14.1 An Example

The basic formalism we use here are logical implications like $p.q \Rightarrow r.t$ meaning “when p and q hold, then also r and t hold”. This is often called *(definite) Horn clauses*.¹⁵ We use here a notion from AIF $_{\omega}$ [24], a novel add-on for OFMC for denoting the implications, while AIF $_{\omega}$ rules actually mean state transitions. For the formulas we are using in this section, however, it does not make a difference. The full language of AIF $_{\omega}$ will be explained in a special chapter later.

Let us begin with the intruder, and let $\text{iknows}(m)$ stand for “the intruder knows m ”. We could specify first his initial knowledge like this:

```
=> iknows(a); => iknows(pk(a));
=> iknows(b); => iknows(pk(b));
=> iknows(i); => iknows(pk(i)); => iknows(inv(pk(i)));
```

to mean that the intruder knows the agents a, b, i their public keys and his own private key. Here we use implications without a left-hand side, i.e., the right-hand side facts holds unconditionally.

If we want to consider more agents, then we would have to make long enumerations. Therefore AIF $_{\omega}$ allows to first define some data types:

```
Honest = {a,b};
Dishonest = {i};
User = Honest ++ Dishonest;
```

We can then use these types in rules and later change the number of agents without changing any rules and without making long enumerations. To that end, every rule has a rule head of the form

$$\text{rule name}(\text{Variable : Type}, \text{Variable : Type}, \dots)$$

The type can be any of the user-defined types (like Honest here) or *Untyped*. Untyped variables, however, have an important restriction: every untyped variable must occur in the left-hand side (and may occur in the right-hand side also). In other words, it is not allowed to have untyped variables that only occur in the right-hand side.

The listing above is then written simply as follows:

```
users(A: User) => iknows(A);

publickeys(A: User) => iknows(pk(A));

privatkeys(D: Dishonest) => iknows(inv(pk(D)));
```

The first rule has the name “users” and says that for any A of type *User*, the intruder knows A . The other rules are similar.

Let now

- $\text{crypt}(k, m)$ stand for $\{m\}_k$, i.e., the asymmetric encryption with key k of message m ,
- and $\text{pair}(m_1, m_2)$ stand for the pair of m_1 and m_2 .

Then the Dolev-Yao model for asymmetric encryption and pair is described by the following formulas:

```
asymenc(M1: untyped, M2: untyped)
iknows(crypt(M1, M2)).iknows(inv(M1)) => iknows(M2);

asymdec(M1: untyped, M2: untyped)
iknows(M1).iknows(M2) => iknows(crypt(M1, M2));
```

¹⁵Actually, by definition, Horn clauses can only have one fact on the right of the arrow, but $p.q \Rightarrow r.t$ can be regarded as an abbreviation of the two Horn clauses $p.q \Rightarrow r$ and $p.q \Rightarrow t$.

```

pair(M1: untyped, M2: untyped)
iknows(M1).iknows(M2) => iknows(pair(M1,M2));

proj(M1: untyped, M2: untyped) iknows(pair(M1,M2))
=> iknows(M1).iknows(M2);

```

The first formula says: if the intruder knows any messages M_1 and M_2 , then he also knows $\text{crypt}(M_1, M_2)$, and similar the second, that he can decrypt $\text{crypt}(M_1, M_2)$ if he knows (the private key) $\text{inv}(M_1)$. The rules for pair follow the same principle. The dot (.) is in this syntax logical “and” and the arrow (\Rightarrow) is implication. In the initial intruder knowledge before, the (\Rightarrow) was simply used without anything on the left-hand side, i.e., the right-hand side holds unconditionally.

From all the formulas so far we can now for instance derive the fact $\text{iknows}(\text{crypt}(\text{pk}(a), \text{pair}(a, b)))$, but not for instance $\text{iknows}(\text{inv}(\text{pk}(a)))$. Note that there are already infinitely many facts, e.g. $\text{iknows}(\text{pair}(a, \text{pair}(a, \text{pair}(a, \dots))))$ is derivable. Most of the time this is however not an issue for the techniques we consider.

Let us now model the honest agents of the NSPK protocol. The first step is that A sends out the message $\text{crypt}(\text{pk}(B), \text{pair}(NA, A))$, but here is a problem: NA is supposed to be freshly created. The logical formulas we have been writing so far, however, do not have a notion of time, as we are building an over-approximation of what can ever happen. Thus we cannot directly work with fresh nonces here.

Suppose now that A would actually not create a fresh random number in every session and instead use always the same random number. Then the intruder would learn this number in any session where he is B , and thus destroying secrecy. So this would be a too *coarse* over-approximation where all nonces are the same, so the intruder learns them. Let us refine a bit: suppose every has one single nonce for every other agent B ; thus A is using in every session with the same B also the same nonces. We can thus write $na(A, B)$ for the nonce that A uses as na to B , and thereby make all nonces a function of the agent names. This gives us the following formula:

```

nspk1(A: User, B: User)
=> iknows(crypt(pk(B), pair(na(A, B), A)));

```

Thus the intruder knows the message that every agent in role A can produce as a first step for any agent in role B . He can decrypt this message, however, only if he is B , i.e., he learns $na(A, i)$ for every agent A . These are exactly the nonces where he is the intended recipient.

Note that the abstract $na(A, B)$ for the fresh nonce NA is still very coarse; we will not be able to check that the protocol is safe against replay attacks for instance, since all freshness is lost. However for secrecy this is fine enough. In fact we can now specify a violation of secrecy by the following rule:

```

secrecy1(A: Honest, B: Honest)
iknows(na(A, B)) => attack;

```

This means that it is an attack, if the intruder ever finds out the nonce $na(A, B)$ of two *honest* agents A and B . This indeed holds so far.

Now we can model how B can receive a message of the first step, i.e., of the form $\{\text{p}\}_{\text{pk}(A)} \text{pair}(NA, A)$ and send the second step, i.e., $\text{crypt}(\text{pk}(A), \text{pair}(NA, NB))$ as an answer. Here, B does not have any “control” over the value NA , he will accept any term here, so let us model NA as an untyped variable. NB however, is again freshly created. We use the exact trick as before and make this a function $nb(B, A)$, so we get:

```

nspk2(A: User, B: User, NA: untyped)
iknows(crypt(pk(B), pair(NA, A))) =>
iknows(crypt(pk(A), pair(NA, pair(nb(B, A), B))));

```

The intruder can decrypt the answer message if $A = i$, thus the intruder learns $nb(B, i)$ for every agent B – as before this is alright, since these are the nonce created by B for i . If $A = i$, he also

can analyze NA , however, no honest agent will claim the name of the intruder in the incoming message, so the intruder must have constructed that himself, and thus, he already had NA before.

Again we can make a secrecy goal for the nonce of B :

```
secrecy2(A: Honest , B: Honest)
iknows(nb(B,A)) => attack;
```

Next, A is waiting for a message of the form $\{NA, NB\}_{pk(A)}$ where NB can be anything and NA is the nonce that A created earlier for B . Then she will respond with the third step of the protocol $\{NB\}_{pk(B)}$. To model that NA must be the nonce that A has sent earlier, we can use again the abstraction: it must be of the form $na(A, B)$. Thus we have the following rule:

```
nspk3(A: User , B: User , NB: untyped)
iknows(crypt(pk(A),pair(na(A,B),NB))) =>
iknows(crypt(pk(B),NB));
```

Now the fact *attack* is derivable:

- From nspk1 the intruder can derive $crypt(pk(i), pair(na(a, i), a))$ and obtain $na(a, i)$.
- He can then construct $crypt(pk(b), pair(na(a, i), a))$.
- Using this message with nspk2, the intruder can then derive $crypt(pk(a), pair(na(a, i), nb(b, a)))$. Note that nothing in here says not explicitly who b is, but you can see it in the abstractions: a constructed $na(a, i)$ for i , and b constructed $nb(b, a)$ for a .
- Using this message with nspk3, the intruder gets $crypt(pk(i), nb(b, a))$.
- From that he gets $nb(b, a)$ which is the nonce of two honest agents, thus it is an attack (secrecy2).

This is the classical attack of Lowe we have discussed previously. Let us now change the protocol according to Lowe's suggestion, i.e., the second message additionally contains the name of B , i.e. $crypt(pk(B), pair(NA, pair(NB, B)))$.

The full specification is now:

```
Problem: nspk;

Types:
Honest      = {...};
Dishonest   = {...};
User        = Honest ++ Dishonest;

Sets:
dummy(User, User); % not using this here, just for syntax

Functions:
public crypt/2, pair/2, pk/1;
private inv/1, na/2, nb/2;

Facts:
iknows/1, attack/0, secret/3;

Rules:

users(A: User)
=> iknows(A);
```

```

publickeys(A: User)
=> iknows(pk(A));

privatkeys(D: Dishonest)
=> iknows(inv(pk(D)));

asymenc(M1: untyped, M2: untyped)
iknows(crypt(M1, M2)).iknows(inv(M1)) => iknows(M2);

asymdec(M1: untyped, M2: untyped)
iknows(M1).iknows(M2) => iknows(crypt(M1, M2));

proj(M1: untyped, M2: untyped) iknows(pair(M1, M2))
=> iknows(M1).iknows(M2);

pair(M1: untyped, M2: untyped)
iknows(M1).iknows(M2) => iknows(pair(M1, M2));

nspk1(A: User, B: User)
=> iknows(crypt(pk(B), pair(na(A, B), A)));

nspk2(A: User, B: User, NA: untyped)
iknows(crypt(pk(B), pair(NA, A))) =>
iknows(crypt(pk(A), pair(NA, pair(nb(B, A), B))));

nspk3(A: User, B: User, NB: untyped)
iknows(crypt(pk(A), pair(na(A, B), pair(NB, B)))) =>
iknows(crypt(pk(B), NB));

secrecy1(A: Honest, B: Honest)
iknows(na(A, B))
=> attack;

secrecy2(A: Honest, B: Honest)
iknows(nb(B, A))
=> attack;

```

In the declaration of Honest and Dishonest, we are using here a special feature of AIF_ω , the \dots that allows us to define an infinite set of new constants, instead of a finite enumeration, so we have now an infinite set of honest and dishonest participants.

We feed this specification into AIF_ω which translates it for the tool ProVerif (or other solvers for this kind of Horn clauses) and within seconds, we get the result: “goal unreachable: attack:”, meaning the fact *attack* is no longer derivable from our specification.

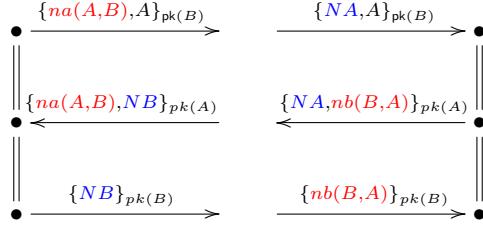
14.2 Two Abstractions

We can actually distinguish two “abstractions” we have made in the above example:

- We have abstracted the fresh nonces into functions like $na(A, B)$ and $nb(B, A)$. This gives us essentially a finite number of nonces (whenever the number of honest agents is finite).
- We have completely disregarded the transition system: there is no notion of state anymore, but we rather talk only about what messages the intruder may obtain in any number of runs.

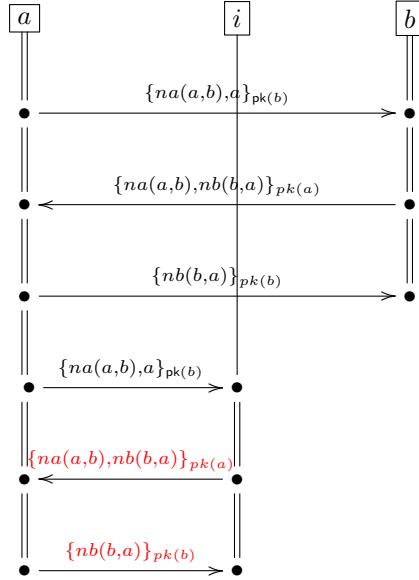
Formally, these two abstractions can be seen as a so-called *Galois-connection* [4].

Let us first look at the abstraction of the fresh data in isolation. We can see this actually as a transformation of the original protocol. For the NSPK example this means replacing in the original strands for role A and role B the freshly created variables with functions, obtaining the following two strands:



It is relatively easy to see that whatever protocol execution was possible with the original (“concrete”) protocol, is still possible with the new (“abstract”) protocol: take any execution of the original one, and replace all fresh nonces with the abstract ones, you get an execution of the new protocol. In this sense the abstraction is *sound*: we do not lose any execution of the original protocol, so if we can prove the new one to be correct, so is the old one.

The inverse however is not true: the new protocol has executions that the original one does not have:



Here the intruder, having observed one session between A and B , can play B in any future session. The protocol in this abstract model is therefore trivially vulnerable to replay, while that is not an issue in the original protocol.

This also means that not all goals can easily be applied to the new protocol. For instance authentication uses negation: it is an attack if there is request-event and *no* corresponding witness-event. Imagine, that in the original protocol a state is reachable where only the following witness and request events have occurred:

$$witness(a, b, n_1).request(b, a, n_2)$$

i.e., a violation of authentication, and suppose both nonces n_1 and n_2 are created by a for communication with b , so they get abstracted into the same constant $na(a, b)$. Then, the new protocol would not have here a violation of authentication.

In fact, it is hard to deal with authentication (and freshness) in this abstract model, but it is possible with some further refinements. Let us however only deal with secrecy in the following, since a violation of secrecy is only described by the positive condition that the intruder can produce a secret, and thus works fine with the over-approximation.

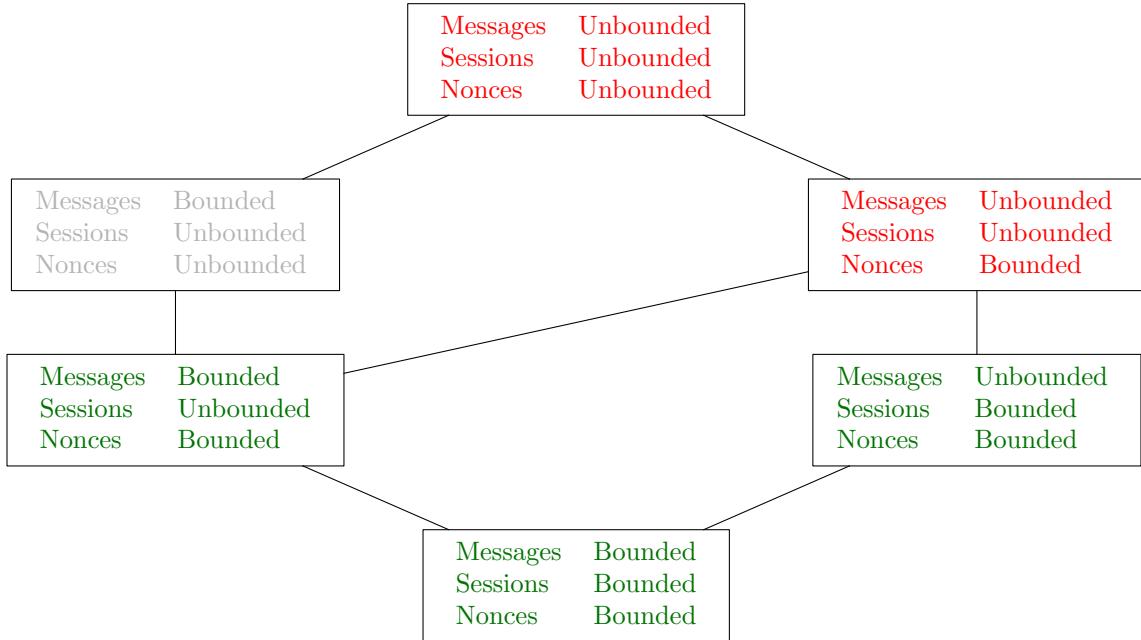
It has been shown that for secrecy goals, it is sufficient to consider only a fixed number of agents, e.g. $\{a, b, i\}$ [8]. Sufficient here means that for every attack with more agents, there is an attack with only these. Then we get also a fixed number of nonces, namely $na(A, B)$ and $nb(B, A)$ for every $A, B \in \{a, b, i\}$, and have removed the infinity of the nonces, even though we can consider infinitely many sessions.

Still, if the intruder is unbounded, this creates an infinite set of reachable states, since agents still have variables for the nonces from the other parties, and thus the intruder can create an arbitrary message and use it as a nonce. Let us therefore briefly consider another restriction:

Definition 29 (Typed Model). *In a typed model all variables have a type (e.g. nonce) and we allow only instantiation of variables with terms of the correct type.*

Then for instance, the variables NA and NB can only be instantiated with the nonces $na(A, B)$ or $nb(B, A)$ for some $A, B \in \{a, b, i\}$. This forbids the intruder to send *ill-typed* messages. Together with the finite number of nonces, there are now only finitely many terms that can be constructed, sent, and received.

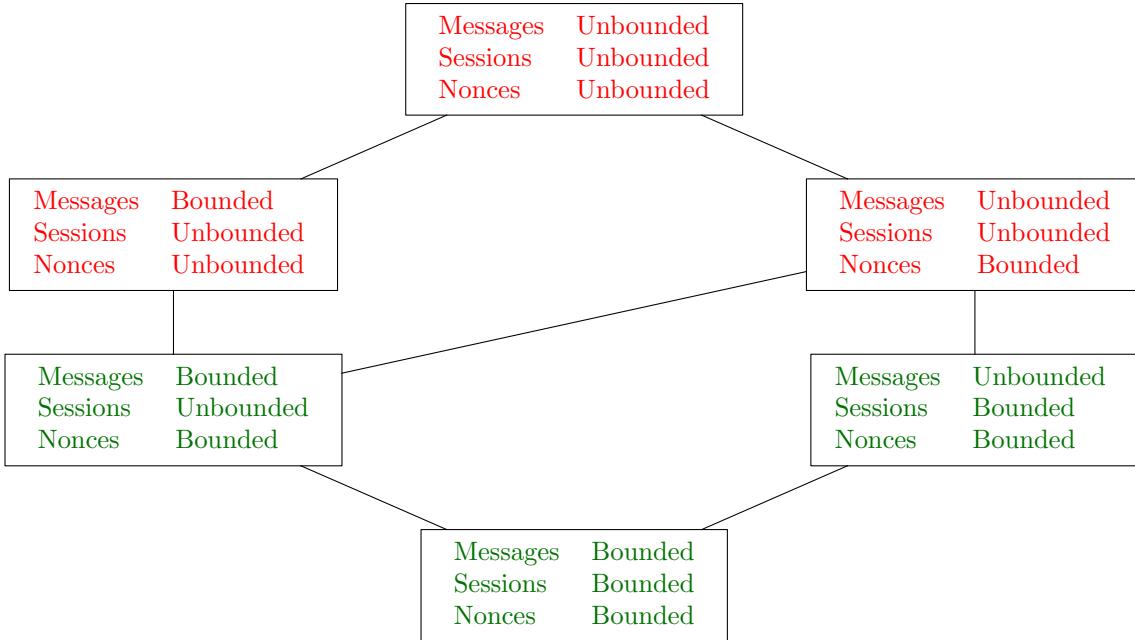
With these restrictions – bounded nonces and bounded messages – it is actually possible to decide secrecy goals for protocols and we thus have:



To complete our picture, we mention the following result that is similar to the first undecidability proof from the previous section:

Theorem 10 ([13]). *For an unbounded number of sessions and an unbounded number of nonces, protocol security is undecidable, even when bounding messages.*

This gives finally:



The conclusion is thus: for decidability, we may have either unbounded messages or unbounded sessions (with bounded nonces), but not both.

Part III

Advanced Topics

15 Channels and Composition

We now introduce a notion of *channels* that allow us to describe complex systems as a composition or layering of several simpler systems. A standard example can be the log-in at a bank. The user (resp. the user's browser) establishes a TLS connection with the bank. There could be now a password based authentication, or a login via a single-sign on solution like NemID. The latter is a protocol between the Bank, the User, and the Identity provider (e.g. NemID) that is also running over TLS channels. Finally, after authentication the actual banking application is communicating with the user via the initial TLS channel. It is considerably easier to consider not the resulting composed protocol as a monolithic system, but rather consider the individual components, where TLS provides a secure channel and the other protocol assume a secure channel. We shall in particular clarify what it means to have a secure channel.

15.1 Bullet Notation

We start with a very notation for channels called the bullet-calculus [22] which was started as an abstract way to model cryptography as the properties of a communication channel. While the original application is a calculus about what channels can be achieved given a number of existing channels, we will use it in AnB as a notation for channels over which messages are sent.

There are basically three kinds of channels; let us first give an intuition before we come to the formal definition.

- $A \bullet \rightarrow B$: A sends a message on an *authentic* channel to B , so B gets the guarantee that the message comes from A (but there is no guarantee that the message is confidential).

- $A \xrightarrow{\bullet} B$: A sends a message on a *confidential* channel to B , so only B can read it (but no guarantee that it comes from A)¹⁶
- $A \xrightarrow{\bullet\bullet} B$: A sends a message on a *secure* channel that is both authentic and confidential: A can be sure that only B can read it, and B can be sure it comes from A .

Note that for each channel there are lots of ways to achieve this in practice, e.g. asymmetric encryption and signatures.

Example 27 (NSL). *We can write the Needham Schroeder Public Key protocol with Lowe's fix (NSL) completely without cryptography, when we replace the public key encryption with confidential channels:*

$$\begin{array}{lll} A \rightarrow B : \{NA, A\}_{pk(B)} & A \xrightarrow{\bullet} B : NA, A \\ B \rightarrow A : \{NA, NB, B\}_{pk(A)} & B \xrightarrow{\bullet} A : NA, NB, B \\ A \rightarrow B : \{NB\}_{pk(B)} & A \xrightarrow{\bullet} B : NB \end{array}$$

One of the best illustrations of channels is actually Diffie-Hellman (compare Sec. 3.2). As we have seen, what Diffie-Hellman requires is an authenticated change of the “public keys” $\exp(g, X)$ and $\exp(g, Y)$. Thus:

$$\begin{array}{ll} A \xrightarrow{\bullet} B : \exp(g, X) \\ B \xrightarrow{\bullet} A : \exp(g, Y) \end{array}$$

This expresses the essence of Diffie-Hellman: the authenticated exchange of Diffie-Hellman public exponents, while we leave completely open how the authentication may happen:

- Cryptographic measures like digital signatures, symmetric encryption/MACs, asymmetric encryption and there are correspondingly many different protocols using these measures like the IPSEC protocols IKE, IKEv2, JFK, (Diffie-Hellman mode of) TLS, and many others.
- Relying on a trust third party.
- Relying on face to face meeting between friends, e.g. device-pairing is a protocol between two mobile devices where an un-authenticated Diffie-Hellman exchange is performed between the two devices and the owners of the devices are presented a fingerprint of the two public exponents, so they can compare and approve the exchange.

Channels can also be thought of as a goal: the goal of Diffie-Hellman for instance is to establish a secure channel between two parties:

$$\begin{array}{ll} A \xrightarrow{\bullet} B : \exp(g, X) \\ B \xrightarrow{\bullet} A : \exp(g, Y) \\ A \rightarrow B : \{A, B, \text{MsgA}\}_{\exp(\exp(g, X), Y)} \\ A \rightarrow B : \{B, A, \text{MsgB}\}_{\exp(\exp(g, X), Y)} \end{array}$$

Goals :

$$\begin{array}{ll} A \xrightarrow{\bullet\bullet} B : \text{MsgA} \\ B \xrightarrow{\bullet\bullet} A : \text{MsgB} \end{array}$$

We have added that A and B exchange two payload messages MsgA and MsgB . As a goal we have specified that these payload messages are like transmissions on a secure channel, i.e., the goal $A \xrightarrow{\bullet\bullet} B : \text{MsgA}$ means that both

- the message is authentically transmitted: B weakly authenticates A on MsgA , and
- the message is confidentially transmitted: MsgA secret between A, B .

¹⁶We use the words confidential and secret as synonymous; the term *confidential channel* had already been established in the literature and thus we did not change it here.

Thus, we can summarize Diffie-Hellman very generally by saying: *Diffie-Hellman creates secure channels from authentic channels*. A very general definition of public key cryptography is: *any scheme that allows us to create secure channels from authentic ones*. Observe that this definition completely avoids even talking about any technical aspect like public and private keys.

15.2 A Cryptographic Implementation

A question is what *authentication*, *confidential*, and *secure* channel should precisely mean. In fact there are several quite different answers. We define them here using asymmetric cryptography. There are many other ways to define them, e.g. describing by their behavior.

Assume every agent A has two key pairs:

- $\langle \text{ck}(A), \text{inv}(\text{ck}(A)) \rangle$ for asymmetric encryption,
- $\langle \text{ak}(A), \text{inv}(\text{ak}(A)) \rangle$ for digital signatures,

and assume that every agent knows the public keys $\text{ck}(A)$ and $\text{ak}(A)$ of every other agent A . We require that neither $\text{ck}(\cdot)$ nor $\text{ak}(\cdot)$ occur in the AnB specification.

Then we can implement authentic channels by signing, confidential channels by encryption, and secure channels by both signing and encrypting:

Definition 30. *Channel implementation using asymmetric cryptography:*

$$\begin{aligned} A &\xrightarrow{\bullet} B : M \quad \text{for } A \rightarrow B : \{B, M\}_{\text{inv}(\text{ak}(A))} \\ A &\rightarrow \bullet B : M \quad \text{for } A \rightarrow B : \{M\}_{\text{ck}(B)} \\ A &\xrightarrow{\bullet \rightarrow \bullet} B : M \quad \text{for } A \rightarrow B : \{\{B, M\}_{\text{inv}(\text{ak}(A))}\}_{\text{ck}(B)} \end{aligned}$$

This ensures the basic properties of channels:

- Only A can produce messages on the channel $A \xrightarrow{\bullet} B$, since nobody else knows $\text{inv}(\text{ak}(A))$.
- Only B can read messages on the channel $A \rightarrow \bullet B$, since nobody else knows $\text{inv}(\text{ck}(B))$.
- Both restrictions hold on a secure channel.

Note that the intruder can still intercept and replay messages – the channels we have defined do neither guarantee resilience against network disruption nor do they guarantee freshness.

A question is why we would need to include the name of the intended recipient in authentic and secure channels implementations. For the secure channels consider the protocol

$$\begin{aligned} A &\rightarrow B : \{\{MsgA\}_{\text{inv}(\text{ak}(A))}\}_{\text{ck}(B)} \\ \text{Goals :} \\ A &\xrightarrow{\bullet \rightarrow \bullet} B : MsgA \end{aligned}$$

Recall that the goal stands for **B authenticates A on $MsgA$ and $MsgA$ secret between A,B**. As an exercise, the reader (manually or with OFMC) should try to find the attack against the authentication goal of this protocol. In fact this is a “classical” mistake in protocol design that appears again and again over the years.

Authentic channels must guarantee that authenticate the information who is the intended recipient of the message. This is different from ensuring that only the intended recipient can *read* the message.

15.3 Channels as Assumptions – Channels as Goals

Many protocols we consider can be regarded as protocols for establishing a channel. For instance TLS is essentially a key-exchange (handshake) that establishes symmetric keys for communication between a client and a server, including a transmission protocol that uses these keys to encrypt arbitrary *Payload* messages with them (e.g. from an email application). We can thus also allow the definition of protocol goals using channels as we did it in previous examples, where authentic channel means a weak authentication goal, confidential channel means a secrecy goal, and secure channel means both goals.

Note that we have used only authentication here: the reason is that we want a correspondence between channels as *assumptions* (when the protocol transmits messages of authentic, confidential, or secure channels) and channels as *goals*. When we look at authentic channels again (and similar secure channels) in our cryptographic implementation, there is no protection against replay built into the channels: an intruder can record any message and replay it as is later. Thus, if replay prevention is needed, it remains the responsibility of the protocol that uses the channel to prevent replay. One may indeed make a variant of authentic channels (and similarly of secure channels) that also include replay protection (e.g. using timestamps, sequence numbers, or challenge response). In fact, one may define a variety of different channel properties.

15.4 Compositionality

The relation between channels as assumption and channels as goals gives rise to an interesting question:

- Given a protocol P_1 has as goal to establish a certain channel type C , e.g. TLS with the goal to establish a secure channel.
- Given another protocol P_2 assumes such a channel C , e.g. a web-application for an online email box.
- Suppose also that both P_1 and P_2 have been verified individually.
- Suppose finally, that we “plug” P_1 into P_2 together, i.e., running the traffic of P_2 over the channel established by P_1 . Let us denote this composition as $P_2[P_1]$, pronounced *running P_2 over P_1* .
- Is the resulting protocol $P_2[P_1]$ correct?

Example 28. Let P_1 be the following protocol:

$$\frac{\begin{array}{l} A \rightarrow s : A, B, \text{Payload}, \text{mac}(sk(A, s), A, B, \text{Payload}) \\ s \rightarrow B : A, B, \text{Payload}, \text{mac}(sk(B, s), A, B, \text{Payload}) \end{array}}{\text{Goal} : A \bullet\rightarrow B : \text{Payload}}$$

This protocol establishes only an authentic channel. It assumes that A and a trusted server s share a secret symmetric key $sk(A, s)$, and similarly B has key $sk(B, s)$ with s . Since only authentication is the goal, the Payload is actually not even encrypted, we just build a mac (Message Authentication Code) that we model like a hash-function that receives as one argument a secret key. This mac can thus be checked by anybody who knows the respective key. In this way, A can authentically send the Payload message via s to B , since s is honest.

Let P_2 be the following protocol:

$$\frac{\begin{array}{l} A \bullet\rightarrow B : \exp(g, X) \\ B \bullet\rightarrow A : \exp(g, Y) \\ A \rightarrow B : \{\text{ApplicationPayload}\}_{\exp(\exp(g, X), Y)} \end{array}}{\text{Goal} : A \bullet\rightarrow\bullet B : \text{ApplicationPayload}}$$

Here we have an application protocol uses Diffie-Hellman to first establish a secure shared key between A and B to transmit a more high-level payload (called *ApplicationPayload*) securely. It assumes authentic channels from A to B and vice-versa, but it is independent of how this application is established.

The composition $P_2[P_1]$ is the following protocol:

$$\begin{array}{l}
 A \rightarrow s : A, B, \exp(g, X), \text{mac}(\text{sk}(A, s), A, B, \exp(g, X)) \\
 s \rightarrow B : A, B, \exp(g, X), \text{mac}(\text{sk}(B, s), A, B, \exp(g, X)) \\
 B \rightarrow s : B, A, \exp(g, Y), \text{mac}(\text{sk}(B, s), B, A, \exp(g, Y)) \\
 s \rightarrow A : B, A, \exp(g, Y), \text{mac}(\text{sk}(A, s), B, A, \exp(g, Y)) \\
 A \rightarrow B : \{\text{ApplicationPayload}\}_{\exp(\exp(g, X), Y)} \\
 \hline
 \text{Goal} : A \bullet \rightarrow \bullet B : \text{ApplicationPayload}
 \end{array}$$

Note that this protocol no longer assumes any channels, i.e., it is completely implemented now. It is more complex than the original protocols, in the sense it is harder to read and understand, and also harder to analyze automatically.

15.5 Pseudonymous Channels

Consider again the TLS handshake protocol from Section 3:

```

Protocol: TLS
Types: Agent A,B,s;
       Number NA,NB,Sid,PA,PB,PMS;
       Function pk,hash,clientK,serverK,prf
Knowledge: A: A, pk(A), pk(s), inv(pk(A)), {A, pk(A)} inv(pk(s)), B,
           hash, clientK, serverK, prf;
           B: B, pk(B), pk(s), inv(pk(B)), {B, pk(B)} inv(pk(s)),
           hash, clientK, serverK, prf
Actions:
A->B: A,NA,Sid,PA
B->A: NB,Sid,PB,
       {B, pk(B)} inv(pk(s))
A->B: {A, pk(A)} inv(pk(s)),
       {PMS}pk(B),
       {hash(NB,B,PMS)} inv(pk(A)),
       {|hash(prf(PMS,NA,NB),A,B,NA,NB,Sid,PA,PB,PMS)|}
           clientK(NA,NB,prf(PMS,NA,NB))
B->A: {|hash(prf(PMS,NA,NB),A,B,NA,NB,Sid,PA,PB,PMS)|}
           serverK(NA,NB,prf(PMS,NA,NB))
Goals:
  B authenticates A on prf(PMS,NA,NB)
  A authenticates B on prf(PMS,NA,NB)
  prf(PMS,NA,NB) secret between A,B

```

This is in fact not the most common way to use TLS, because it requires the client A to own a certificate of its public key, what is simply modeled here as the message $\{A, \text{pk}(A)\}_{\text{inv}(\text{pk}(s))}$, i.e., a trusted server s (the certificate authority) has signed the statement that A has public key $\text{pk}(A)$. Normally, only the server B will have such a certificate, but not the ordinary Internet user who runs TLS in role A . Essentially, the protocol is simply executed simply without A 's certificate (while B 's certificate must always be present):

```

Protocol: TLS
Types: Agent A,B,s;
       Number NA,NB,Sid,PA,PB,PMS;
       Function pk,hash,clientK,serverK,prf

```

```

Knowledge: A: A, pk(A), pk(s), B,
               hash, clientK, serverK, prf;
B: B, pk(B), pk(s), inv(pk(B)), {B, pk(B)} inv(pk(s)),
   hash, clientK, serverK, prf

Actions:
A->B: A, NA, Sid, PA
B->A: NB, Sid, PB,
       {B, pk(B)} inv(pk(s))
A->B: A, PK,
       {PMS} pk(B),
       {hash(NB, B, PMS)} inv(PK),
       {| hash(prf(PMS, NA, NB), A, B, NA, NB, Sid, PA, PB, PMS) |}
       clientK(NA, NB, prf(PMS, NA, NB))
B->A: {| hash(prf(PMS, NA, NB), A, B, NA, NB, Sid, PA, PB, PMS) |}
       serverK(NA, NB, prf(PMS, NA, NB))

Goals:
#B authenticates A on prf(PMS, NA, NB)
A authenticates B on prf(PMS, NA, NB)
#prf(PMS, NA, NB) secret between A, B

```

We have here chosen to model that A has no key with a certificate as A creating a fresh public key PK and sending it to B .

Obviously, without the client certificate, this protocol does not allow B to authenticate A , and thus both secrecy and part of authentication do no longer hold (thus commented in the listing). In other words, the user who claims to be A (and the creator/owner of PK) could in fact be anybody. So what does it even help to have this exchange without client authentication in the first place?

The answer is, we get slightly more than the remaining authentication goal (that A can be sure about the server). In fact, we still get something like a secure channel: the intruder cannot read or impersonate any messages in a session between an honest client A and an honest server B . This is since only the creator of PK (who knows $inv(PK)$) and B know the resulting keys of this exchange. So the intruder can start any number of sessions with fresh public keys for PK and under any agent name, but only read in these sessions (and in the sessions where he is B), but he cannot interfere with sessions between an honest client who created PK and an honest server (since he does not have $inv(PK)$ then).

We can actually regard the key PK as a *pseudonym* that A has chosen, nobody else can claim the ownership of that pseudonym, since that requires to know $inv(PK)$, and TLS thus establishes a secure channel between the owner of PK (whoever it is) and B .

We write for this kind of channel $[A] \bullet\rightarrow\bullet B$ (respectively, $B \bullet\rightarrow\bullet [A]$) to indicate that A is not authenticated with respect to her real name, but only with respect to a some pseudonym. We could express this for TLS without client authentication as follows:

TLS handshake

$$\frac{\begin{array}{c} \dots \\ A \rightarrow B : \{Payload_A\}_{clientK(NA,NB,prf(PMS,NA,NB))} \\ B \rightarrow A : \{Payload_B\}_{serverK(NA,NB,prf(PMS,NA,NB))} \end{array}}{\begin{array}{l} \text{Goal : } [A] \bullet\rightarrow\bullet B : Payload_A \\ \quad B \bullet\rightarrow\bullet [A] : Payload_B \end{array}}$$

This is sometimes also called sender/receiver invariance: B cannot be sure about A 's real identity, but that it is the *same entity* in several transmissions (namely the owner of a certain key pair).

This kind of channel is good enough for many applications such as transmitting credit card data: $[A] \bullet\rightarrow\bullet B : \text{Order \& Credit Card Data}$ The intruder can also make orders, or, as a

dishonest merchant B receive credit card data, but cannot see the credit card data from an honest A sent to an honest B .

The secure pseudonymous channel is also good enough for a login protocol:

$$\frac{[A] \xrightarrow{\bullet} B : A, \text{password}(A, B) \\ B \xrightarrow{\bullet} [A] : \text{Payload}}{\text{Goal} : B \xrightarrow{\bullet} A : \text{Payload}}$$

where $\text{password}(A, B)$ is A 's password at server B . We establish a “classical” secure channel in two steps:

1. We establish a secure pseudonymous channel $[A] \xrightarrow{\bullet} B$ using TLS without client authentication.
2. We use this channel to authenticate the client by a shared secret (which possibly has low entropy).

Further replies (e.g. the data of client A stored on server B) are now bound to this authentication. For more on passwords, please also consider Section 16 on guessing attacks.

16 Guessing Attacks

OFMC implements support for checking guessing attacks. This is relevant for protocols that rely on potentially low-entropy secrets, like passwords. Low entropy means that bits of the secret are far from randomly chosen; thus for a brute-force attack (trying out all possible keys), the intruder may be able to restrict the search space to a few million guesses and still have a good success rate (passwords that fall into the restricted space). This restricted search space is often called a *dictionary*, i.e., a collection of popular passwords and variants thereof, that the intruder will try out.

In principle, we cannot prevent against somebody trying to guess passwords (as they can try to guess good cryptographic keys), but one can do a few things to mitigate the risks. First, there is online guessing: servers accepting a password login should pause after each time a wrong password entered before a new attempt can be made, in order to prevent an attacker trying out millions of guesses. In general it is not recommended to shut down after a number of failed attempts, because that opens the door for denial of service attacks.

But a second problem of guessing can be seen in the following example:

Protocol: PW

Types: Agent A, B;
Number NB;
Function pw, h;

Knowledge:

A: A, B, pw(A, B), h;
B: A, B, pw(A, B), h;

Actions:

B -> A: NB
A -> B: h(pw(A, B), NB)

Goals:

B authenticates A on NB

This is a very simple handshake for authentication, where the server B first sends a fresh nonce NB and A answers with a hash of that nonce and the password. (We assume here in fact that h is a perfect random function, so it does not leak the password itself.)

If the intruder observes such an exchange, he can try to guess the password *offline*: he has seen for a concrete nonce nb and a response $h(pw(a, b), nb)$ containing the password of between particular a and b . Now for every password p in his dictionary, the intruder constructs $h(p, nb)$ and compares the result with observed hash. When $p = pw(a, b)$, the hash is the same and he knows what is the right password. Even though we have not given a secrecy goal on the password, the intruder is now able to authenticate as a to b , and has thus a successful attack against the protocol.

Is this not just secrecy? One may consider to just say “the password is simply not a secret” and give the intruder the entire function pw . However, that is a bit too pessimistic: that immediately rules out any protocols that rely on secret passwords, e.g., the entire setup of a TLS channel that is then used for a password-based login as discussed in the previous section. As a simplified version of this password over TLS consider the following protocol:

Protocol: PW

Types: Agent A, B;
Number NB;
Function pw, pk;

Knowledge:

A: A, B, pw(A, B), pk(B);
B: A, B, pw(A, B), pk(B), inv(pk(B));

Actions:

A->B: { A, pw(A, B), K }pk(B)
B->A: {| NB |}K

Goals:

B authenticates A on K
A authenticates B on NB
K secret between A, B

Here we make use of a public key $pk(B)$ that A knows in advance: we encrypt the password with $pk(B)$ and send it together with a fresh session key K that A has generated and that we can use for further communication.

In fact, this protocol is secure against offline-guessing: while the password is guessable, the fresh random key K is not, and thus the intruder cannot guess the first message of the protocol, and he also cannot decrypt it, because he lacks the private key $inv(pk(B))$ for that.

Guessable Secrecy Goal To properly distinguish between protocols that are vulnerable to guessing (like the first example) and ones that are not (like the second example), OFMC has a special goal to check for guessing attacks inspired by [20, 12].

This can be used by declaring an additional goal – the last line of the following listing:

Protocol: PW

Types: Agent A, B;
Number NB;

```
Function pw,h;
```

Knowledge :

```
A: A,B,pw(A,B),h;
B: A,B,pw(A,B),h;
```

Actions :

```
B->A: NB
A->B: h(pw(A,B),NB)
```

Goals :

```
B authenticates A on NB
pw(A,B) guessable secret between A,B
```

which leads to the attack

GOAL:

```
guesswhat
```

BACKEND:

```
Open-Source Fixedpoint Model-Checker version 2020
```

STATISTICS:

```
TIME 7 ms
```

```
parseTime 0 ms
```

ATTACK TRACE:

```
i -> (x20,1): x205
(x20,1) -> i: h(pw(x20,x25),x205)
i -> (i,17): h(guessPW,x205)
i -> (i,17): h(guessPW,x205)
```

As the reader might guess this is implemented using the secrecy mechanism in OFMC. In fact, whenever a message is produced by an honest agent that contains the password, this triggers a new secrecy goal, namely to produce the same message with the password replaced by *guessPW*, which is a special constant we give the intruder initially. This acts as a witness that the intruder could produce the same message by guessing.

This is in fact the simplest case: that the intruder guesses the entire message. There are however special cases depending on the top-level operator of the message that contains the password

- When the message containing the password is a pair $\langle m_1, m_2 \rangle$, then it is sufficient if the intruder can make a guessing attack on the m_i that contains the password, so we recursively check that m_i .
- When the message is a symmetric encryption $\{m\}_k$ then we check whether k contains the password. If so, for a guessing attack we consider it sufficient that the intruder can perform a guessing attack on k (because we assume he can check for correct decryption of k). (If k does not contain the password, then the entire message needs to be guessed.)
- For asymmetric encryption we do not have a special case (since public and private keys are not password-based). Note however that for instance a message like

$$\{pw(A,B)\}_{pk}$$

is vulnerable if pk is known and $pw(A,B)$ is guessable. The point is that public-key encryption should be randomized, and failure to do so explicitly will now lead to guessing attacks. Including a fresh key like in the previous TLS-style example is sufficient to thwart the attack.

- Exponentiation currently also does not include a special case since the AnB interface does not support inverses of exponents, thus the intruder cannot decompose exponentiations with known exponents. For the same reason all user-declared functions are not a special case either.
- XOR is not support currently for guessing attacks and will lead to OFMC stoping with an error when encountered at guessing.

References

- [1] O. Almousa, S. Mödersheim, and L. Viganò. Alice and bob: Reconciling formal models and implementation. In *Festschrift in honor of Pierpaolo Degano*, 2015.
- [2] AVISPA. The Intermediate Format. Deliverable D2.3, Automated Validation of Internet Security Protocols and Applications (AVISPA), 2003. <http://www.avispa-project.org/delivs/2.3/d2-3.pdf>
- [3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [4] B. Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, 2005.
- [5] F. Böhl, V. Cortier, and B. Warinschi. Deduction soundness: prove one, get five for free. In *CCS 2013*, 2013.
- [6] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.
- [7] Y. Chevalier and M. Rusinowitch. Compiling and securing cryptographic protocols, 2010.
- [8] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Sci. Comput. Program.*, 50(1-3):51–71, 2004.
- [9] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24(8):533–536, 1981.
- [10] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [11] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198 – 208, Mar. 1983.
- [12] P. H. Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 363–379. Springer, 2004.
- [13] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, July 1999.
- [14] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.
- [15] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, (2/3):191–230, 1999.

- [16] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In M. Parigot and A. Voronkov, editors, *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160. Springer, 2000.
- [17] C. Kirchner and H. Kirchner. *Rewriting, Solving, Proving*. 1999–2006. Available at <https://wiki.bordeaux.inria.fr/Helene-Kirchner/lib/exe/fetch.php?media=wiki:rsp.pdf>.
- [18] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In T. Margaria and B. Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
- [19] G. Lowe. A hierarchy of authentication specifications. pages 31–43. IEEE Computer Society Press, 1997.
- [20] G. Lowe. Analysing protocol subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.
- [21] U. Maurer and R. Renner. Abstract cryptography. In *ICS*, 2011.
- [22] U. Maurer and P. Schmid. A calculus for security bootstrapping in distributed systems. *Journal of Computer Security*, 4(1):55–80, 1996.
- [23] S. Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares’09*, 2009.
- [24] S. Mödersheim and A. Bruni. Aif-omega: Set-based protocol abstraction with countable families. In *Principle of Security and Trust (POST)*, 2016. Tool webpage: <http://imm.dtu.dk/~samo/aifom.html>.
- [25] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In M. Backes and P. Ning, editors, *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 337–354. Springer, 2009.
- [26] S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
- [27] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
- [28] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.