Secure and Dependable Systems

Jürgen Schönwälder

Jacobs University Bremen

August 22, 2020

https://cnds.jacobs-university.de/courses/sads-2020

Part: Administrivia

Course Objectives and Grading

Topics and learning goals

- This course introduces (formal and non-formal) methods for analyzing and assuring dependability and security of software systems.
- Definition of concepts such as dependability, reliability, availability, safety, and security of software systems
- Introduction of paradigms of safety/security analysis such as
    - testing techniques (code coverage),
    - program verification (Hoare calculus).
- Introduction into cryptography and its application for building secure systems.
- Discussing aspects of system security, such as authentication mechanisms and protocols, authorization mechanisms, concepts of anonymity and privacy, classic security protocols and mechanisms widely used.

Grading Scheme

- Assignments
  - Learning by solving assignments
  - Test whether you can apply concepts learned
  - Prepare yourself for the final exam
- Final examination (100%)
  - Covers the whole course
  - Closed book (and closed computers)
  - Cheat sheet (handwritten A4 single sided) allowed

Teaching and learning strategy

- Homework assignments: Reinforce and apply what is taught in class
- Assignments will be small individual assignments (but may take time to solve)
- Consider forming study groups. It helps to discuss questions and course material in study groups or to explore different directions to solve an assignment.
- You can audit the course. To earn an audit, you have to pass an oral interview about key concepts introduced in the course at the end of the semester.

Organizational aspects and tutorials

- All assignments will be linked to the course web page.
- Solutions for assignments can be submitted using the Moodle system.
  `https://moodle.jacobs-university.de/`
- Feedback will be accessible via the Moodle system as well.
- Teaching assistant will be available to discuss course topics and or questions related to homeworks or to get help during exam preparations.

Rules of the Game

Code of Academic Integrity

- Jacobs University has a “Code of Academic Integrity”
  - this is a document approved by the Jacobs community
  - you have signed it during enrollment
  - it is a law of the university, we take it seriously
- It mandates good behaviours from faculty and students and it penalizes bad ones:
  - honest academic behavior (e.g., no cheating)
  - respect and protect intellectual property of others (e.g., no plagiarism)
  - treat all Jacobs University members equally (e.g., no favoritism)
- It protects you and it builds an atmosphere of mutual respect
  - we treat each other as reasonable persons
  - the other’s requests and needs are reasonable until proven otherwise
  - if others violate our trust, we are deeply disappointed (may be leading to severe and uncompromising consequences)

Academic Integrity Committee (AIC)

- The Academic Integrity Committee is a joint committee by students and faculty.
- Mandate: to hear and decide on any major or contested allegations, in particular,
  - the AIC decides based on evidence in a timely manner,
  - the AIC makes recommendations that are executed by academic affairs,
  - the AIC tries to keep allegations against faculty anonymous for the student.
- Every member of Jacobs University (faculty, student, . . .) can appeal any academic integrity allegations to the AIC.

Cheating

- There is no need to cheat, cheating prevents you from learning.
- Useful collaboration versus cheating:
  - You will be required to hand in your own original code/text/math for all assignments
  - You may discuss your homework assignments with others, but if doing so impairs your ability to write truly original code/text/math, you will be cheating
  - Copying from peers, books or the Internet is plagiarism unless properly attributed
- What happens if we catch you cheating?
  - We will confront you with the allegation (you can explain yourself)
  - If you admit or are silent, we impose a grade sanction and we notify the student records office
  - Repeated infractions to go the AIC for deliberation
- Note: Both active (copying from others) and passive cheating (allowing others to copy) are penalized equally

Culture of Questions, Answers, and Explanations

- Answers to questions require an explanation even if this is not stated explicitly
    - A question like 'Does this algorithm always terminate?' can in principle be answered with 'yes' or 'no'.
    - We expect, however, that an explanation is given why the answer is 'yes' or 'no', even if this is not explicitly stated.
- Answers should be written in your own words
    - Sometimes it is possible to find perfect answers on Wikipedia or Stack Exchange or in good old textbooks.
    - Simply copying the answer of someone else is plagiarism.
    - Copying the answer and providing the reference solves the plagiarism issue but usually does not show that you understood the answer.
    - Hence, we want you to write the answer in your own words.
    - Learning how to write concise and precise answers is an important academic skill.

Culture of Interaction

- I am here to help you learn the material.
- If things are unclear, ask questions.
- If I am going too fast, tell me.
- If I am going too slow, tell me.
- Discussions in class are most welcome - don't be shy.
- Discussions in tutorials are even more welcome - don't be shy.
- If you do not understand something, chances are pretty high your neighbor does not understand either.
- Don't be afraid of asking teaching assistants or myself for help and additional explanations.

Resources

Study Material and Forums

- There is no required textbook.
- The slides and notes are available on the course web page.
  `http://cnds.jacobs-university.de/courses/sads-2019`
- We will be using Moodle and it hosts a forum for this course.
  `https://moodle.jacobs-university.de/`
- General questions should be asked on the Moodle forum.
  - Faster responses since many people can answer
  - Better responses since people can collaborate on the answer
- For individual questions, see me at my office (or talk to me after class).

Part: Introduction

Motivation

Can we trust computers?

- How much do you trust (to function correctly)
  - personal computer systems and mobile phones?
  - cloud computing systems?
  - planes, trains, cars, ships?
  - navigation systems?
  - communication networks (telephones, radios, tv)?
  - power plants and power grids?
  - banks and financial trading systems?
  - online shopping and e-commerce systems?
  - social networks and online information systems?
  - information used by insurance companies?
  - ...
- Distinguish between (i) what your intellect tells you to trust and (ii) what you trust in your everyday life.

Importance of Security and Dependability

- Software development processes are often too focused on functional aspects and user interface aspects (since this is what sells products).
- Aspects such as reliability, robustness against failures and attacks, long-term availability of the software and data, integrity of data, protection of data against unauthorized access, etc. are often not given enough consideration.
- Software failures can not only have significant financial consequences, they can also lead to environmental damages or even losses of human lifes.
- Due to the complexity of computing systems, the consequences of faults in one component are very difficult to estimate.
- Security and dependability aspects must be considered during all phases of a software development project.

Recent Computing Disasters

IoT Remove Control Light Bulbs 2018

Spectre: Vulnerability of the Year 2018

```
#define PAGESIZE 4096
unsigned char array1[16]                /* base array */
unsigned int array1_size = 16;          /* size of the base array */
int x;                                  /* the out of bounds index */
unsigned char array2[256 * PAGESIZE];   /* instrument for timing channel */

// ...

if (x < array1_size) {
    y = array2[array1[x] * PAGESIZE];
}
```

- Is the code shown above a vulnerability?

Spectre: Main Memory and CPU Memory Caches

- Memory in modern computing systems is layered
- Main memory is large but relatively slow compared to the speed of the CPUs
- CPUs have several internal layers of memory caches, each layer faster but smaller
- CPU memory caches are not accessible from outside of the CPU
- When a CPU instruction needs data that is in the main memory but not in the caches, then the CPU has to wait quite a while...

Spectre: Timing Side Channel Attack

- A side-channel attack is an attack where information is gained from the physical implementation of a computer system (e.g., timing, power consumption, radiation), rather than weaknesses in an implemented algorithm itself.
- A timing side-channel attack infers data from timing observations.
- Even though the CPU memory cache cannot be read directly, it is possible to infer from timing observations whether certain data resides in a CPU memory cache or not.
- By accessing specific uncached memory locations and later checking via timing observations whether these locations are cached, it is possible to communicate data from the CPU using a cache timing side channel attack.

Spectre: Speculative Execution

- In a situation where a CPU would have to wait for slow memory, simply guess a value and continue excution speculatively; be prepared to rollback the speculative computation if the guess later turns out to be wrong; if the guess was correct, commit the speculative computation and move on.
- Speculative execution is in particular interesting for branch instructions that depend on memory cell content that is not found in the CPU memory caches
- Some CPUs collect statistics about past branching behavior in order to do an informed guess. This means we can train the CPUs to make a certain guess.
- Cache state is not restored during the rollback of a speculative execution.

Spectre: Reading Arbitrary Memory

- Algorithm:
  1. create a small array `array1`
  2. choose an index `x` such that `array1[x]` is out of bounds
  3. trick the CPU into speculative execution (make it to read `array1_size` from slow memory and to guess wrongly)
  4. create another uncached memory array called `array2` and read `array2[array1[x]]` to load this cell into the cache
  5. read the entire `array2` and observe the timing; it will reveal what the value of `array1[x]` was
- This could be done with JavaScript running in your web browser; the first easy "fix" was to make the JavaScript time API less precise, thereby killing the timing side channel. (Obviously, this is a hack and not a fix.)

Dependability Concepts and Terminology

System and Environment and System Boundary

Definition (system, environment, system boundary)

A *system* is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena. The other systems are the *environment* of the given system. The *system boundary* is the common frontier between the system and its environment.

- Note that systems almost never exist in isolation.
- We often forget to think about all interactions of a system with its environment.
- Well-defined system boundaries are essential for the design of complex systems.

Components and State

Definition (components)

The structure of a system is composed out of a set of *components*, where each component is another system. The recursion stops when a component is considered atomic.

Definition (total state)

The *total state* of a given system is the set of the following states: computation, communication, stored information, interconnection, and physical condition.

Function and Behaviour

Definition (function and functional specification)

The *function* of a system is what the system is intended to do and is described by the *functional specification*.

Definition (behaviour)

The *behaviour* of a system is what the system does to implement its function and is described by a sequence of states.

Service and Correct Service

Definition (service)

The *service* delivered by a system is its behaviour as it is perceived by a its user(s); a user is another system that receives service from the service provider.

Definition (correct service)

Correct service is delivered when the service implement the system function.

Failure versus Error versus Fault

Definition (failure)

A *service failure*, often abbreviated as *failure*, is an event that occurs when the delivered service deviates from correct service.

Definition (error)

An *error* is the part of the total state of the system that may lead to its subsequent service failure.

Definition (fault)

A *fault* is the adjudged or hypothesized cause of an error. A fault is *active* when it produces an error, otherwise it is *dormant*.

Dependability

Definition (dependability - original)

Dependability is the ability of a system to deliver service than can justifiably be trusted.

Definition (dependability - revised)

Dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

- The revised definition provides a criterion for deciding if a system is dependable.
- Trust can be understood as a form of accepted dependance.

Dependability Attributes

Definition (dependability attributes)

Dependability has the following attributes:

- *Availability*: readiness to deliver correct service
- *Reliability*: continuity of correct service
- *Safety*: absence of catastrophic consequences on the user(s) and the environment
- *Integrity*: absence of improper system alterations
- *Maintainability*: ability to undergo modifications and repairs
- *Confidentiality*: absence of unauthorized disclosure of information

Dependability and Security

Definition (security)

Security is a composite of the attributes of confidentiality, integrity, and availability.

- The definition of dependability considers security as a subfield of dependability. This does, however, not reflect how research communities have organized themselves.
- As a consequence, terminology is generally not consistent. Security people, for example, talk about vulnerabilities while dependability people talk about dormant faults.

Fault Prevention

Definition (fault prevention)

Fault prevention aims at preventing the occurance or introduction of faults.

- Application of good software engineering techniques and quality management techniques during the entire development process.
- Hardening, shielding, etc. of physical systems to prevent physical faults.
- Maintenance and deployment procedures (e.g., firewalls, installation in access controlled rooms, backup procedures) to prevent malicious faults.

Fault Tolerance

Definition (fault tolerance)

Fault tolerance aims at avoiding service failures in the presence of faults.

- Error detection aims at detecting errors that are present in the system so that recovery actions can be taken.
- Recovery handling eliminates errors from the system by rollback to an error-free state or by error compensation (exploiting redundancy) or by rollforward to an error-free state.
- Fault handling prevents located faults from being activated again.

Fault Removal

Definition (fault removal)

Fault removal aims at reducing the number and severity of faults.

- Fault removal during the development phase usually involves verification checks whether the system satisfies required properties.
- Fault removal during the operational phase is often driven by errors that have been detected and reported (corrective maintenance) or by faults that have been observed in similar systems or that were found in the specification but which have not led to errors yet (preventive maintenance).
- Sometimes it is impossible or too costly to remove a fault but it is possible to prevent the activation of the fault or to limit the possible impact of the fault, i.e, its severity.

Fault Forecasting

Definition (fault forecasting)

Fault forecasting aims at estimating the present number, the future incidences, and the likely consequences of faults.

- Qualitative evaluation identifies, classifies, and ranks the failure modes, or the event combinations that would lead to failures.
- Quantitative evaluation determines the probabilities to which some of the dependability attributes are satisfied.

Dependability Metrics

Reliability and MTTF/MTBF/MTTR

Definition (reliability)

The *reliability* $R(t)$ of a system S is defined as the probability that S is delivering correct service in the time interval $[0, t]$.

- A metric for the reliability $R(t)$ for non repairable systems is the Mean Time To Failure (MTTF), normally expressed in hours.
- A metric for the reliability $R(t)$ for repairable systems is the Mean Time Between Failures (MTBF), normally expressed in hours.
- The mean time it takes to repair a repairable system is called the Mean Time To Repair (MTTR), normally expressed in hours.
- These metrics are meaningful in the steady-state, i.e., when the system does not change or evolve.

Availability

Definition (availability)

The *availability* $A(t)$ of a system S is defined as the probability that S is delivering correct service at time t.

- A metric for the average, steady-state availability of a repairable system is $A = MTBF/(MTBF + MTTR)$, normally expressed in percent.
- A certain percentage-value may be more or less useful depending on the "failure distribution" (the "burstiness" of the failures).
- Critical computing systems often have to guarantee a certain availability. Availability requirements are usually defined in service level agreements.

Availability and the "number of nines"

Availability	Downtime per year	Downtime per month	Downtime per week	Downtime per day
90%	36.5 d	72 h	16.8 h	2.4 h
99%	3.65 d	7.20 h	1.68 h	14.4 min
99.9%	8.76 h	43.8 min	10.1 min	1.44 min
99.99%	52.56 min	4.38 min	1.01 min	8.64 s
99.999%	5.26 min	25.9 s	6.05 s	864.3 ms
99.9999%	31.5 s	2.59 s	604.8 ms	86.4 ms

- It is common practice to express the degrees of availability by the number of nines. For example, "5 nines availability" means 99.999% availability.

Safety

Definition (safety)

The *safety* $S(t)$ of a system S is defined as the probability that S is delivering correct service or has failed in a manner that does cause no harm in $[0, t]$.

- A metric for safety $S(t)$ is the Mean Time To Catastrophic Failure (MTTC), defined similarly to MTTF and normally expressed in hours.
- Safety is reliability with respect to malign failures.

Part: Software Engineering Aspects

General Aspects

Definitions of Software Engineering

Definition

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. (IEEE Standard Glossary of Software Engineering Terminology)

Definition

The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines. (Fritz Bauer)

Definition

An engineering discipline that is concerned with all aspects of software production. (Ian Sommerville)

Good Software Development Practices

- Coding Styles
- Documentation
- Version Control Systems
- Code Reviews and Pair Programming
- Automated Build and Testing Procedures
- Issue Tracking Systems

Choice of Programming Languages

- Programming languages serve different purposes and it is important to select a language that fits the given task
- Low-level languages can be very efficient but they tend to allow programmers to make more mistakes
- High-level languages and in particular functional languages can lead to very abstract but also very robust code
- Concurrency is important these days and the mechanisms available in different programming languages can largely impact the robustness of the code
- Programming languages must match the skills of the developer team; introducing a new languages requires to train developers
- Maintainability of code must be considered when programming languages are selected

Defensive Programming

- It is common that functions are only partially defined.
- Defensive programming requires that the preconditions for a function are checked when a function is called.
- For some complex functions, it might even be useful to check the postcondition, i.e., that the function did achieve the desired result.
- Many programming languages have mechanisms to insert assertions into the source code in order to check pre- and postconditions.

Software Testing

Unit and Regression Testing

- Unit testing
  - Testing of units (abstract data types, classes, ...) of source code.
  - Usually supported by special unit testing libraries and frameworks.
- Regression testing
  - Testing of an entire program to ensure that a modified version of a program still handles all input correctly that an older version of a program handled correctly.
- A software bug reported by a customer is primarily a weakness of the regression test suite.
- Modern agile software development techniques rely on unit testing and regression testing techniques.

Test Coverages

- The test coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs.
- Function coverage:
  - Has each function in the program been called?
- Statement coverage:
  - Has each statement in the program been executed?
- Branch coverage:
  - Has each branch of each control structure been executed?
- Predicate coverage:
  - Has each Boolean sub-expression evaluated both to true and false?

Mutation Testing

- Mutation testing evaluates the effectiveness of a test suite.
- The source code of a program is modified algorithmically by applying mutation operations in order to produce mutants.
- A mutant is “killed” by a test suite if tests fail for the mutant. Mutants that are not “killed” indicate that the test suite is incomplete.
- Mutation operators often mimic typical programming errors:
  - Statement deletion, duplication, reordering, ...
  - Replacement of arithmetic operators with others
  - Replacement of boolean operators with others
  - Replacement of comparison relations with others
  - Replacement of variables with others (of the same type)
- The mutation score is the number of mutants killed normalized by the number of mutants.

Fuzzying

- Fuzzying or fuzz testing feeds invalid, unexpected, or simply random data into computer programs.
  - Some fuzzers can generate input based on their awareness of the structure of input data.
  - Some fuzzers can adapt the input based on their awareness of the code structure and which code paths have already been covered.
- The “american fuzzy lop” (AFL) uses genetic algorithms to adjust generated inputs in order to quickly increase code coverage.
- AFL has detected a significant number of serious software bugs.

Fault Injection

- Fault injection techniques inject faults into a program by either
  - modifying source code (very similar to mutation testing) or
  - injecting faults at runtime (often via modified library calls).
- Fault injection can be highly effective to test whether software deals with rare failure situations, e.g., the injection of system calls failures that usually work.
- Fault injection can be used to evaluate the robustness of the communication between programs (deleting, injecting, reordering messages).
- Can be implemented using library call interception techniques.

Multiple Independent Computations

- Dionysius Lardner 1834:

  The most certain and effectual check upon errors which arise in the process of computation is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods.

- Charles Babbage, 1837:

  When the formula to be computed is very complicated, it may be algebraically arranged for computation in two or more totally distinct ways, and two or more sets of cards may be made. If the same constants are now employed with each set, we may then be quite sure of the accuracy of them all.

Software Specification

Formal Specification and Verification

Definition (formal specification)

A *formal specification* uses a formal (mathematical) notation to provide a precise definition of what a program should do.

Definition (formal verification)

A *formal verification* uses logical rules to mathematically prove that a program satisfies a formal specification.

- For many non-trivial problems, creating a formal, correct, and complete specification is a problem by itself.
- A bug in a formal specification leads to programs with verified bugs.

Floyd-Hoare Triple

Definition (hoare triple)

Given a state that satisfies precondition P, executing a program C (and assuming it terminates) results in a state that satisfies postcondition Q. This is also known as the "Hoare triple":

$$\{P\}\ C\ \{Q\}$$

- Invented by Charles Anthony ("Tony") Richard Hoare with original ideas from Robert Floyd (1969).
- Hoare triple can be used to specify what a program should do.
- Example:

$$\{X = 1\}\ X := X + 1\ \{X = 2\}$$

Partial Correctness and Total Correctness

Definition (partial correctness)

An algorithm starting in a state that satisfies a precondition P is *partially correct with respect to P and Q* if results produced by the algorithm satisfy the postcondition Q. Partial correctness does not require that always a result is produced, i.e., the algorithm may not always terminate.

Definition (total correctness)

An algorithm is *totally correct with respect to P and Q* if it is partially correct with respect to P and Q and it always terminates.

Hoare Notation Conventions

1. The symbols V, V_1, ..., V_n stand for arbitrary variables. Examples of particular variables are X, Y, R etc.
2. The symbols E, E_1, ..., E_n stand for arbitrary expressions (or terms). These are expressions like $X + 1$, $\sqrt{2}$ etc., which denote values (usually numbers).
3. The symbols S, S_1, ..., S_n stand for arbitrary statements. These are conditions like $X < Y$, $X^2 = 1$ etc., which are either true or false.
4. The symbols C, C_1 , ..., C_n stand for arbitrary commands of our programming language; these commands are described in the following slides.

- We will use lowercase letters such as x and y to denote auxiliary variables (e.g., to denote values stored in variables).

Hoare Assignments

- Syntax: $V := E$
- Semantics: The state is changed by assigning the value of the term E to the variable V . All variables are assumed to have global scope.
- Example: $X := X + 1$

Hoare Skip Command

- Syntax: *SKIP*
- Semantics: Do nothing. The state after execution the command *SKIP* is the same as the state before executing the command *SKIP*.
- Example: *SKIP*

Hoare Command Sequences

- Syntax: $C_1; \ldots ; C_n$
- Semantics: The commands $C_1, \ldots , C_n$ are executed in that order.
- Example: $R := X; X := Y; Y := R$

Hoare Conditionals

- Syntax: $\mathit{IF}\ S\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2\ \mathit{FI}$
- Semantics: If the statement S is true in the current state, then C_1 is executed. If S is false, then C_2 is executed.
- Example: $\mathit{IF}\ X < Y\ \mathit{THEN}\ M := Y\ \mathit{ELSE}\ M := X\ \mathit{FI}$

Hoare While Loop

- Syntax: $WHILE\ S\ DO\ C\ OD$
- Semantics: If the statement S is true in the current state, then C is executed and the WHILE-command is repeated. If S is false, then nothing is done. Thus C is repeatedly executed until the value of S becomes false. If S never becomes false, then the execution of the command never terminates.
- Example: $WHILE\ \neg(X = 0)\ DO\ X := X - 2\ OD$

Termination can be Tricky

```
function COLLATZ($X$)
    while $X > 1$ do
        if $(X \% 2) \neq 0$ then
            $X \leftarrow (3 \cdot X) + 1$
        else
            $X \leftarrow X/2$
        end if
    end while
    return $X$
end function
```

- Collatz conjecture: The program will eventually return the number 1, regardless of which positive integer is chosen initially.

Specification can be Tricky

- Specification for the maximum of two variables:

$$\{\mathrm{T}\}\ C\ \{Y = max(X, Y)\}$$

- C could be:

```
IF X > Y THEN Y := X ELSE SKIP FI
```

- But C could also be:

```
IF X > Y THEN X := Y ELSE SKIP FI
```

- And C could also be:

```
Y := X
```

- Use auxiliary variables x and y to associate Q with P:

$$\{X = x \land Y = y\}\ C\ \{Y = max(x, y)\}$$

Software Verification

Floyd-Hoare Logic

- Floyd-Hoare Logic is a set of inference rules that enable us to formally proof partial correctness of a program.
- If S is a statement, we write $\vdash S$ to mean that S has a proof.
- The axioms of Hoare logic will be specified with a notation of the following form:

$$\frac{\vdash S_1, \ldots, \vdash S_n}{\vdash S}$$

- The conclusion S may be deduced from $\vdash S_1, \ldots, \vdash S_n$, which are the hypotheses of the rule.
- The hypotheses can be theorems of Floyd-Hoare logic or they can be theorems of mathematics.

Precondition Strengthening

- If P implies P' and we have shown $\{P'\}\ C\ \{Q\}$, then $\{P\}\ C\ \{Q\}$ holds as well:

$$\frac{\vdash P \rightarrow P', \quad \vdash \{P'\}\ C\ \{Q\}}{\vdash \{P\}\ C\ \{Q\}}$$

- Example: Since $\vdash X = n \rightarrow X + 1 = n + 1$, we can strengthen

$$\vdash \{X + 1 = n + 1\}\ X := X + 1\ \{X = n + 1\}$$

  to

$$\vdash \{X = n\}\ X := X + 1\ \{X = n + 1\}.$$

Postcondition Weakening

- If Q' implies Q and we have shown $\{P\}\ C\ \{Q'\}$, then $\{P\}\ C\ \{Q\}$ holds as well:

$$\frac{\vdash \{P\}\ C\ \{Q'\}, \quad \vdash Q' \rightarrow Q}{\vdash \{P\}\ C\ \{Q\}}$$

- Example: Since $X = n + 1 \rightarrow X > n$, we can weaken

$$\vdash \{X = n\}\ X := X + 1\ \{X = n + 1\}$$

to

$$\vdash \{X = n\}\ X := X + 1\ \{X > n\}$$

Weakest Precondition

Definition (weakest precondition)

Given a program C and a postcondition Q, the *weakest precondition* $wp(C, Q)$ denotes the largest set of states for which C terminates and the resulting state satisfies Q.

Definition (weakest liberal precondition)

Given a program C and a postcondition Q, the *weakest liberal precondition* $wlp(C, Q)$ denotes the largest set of states for which C leads to a resulting state satisfying Q.

- The "weakest" precondition P means that any other valid precondition implies P.
- The definition of $wp(C, Q)$ is due to Dijkstra (1976) and it requires termination while $wlp(C, Q)$ does not require termination.

Strongest Postcondition

Definition (stronges postcondition)

Given a program C and a precondition P, the *strongest postcondition* $sp(C, P)$ has the property that $\vdash \{P\}\ C\ \{sp(C, P)\}$ and for any Q with $\vdash \{P\}\ C\ \{Q\}$, we have $\vdash sp(C, P) \rightarrow Q$.

- The "strongest" postcondition Q means that any other valid postcondition is implied by Q (via postcondition weakening).

Assignment Axiom

- Let $P[E/V]$ (P with E for V) denote the result of substituting the term E for all occurances of the variable V in the statement P.
- An assignment assigns a variable V an expression E:

$$\vdash \{P[E/V]\}\ V := E\ \{P\}$$

- Example:

$$\{X + 1 = n + 1\}\ X := X + 1\ \{X = n + 1\}$$

Specification Conjunction and Disjunction

- If we have shown $\{P_1\}\ C\ \{Q_1\}$ and $\{P_2\}\ C\ \{Q_2\}$, then $\{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}$ holds as well:

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\}, \quad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \wedge P_2\}\ C\ \{Q_1 \wedge Q_2\}}$$

- We get a similar rule for disjunctions:

$$\frac{\vdash \{P_1\}\ C\ \{Q_1\}, \quad \vdash \{P_2\}\ C\ \{Q_2\}}{\vdash \{P_1 \vee P_2\}\ C\ \{Q_1 \vee Q_2\}}$$

- These rules allows us to prove $\vdash \{P\}\ C\ \{Q_1 \wedge Q_2\}$ by proving both $\vdash \{P\}\ C\ \{Q_1\}$ and $\vdash \{P\}\ C\ \{Q_2\}$.

Skip Command Rule

- Syntax: *SKIP*
- Semantics: Do nothing. The state after execution the command *SKIP* is the same as the state before executing the command *SKIP*.
- Skip Command Rule:

$$\frac{}{\vdash \{P\}\ \mathit{SKIP}\ \{P\}}$$

Sequence Rule

- Syntax: $C_1; \ldots; C_n$
- Semantics: The commands $C_1, \ldots, C_n$ are executed in that order.
- Sequence Rule:

$$\frac{\vdash \{P\}\ C_1\ \{R\}, \quad \vdash \{R\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1; C_2\ \{Q\}}$$

The sequence rule can be easily generalized to $n > 2$ commands:

$$\frac{\vdash \{P\}\ C_1\ \{R_1\}, \vdash \{R_1\}\ C_2\ \{R_2\},\ \ldots,\ \vdash \{R_{n-1}\}\ C_n\ \{Q\}}{\vdash \{P\}\ C_1; C_2; \ldots; C_n\ \{Q\}}$$

Conditional Command Rule

- Syntax: *IF S THEN* C_1 *ELSE* C_2 *FI*
- Semantics: If the statement S is true in the current state, then C_1 is executed. If S is false, then C_2 is executed.
- Conditional Rule:

$$\frac{\vdash \{P \land S\}\ C_1\ \{Q\}, \quad \vdash \{P \land \neg S\}\ C_2\ \{Q\}}{\vdash \{P\}\ \mathit{IF}\ S\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2\ \mathit{FI}\ \{Q\}}$$

While Command Rule

- Syntax: *WHILE S DO C OD*
- Semantics: If the statement S is true in the current state, then C is executed and the WHILE-command is repeated. If S is false, then nothing is done. Thus C is repeatedly executed until the value of S becomes false. If S never becomes false, then the execution of the command never terminates.
- While Rule:

$$\frac{\vdash \{P \land S\}\ C\ \{P\}}{\vdash \{P\}\ \mathit{WHILE}\ S\ \mathit{DO}\ C\ \mathit{OD}\ \{P \land \neg S\}}$$

P is an invariant of C whenever S holds. Since executing C preserves the truth of P, executing C any numbner of times also preserves the truth of P.

Arrays

- Let the terms $A\{E_1 \leftarrow E2\}$ denote an array identical to A with the E_1-th component changed to the value E_2.
- With this, the assignment command can be extended to support arrays, i.e., the array assignment is a special case of an ordinary variable assignment.

$$\vdash \{P[A\{E_1 \leftarrow E_2\}/A]\}\ A[E_1] := E_2\ \{P\}$$

- The following axioms are needed to reason about arrays:

$$\vdash A\{E_1 \leftarrow E_2\}[E_1] = E_2$$

$$E_1 \neq E_2\ \rightarrow \vdash A\{E_1 \leftarrow E2\}[E_3] = A[E_3]$$

Proof Automation

- Proving even simple programs manually takes a lot of effort
- There is a high risk to make mistakes during the process
- General idea how to automate the proof:
  (i) Let the human expert provide annotations of the specification (e.g., loop invariants) that help with the generation of proof obligations
  (ii) Generate proof obligations automatically (verification conditions)
  (iii) Use automated theorem provers to verify some of the proof obligations
  (iv) Let the human expert prove the remaining proof obligations (or let the human expert provide additional annotations that help the automated theorem prover)
- Step (ii) essentially compiles an annotated program into a conventional mathematical problem.

Annotations

- Annotations are required
  - (i) before each command C_i (with $i > 1$) in a sequence $C_1; C_2; \ldots; C_n$, where C_i is not an assignment command and
  - (ii) after the keyword *DO* in a *WHILE* command (loop invariant)
- The inserted annotation is expected to be true whenever the execution reaches the point of the annotation.
- For a properly annotated program, it is possible to generate a set of proof goals (verification conditions).
- It can be shown that once all generated verification conditions have been proved, then $\vdash \{P\}\ C\ \{Q\}$.

Generation of Verification Conditions

- Assignment $\{P\}\ V := E\ \{Q\}$:
  Add verification condition $P \rightarrow Q[E/V]$.
- Conditions $\{P\}\ \mathit{IF}\ S\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2\ \mathit{FI}\ \{Q\}$
  Add verification conditions generated by $\{P \land S\}\ C_1\ \{Q\}$ and $\{P \land \neg S\}\ C_2\ \{Q\}$
- Sequences of the form $\{P\}\ C_1; \ldots; C_{n-1};\ \{R\}\ C_n\ \{Q\}$
  Add verification conditions generated by $\{P\}\ C_1; \ldots; C_{n-1}\ \{R\}$ and $\{R\}\ C_n\ \{Q\}$
- Sequences of the form $\{P\}\ C_1; \ldots; C_{n-1};\ V := E\ \{Q\}$
  Add verification conditions generated by $\{P\}\ C_1; \ldots; C_{n-1}\ \{Q[E/V]\}$
- While loops $\{P\}\ \mathit{WHILE}\ S\ \mathit{DO}\ \{R\}\ C\ \mathit{OD}\ \{Q\}$
  Add verification conditions $P \rightarrow R$ and $R \land \neg S \rightarrow Q$
  Add verificiation conditions generated by $\{R \land S\}\ C\ \{R\}$

Total Correctness

- We assume that the evaluation of expressions always terminates.
- With this simplifying assumption, only *WHILE* commands can cause loops that potentially do not terminate.
- All rules for the other commands can simply be extended to cover total correctness.
- The assumption that expression evaluation always terminates is often not true. (Consider recursive functions that can go into an endless recursion.)
- We have so far also silently assumed that the evaluation of expressions always yields a proper value, which is not the case for a division by zero.
- Relaxing our assumptions for expressions is possible but complicates matters significantly.

Rules for Total Correctness [1/4]

- Assignment axiom

$$\vdash [P[E/V]]\ V := E\ [P]$$

- Precondition strengthening

$$\frac{\vdash P \rightarrow P', \quad \vdash [P']\ C\ [Q]}{\vdash [P]\ C\ [Q]}$$

- Postcondition weakening

$$\frac{\vdash [P]\ C\ [Q'], \quad \vdash Q' \rightarrow Q}{\vdash [P]\ C\ [Q]}$$

Rules for Total Correctness [2/4]

- Specification conjunction

$$\frac{\vdash [P_1]\ C\ [Q_1], \quad \vdash [P_2]\ C\ [Q_2]}{\vdash [P_1 \land P_2]\ C\ [Q_1 \land Q_2]}$$

- Specification disjunction

$$\frac{\vdash [P_1]\ C\ [Q_1], \quad \vdash [P_2]\ C\ [Q_2]}{\vdash [P_1 \lor P_2]\ C\ [Q_1 \lor Q_2]}$$

- Skip command rule

$$\overline{[P]\ SKIP\ [P]}$$

Rules for Total Correctness [3/4]

- Sequence rule

$$\frac{\vdash [P]\ C_1\ [R_1],\ \vdash [R_1]\ C_2\ [R_2],\ \ldots,\ \vdash [R_{n-1}]\ C_n\ [Q]}{\vdash [P]\ C_1; C_2; \ldots; C_n\ [Q]}$$

- Conditional rule

$$\frac{\vdash [P \land S]\ C_1\ [Q], \quad \vdash [P \land \neg S]\ C_2\ [Q]}{\vdash [P]\ \mathit{IF}\ S\ \mathit{THEN}\ C_1\ \mathit{ELSE}\ C_2\ \mathit{FI}\ [Q]}$$

Rules for Total Correctness [4/4]

- While rule

$$\frac{\vdash [P \land S \land E = n]\ C\ [P \land (E < n)], \quad \vdash P \land S \rightarrow E \geq 0}{\vdash [P]\ \mathit{WHILE}\ S\ \mathit{DO}\ C\ \mathit{OD}\ [P \land \neg S]}$$

  E is an integer-valued expression
  n is an auxiliary variable not occuring in P, C, S, or E

- A prove has to show that a non-negative integer, called a *variant*, decreases on each iteration of the loop command C.

Generation of Termination Verification Conditions

- The rules for the generation of termination verificiation conditions follow directly from the rules for the generation of partial correctness verificiation conditions, except for the while command.
- To handle the while command, we need an additional annotation (in square brackets) that provides the variant expression.
- For while loops of the form $\{P\}$ *WHILE S DO* $\{R\}$ $[E]$ *C OD* $\{Q\}$ add the verification conditions

$$P \rightarrow R$$
$$R \wedge \neg S \rightarrow Q$$
$$R \wedge S \rightarrow E \geq 0$$

and add verificiation conditions generated by $\{R \wedge S \wedge (E = n)\}\ C\ \{R \wedge (E < n)\}$

Termination and Correctness

- Partial correctness and termination implies total correctness:

$$\frac{\vdash \{P\}\ C\ \{Q\}, \quad \vdash [P]\ C\ [\mathrm{T}]}{\vdash [P]\ C\ [Q]}$$

- Total correctness implies partial correctness and termination:

$$\frac{\vdash [P]\ C\ [Q]}{\vdash \{P\}\ C\ \{Q\}, \quad \vdash [P]\ C\ [\mathrm{T}]}$$

Part: Software Vulnerabilities and Exploits

Terminology

Malware

Definition (malware)

Malware (short for malicious software) is software intentionally designed to cause damage to a computer system or a computer network.

- A *virus* depends on a “host” and when activated replicates itself by modifying other computer programs.
- A *worm* is self-contained malware replicating itself in order to spread to other computers.
- A *trojan horse* is malware misleading users of its true intent.
- *Ransomware* blocks access to computers or data until a ransom has been paid.
- *Spyware* gathers information about a person or organization, without their knowledge.

Social Engineering

Definition (social engineering)

Social engineering is the psychological manipulation of people into performing actions or divulging confidential information.

Examples:

- *A*n attacker sends a document that appears to be legitimate in order to attract the victim to a fraudulent web page requesting access codes (phishing).
- An attacker pretends to be another person with the goal of gaining access physically to a system or building (impersonation).
- An attacker drops devices that contain malware and look like USB sticks in spaces visited by a victim (USB drop).

Backdoors

Definition (backdoor)

A *backdoor* is a method of bypassing normal authentication systems in order to gain access to a computer program and computing system. Backdoors might be created by malicious software developers, tools such as compilers, or by other forms of malware.

Examples:

- Well-known default passwords effectively function as backdoors.
- Debugging features used during development phases can act as backdoors.
- Backdoors may be inserted by a malicious compiler.

Rootkits

Definition (rootkit)

A *rootkit* . . .

Advanced Persistent Threats

Control Flow Exploits

Stacks (Intel x86)

Shellcodes (Intel x86)

Stack Smashing (Intel x86)

Return Oriented Programming (Intel x86)

Format String Attacks (Intel x86)

Attacks and Stealthiness

Part: Cryptography

Cryptography Primer

Try to read the following text. . .

```
Jrypbzr gb Frpher naq Qrcraqnoyr Flfgrzf!

W!eslmceotmsey St oe lSbeacdunreep eaDn d

J!rfyzprbgzfrl Fg br yFornpqhaerrc rnQa q
```

Terminology (Cryptography)

- *Cryptology* subsumes cryptography and cryptanalysis:
  - *Cryptography* is the art of secret writing.
  - *Cryptanalysis* is the art of breaking ciphers.
- *Encryption* is the process of converting *plaintext* into an unreadable form, termed *ciphertext*.
- *Decryption* is the reverse process, recovering the plaintext back from the ciphertext.
- A *cipher* is an algorithm for encryption and decryption.
- A *key* is some secret piece of information used as a parameter of a cipher and customizes the algorithm used to produce ciphertext.

Cryptosystem

Definition (cryptosystem)

A *cryptosystem* is a quintuple (M, C, K, E_k, D_k), where

- M is a cleartext space,
- C is a chiffretext space,
- K is a key space,
- $E_k : M \to C$ is an encryption transformation with $k \in K$, and
- $D_k : C \to M$ is a decryption transformation with $k \in K$.

For a given k and all $m \in M$, the following holds:

$$D_k(E_k(m)) = m$$

Cryptosystem Requirements

- The transformations E_k and D_k must be efficient to compute.
- It must be easy to find a key $k \in K$ and the functions E_k and D_k.
- The security of the system rests on the secrecy of the key and not on the secrecy of the transformations E_k and D_k (the algorithms).
- For a given $c \in C$, it is difficult to systematically compute
  - D_k even if $m \in M$ with $E_k(m) = c$ is known
  - a cleartext $m \in M$ such that $E_k(m) = c$.
- For a given $c \in C$, it is difficult to systematically determine
  - E_k even if $m \in M$ with $E_k(m) = c$ is known
  - $c' \in C$ with $c' \neq c$ such that $D_k(c')$ is a valid cleartext in M.

Symmetric vs. Asymmetric Cryptosystems

Symmetric Cryptosystems

- Both (all) parties share the same key and the key needs to be kept secret.
- Examples: AES, DES (outdated), Twofish, Serpent, IDEA, . . .

Asymmetric Cryptosystems

- Each party has a pair of keys: one key is public and used for encryption while the other key is private and used for decryption.
- Examples: RSA, DSA, ElGamal, ECC, . . .

- For asymmetric cryptosystems, a key is a key pair (k, k^{-1}) where k denotes the public key and k^{-1} the associated private key.

Cryptographic Hash Functions

Definition (cryptographic hash function)

A *cryptographic hash function* H is a hash function that meets the following requirements:

1. The hash function H is efficient to compute for arbitrary input m.
2. Given a hash value h, it should be difficult to find an input m such that $h = H(m)$ (preimage resistance).
3. Given an input m, it should be difficult to find another input $m' \neq m$ such that $H(m) = H(m')$ (2nd-preimage resistance).
4. It should be difficult to find two different inputs m and m' such that $H(m) = H(m')$ (collision resistance).

Digital Signatures

- Digital signatures are used to prove the authenticity of a message (or document) and its integrity.
  - The receiver can verify the claimed identity of the sender (authentication).
  - The sender can not deny that it did sent the message (non-repudiation).
  - The receiver can verify that the messages was not tampered with (integrity).
- Digitally signing a message (or document) means that
  - the sender puts a signature into a message (or document) that can be verified and
  - that we can be sure that the signature cannot be faked (e.g., copied from some other message)
- Digital signatures are often implemented by signing a cryptographic hash of the original message (or document) since this is usually computationally less expensive

Usage of Cryptography

- Encrypting data in communication protocols (prevent eavesdropping)
- Encrypting data elements of files (e.g., passwords stored in a database)
- Encrypting entire files (prevent data leakage if machines are stolen or attacked)
- Encrypting entire file systems (prevent data leakage if machines are stolen or attacked)
- Encrypting backups stored on 3rd party storage systems
- Encrypting digital media to obtain revenue by selling keys (for example pay TV)
- Digital signatures of files to ensure that changes of file content can be detected or that the content of a file can be proven to originate from a certain source
- Encrypted token needed to obtain certain services or to authorize transactions
- Modern electronic currencies (cryptocurrency)

Symmetric Encryption Algorithms and Block Ciphers

Substitution Ciphers

Definition (monoalphabetic and polyalphabetic substitution ciphers)

A *monoalphabetic substitution cipher* is a bijection on the set of symbols of an alphabet. A *polyalphabetic substitution cipher* is a substitution cipher with multiple bijections, i.e., a collection of monoalphabetic substitution ciphers.

- There are $|M|!$ different bijections of a finite alphabet M.
- Monoalphabetic substitution ciphers are easy to attack via frequency analysis since the bijection does not change the frequency of cleartext characters in the ciphertext.
- Polyalphabetic substitution ciphers are still relatively easy to attack if the length of the message is significantly longer than the key.

Permutation Cipher

Definition (permutation cipher)

A *permutation cipher* maps a plaintext $m_0, \ldots, m_{l-1}$ to $m_{\tau(0)}, \ldots, m_{\tau(l-1)}$ where τ is a bijection of the positions $0, \ldots, l-1$ in the message.

- Permutation ciphers are also called transposition ciphers.
- To make the cipher parametric in a key, we can use a function τ_k that maps a key k to bijections.

Product Cipher

Definition (product cipher)

A *product cipher* combines two or more ciphers in a manner that the resulting cipher is more secure than the individual components to make it resistant to cryptanalysis.

- Combining multiple substitution ciphers results in another substitution cipher and hence is of little value.
- Combining multiple permutation ciphers results in another permutation cipher and hence is of little value.
- Combining substitution ciphers with permutation ciphers gives us ciphers that are much harder to break.

Chosen-Plaintext and Chosen-Ciphertext Attack

Definition (chosen plaintext attack)

In a *chosen-plaintext attack* the adversary can chose arbitrary cleartext messages m and feed them into the encryption function E to obtain the corresponding ciphertext.

Definition (chosen ciphertext attack)

In a *chosen-ciphertext attack* the adversary can chose arbitrary ciphertext messages c and feed them into the decryption function D to obtain the corresponding cleartext.

Polynomial and Negligible Functions

Definition (polynomial and negligible functions)

A function $f : \mathbb{N} \to \mathbb{R}^+$ is called

- *polynomial* if $f \in O(p)$ for some polynomial p
- *super-polynomial* if $f \notin O(p)$ for every polynomial p
- *negligible* if $f \in O(1/|p|)$ for every polynomial $p : \mathbb{N} \to \mathbb{R}^+$

Polynomial Time and Probabilistic Algorithms

Definition (polynomial time)

An algorithm A is called *polynomial time* if the worst-case time complexity of A for input of size n is a polynomial function.

Definition (probabilistic algorithm)

A *probabilistic algorithm* is an algorithm that may return different results when called multiple times for the same input.

Definition (probabilistic polynomial time)

A *probabilistic polynomial time* (PPT) algorithm is a probabilistic algorithm with polynomial time.

One-way Functions

Definition (one-way function)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is a *one-way function* if and only if f can be computed by a polynomial time algorithm, but any polynomial time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with negligible probability.

- The existence of such one-way functions is still an open conjecture.
- Their existence would prove that the complexity classes P and NP are not equal.

Security of Ciphers

- What does it mean for an encryption scheme to be secure?
- Consider an adversary who can pick two plaintexts m_0 and m_1 and who randomly receives either $\mathrm{E}(m_0)$ or $\mathrm{E}(m_1)$.
- An encryption scheme can be considered secure if the adversary cannot distinguish between the two situations with a probability that is non-negligibly better than $\frac{1}{2}$.

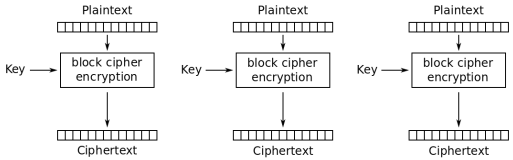
Block Cipher

Definition (block cipher)

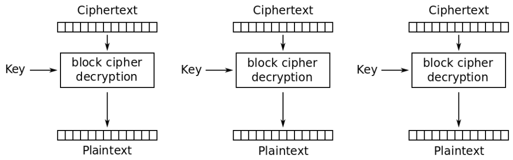
A *block cipher* is a cipher that operates on fixed-length groups of bits called a block.

- A given variable-length plaintext is split into blocks of fixed size and then each block is encrypted individually.
- The last block may need to be padded using zeros or random bits.
- Encrypting each block individually has certain shortcomings:
  - the same plaintext block yields the same ciphertext block
  - encrypted blocks can be rearranged and the receiver may not necessarily detect this
- Hence, block ciphers are usually used in more advanced modes in order to produce better results that reveal less information about the cleartext.

Electronic Codebook Mode

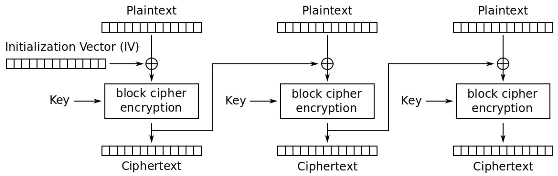


Electronic Codebook (ECB) mode encryption

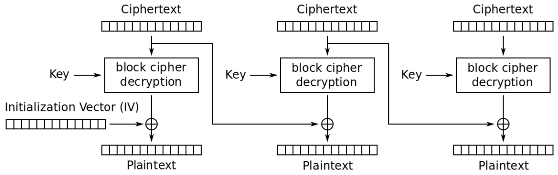


Electronic Codebook (ECB) mode decryption

Cipher Block Chaining Mode

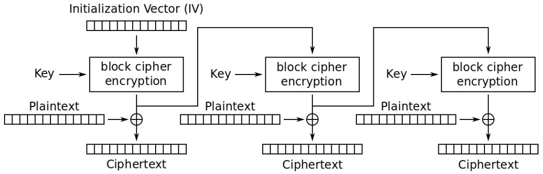


Cipher Block Chaining (CBC) mode encryption

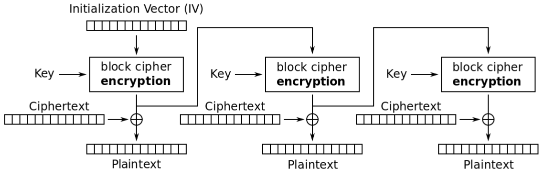


Cipher Block Chaining (CBC) mode decryption

Output Feedback Mode

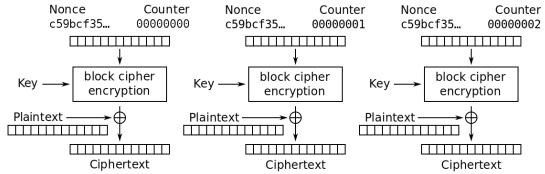


Output Feedback (OFB) mode encryption

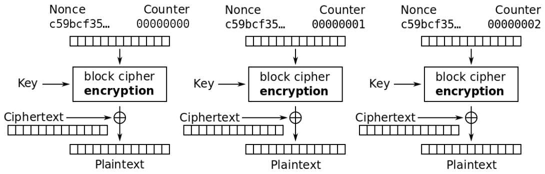

Initialization Vector (IV)
Key
block cipher **encryption**
Key
block cipher **encryption**
Key
block cipher **encryption**
Ciphertext
Ciphertext
Ciphertext
Plaintext
Plaintext
Plaintext

Output Feedback (OFB) mode decryption

Counter Mode



Counter (CTR) mode encryption



Counter (CTR) mode decryption

Substitution-Permutation Networks

Definition (substitution-permutation network)

A *substitution-permutation network* is a block cipher whose bijections arise as products of substitution and permutation ciphers.

- To process a block of N bits, the block is typically devided into b chunks of $n = N/b$ bits each.
- Each block is processed by a sequence of rounds:
  - Key step: A key step maps a block by xor-ing it with a round key.
  - Substitution step: A chunk of n bits is substituted by a substitution box (S-box).
  - Permutation step: A permutation box (P-box) permutes the bits received from S-boxes to produce bits for the next round.

Advanced Encryption Standard (AES)

- Designed by two at that time relatively unknown cryptographers from Belgium (Vincent Rijmen and Joan Daemen, hence the name Rijndael of the proposal).
- Choosen by NIST (National Institute of Standards and Technology of the USA) after an open call for encryption algorithms.
- Characteristics:
  - AES has a blocksize of 128 bits.
  - AES with 128 bit keys uses 10 rounds.
  - AES with 192 bit keys uses 12 rounds.
  - AES with 256 bit keys uses 14 rounds.

Advanced Encryption Standard (AES) Rounds

- Round 0:
  - (a) key step with k_0
- Round i: (i = 1, ..., r-1)
  - (a) substitution step (called sub-bytes) with fixed 8-bit S-box (used 16 times)
  - (b) permutation step (called shift-row) with a fixed permutation of 128 bits
  - (c) substitution step (called mix-columns) with a fixed 32-bit S-box (used 4 times)
  - (d) key step (called add-round-key) with a key k_i
- Round r: (no mix-columns)
  - (a) substitution step (called sub-bytes) with fixed 8-bit S-box (used 16 times)
  - (b) permutation step (called shift-row) with a fixed permutation of 128 bits
  - (c) key step (called add-round-key) with a key k_r

Asymmetric Encryption Algorithms

Asymmetric Encryption Algorithms

- Asymmetric encryption schemes work with a key pair:
  - a public key used for encryption
  - a private key used for decryption
- Everybody can send a protected message to a receiver by using the receiver's public key to encrypt the message. Only the receiver knowing the matching private key will be able to decrypt the message.
- Asymmetric encryption schemes give us a very easy way to digitally sign a message: A message encrypted by a sender with the sender's private key can be verified by any receiver using the sender's public key.
- Ron Rivest, Adi Shamir and Leonard Adleman (all then at MIT) published the RSA cryptosystem in 1978, which relies on the factorization problem of large numbers.
- Newer asynchronous cryptosystems often rely on the problem of finding discrete logarithms.

Rivest-Shamir-Adleman (RSA)

- Key generation:
  1. Generate two large prime numbers p and q of roughly the same length.
  2. Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$.
  3. Choose a number e satisfying $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$.
  4. Compute d satisfying $1 < d < \varphi(n)$ and $ed \bmod \varphi(n) = 1$.
  5. The public key is (n, e), the private key is (n, d); p, q and $\varphi(n)$ are discarded.
- Encryption:
  1. The cleartext m is represented as a sequence of numbers m_i with $m_i \in \{0, 1, \ldots, n-1\}$ and $m_i \neq p$ and $m_i \neq q$.
  2. Using the public key (n, e) compute $c_i = m_i^e \bmod n$ for all m_i.
- Decryption:
  1. Using the private key (n, d) compute $m_i = c_i^d \bmod n$ for all c_i.
  2. Transform the number sequence m_i back into the original cleartext m.

RSA Math Background

Definition (coprime)

Two integers a and b are *coprime* if the only positive integer that divides both is 1.

Definition (Euler function)

The function $\varphi(n) = |\{a \in \mathbb{N} | 1 \leq a \leq n \wedge gcd(a, n) = 1\}|$ is called the Euler function.

Theorem (Euler's theorem)

If a and n are coprime, then $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Theorem

Let m and n be coprime integers. Then $\varphi(n \cdot m) = \varphi(n) \cdot \varphi(m)$.
If p is a prime number, then $\varphi(p) = p - 1$.

RSA Properties

- Security relies on the problem of factoring very large numbers.
- Quantum computers may solve this problem in polynomial time — so RSA will become obsolete once someone manages to build quantum computers.
- The prime numbers p and q should be at least 1024 (better 2048) bit long and not be too close to each other (otherwise an attacker can search in the proximity of $\sqrt{n}$).
- Since two identical cleartexts m_i and m_j would lead to two identical ciphertexts c_i and c_j, it is advisable to pad the cleartext numbers with some random digits.
- Large prime numbers can be found using probabilistic prime number tests.
- RSA encryption and decryption is compute intensive and hence usually used only on small cleartexts.

Elliptic Curve Cryptography (ECC)

Definition (elliptic curve)

An *elliptic curve* is a plane curve over a finite field which consists of the points

$$E = \{(x, y) | y^2 = x^3 + ax + b\} \cup \{\infty\}$$

with the parameters a and b along with a distinguished point at infinity, denoted ∞.

- It is possible to define $R = P + Q$ with R, P, Q on an elliptic curve E.
- With the addition defined, it is possible to define scalar multiplication $k \cdot P$
- Given P and k, it is efficient to calculate $Q = k \cdot P$
- Given Q and P, it is difficult to find k such that $Q = k \cdot P$

Cryptographic Hash Functions

Cryptographic Hash Functions

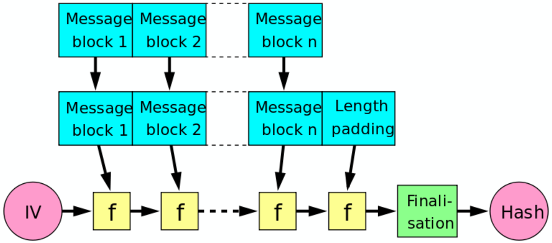
- Cryptographic hash functions serve many purposes:
    - data integrity verification
    - integrity verification and authentication (via keyed hashes)
    - calculation of fingerprints for efficient digital signatures
    - adjustable proof of work mechanisms
- A cryptographic hash function can be obtained from a symmetric block encryption algorithm in cipher-block-chaining mode by using the last ciphertext block as the hash value.
- It is possible to construct more efficient cryptographic hash functions.

Cryptographic Hash Functions

Name	Published	Digest size	Block size	Rounds
MD-5	1992	128 b	512 b	4
SHA-1	1995	160 b	512 b	80
SHA-256	2002	256 b	512 b	64
SHA-512	2002	512 b	1024 b	80
SHA3-256	2015	256 b	1088 b	24
SHA3-512	2015	512 b	576 b	24

- MD-5 has been widely used but is largely considered insecure since the late 1990s.
- SHA-1 is largely considered insecure since the early 2000s.

Merkle-Damgård Construction



- The message is padded and postfixed with a length value.
- The function f is a collision-resistant compression function, which compresses a digest-sized input from the previous step (or the initialization vector) and a block-sized input from the message into a digest-sized value.

Hashed Message Authentication Codes

- A keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key.
- An HMAC can be used to verify both data integrity and authenticity.
- An HMAC does not encrypt the message.
- The message must be sent alongside the HMAC hash. Parties with the secret key will hash the message again themselves, and if it is authentic, the received and computed hashes will match.

HMAC Computation

Given a key k, a hash function H, and a message m, the HMAC using H ($HMAC_H$) is calculated as follows:

$$HMAC_H(k, m) = H((k' \oplus opad) \parallel H((k' \oplus ipad) \parallel m))$$

- The key k' is derived from the original key k by padding k to the right with extra zeroes to the input block size of the hash function, or by hashing k if it is longer than that block size.
- The *opad* is the outer padding (0x5c5c5c. . . 5c, one-block-long hexadecimal constant). The *ipad* is the inner padding (0x363636. . . 36, one-block-long hexadecimal constant).
- The symbol $\oplus$ denotes bitwise exclusive or and the symbol $\parallel$ denotes concatenation.

Authenticated Encryption with Associated Data

- It is often necessary to combine encryption with authentication of the data.
- Encryption protects the data and a message authentication code (MAC) protects the data against attempts to insert, remove, or modify data.
- Let E_k be an encryption function with key k and H_k a hash-based MAC with key k and $\|$ denotes concatenation.
- Encrypt-then-Mac (EtM)
$$E_k(M) \| H_k(E_k(M))$$
- Encrypt-and-Mac (EaM)
$$E_k(M) \| H_k(M)$$
- Mac-then-Encrypt (MtE)
$$E_k(M \| H_k(M))$$

Digital Signatures and Certificates

Digital Signatures

- Digital signatures are used to prove the authenticity of a message (or document) and its integrity.
  - Receiver can verify the claimed identity of the sender (authentiation)
  - The sender can later not deny that he/she sent the message (non-repudiation)
  - The message cannot be modified with invalidating the signature (integrity)
- A digital signature means that
  - the sender puts a signature into a message (or document) that can be verified and
  - that we can be sure that the signature cannot be faked (e.g., copied from some other message)
- Do not confuse digital signatures, which use cryptographic mechanisms, with electronic signatures, which may just use a scanned signature or a name entered into a form.

Digital Signatures using Asymmetric Cryptosystems

- Direct signature of a document m:
    - Signer: $S = E_{k^{-1}}(m)$
    - Verifier: $D_k(S) \stackrel{?}{=} m$
- Indirect signature of a hash of a document m:
    - Signer: $S = E_{k^{-1}}(H(m))$
    - Verifier: $D_k(S) \stackrel{?}{=} H(m)$
- The verifier needs to be able to obtain the public key k of the signer from a trustworthy source.
- The signature of a hash is faster (and hence more common) but it requires to send the signature S along with the document m.

Public Key Certificates

Definition (public key certificate)

A *public key certificate* is an electronic document used to prove the ownership of a public key. The certificate includes

- information about the public key,
- information about the identity of its owner (called the subject),
- information about the lifetime of the certificate, and
- the digital signature of an entity that has verified the certificate's contents (called the issuer of the certificate).

- If the signature is valid, and the software examining the certificate trusts the issuer of the certificate, then it can trust the public key contained in the certificate to belong to the subject of the certificate.

Public Key Infrastructure (PKI)

Definition

A *public key infrastructure* (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

- A central element of a PKI is the certificate authority (CA), which is responsible for storing, issuing and signing digital certificates.
- CAs are often hierarchically organized. A root CA may delegate some of the work to trusted secondary CAs if they execute their tasks according to certain rules defined by the root CA.
- A key function of a CA is to verify the identity of the subject (the owner) of a public key certificate.

X.509 Certificate ASN.1 Definition

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
     }
```

X.509 Certificate ASN.1 Definition

```
Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

CertificateSerialNumber  ::=  INTEGER

Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time }

Time ::= CHOICE {
     utcTime        UTCTime,
     generalTime    GeneralizedTime }

UniqueIdentifier  ::=  BIT STRING

SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING
                 -- contains the DER encoding of an ASN.1 value
                 -- corresponding to the extension type identified
                 -- by extnID
     }
```

X.509 Subject Alternative Name Extension

```
id-ce-subjectAltName OBJECT IDENTIFIER ::=  { id-ce 17 }

SubjectAltName ::= GeneralNames

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
     otherName                       [0]     OtherName,
     rfc822Name                      [1]     IA5String,
     dNSName                         [2]     IA5String,
     x400Address                     [3]     ORAddress,
     directoryName                   [4]     Name,
     ediPartyName                    [5]     EDIPartyName,
     uniformResourceIdentifier       [6]     IA5String,
     iPAddress                       [7]     OCTET STRING,
     registeredID                    [8]     OBJECT IDENTIFIER }

OtherName ::= SEQUENCE {
     type-id    OBJECT IDENTIFIER,
     value      [0] EXPLICIT ANY DEFINED BY type-id }

EDIPartyName ::= SEQUENCE {
     nameAssigner            [0]     DirectoryString OPTIONAL,
     partyName               [1]     DirectoryString }
```

Automatic Certificate Management Environment (ACME)

- The ACME protocol provides so called Domain Validation certificates.
- It is a challenge-response protocol that aims to verify whether the client has effective control over a domain name.
- The CA might challenge a client requesting a certificate for `example.com`
    - to provision a DNS record under `example.com` or
    - to provide an HTTP resource under `http://example.com`.
- ACME runs over HTTPS and message bodies are signed JSON objects.
- The client periodically contacts the server to obtain updated certificates or Online Certificate Status Protocol (OCSP) responses.

Key Exchange Schemes

Cryptographic Protocol Notation

$A, B, \ldots$	principals
$K_{AB}, \ldots$	symmetric key shared between A and B
$K_A, \ldots$	public key of A
$K_A^{-1}, \ldots$	private key of A
H	cryptographic hash function
$N_A, N_B, \ldots$	nonces (fresh random messages) chosen by A, B, ...
P, Q, R	variables ranging over principals
X, Y	variables ranging over statements
K	variable over a key
$\{m\}_K$	message m encrypted with key K

Key Exchange and Ephemeral Keys

Definition (key exchange)

A method by which cryptographic keys are established between two parties is called a *key exchange* or *key establishment* method.

Definition (ephemeral key)

A cryptographic key that is established for the use in a single session and discarded afterwards is called an *ephemeral key*.

Definition (forward secrecy)

A key exchange protocol has *forward secrecy* (also called perfect forward secrecy) if the ephemeral key will not be compromised even any long-term keys used during the key exchange are compromised.

Diffie-Hellman Key Exchange

- Initialization:
  - Define a prime number p and a primitive root g of $\mathbb{Z}_p$ with $g < p$. The numbers p and g can be made public.
- Exchange:
  - A randomly picks $x_A \in \mathbb{Z}_p$ and computes $y_A = g^{x_A} \bmod p$. x_A is kept secret while y_A is sent to B.
  - B randomly picks $x_B \in \mathbb{Z}_p$ and computes $y_B = g^{x_B} \bmod p$. x_B is kept secret while y_B is sent to A.
  - A computes:

$$K_{AB} = y_B^{x_A} \bmod p = (g^{x_B} \bmod p)^{x_A} \bmod p = g^{x_A x_B} \bmod p$$

  - B computes:

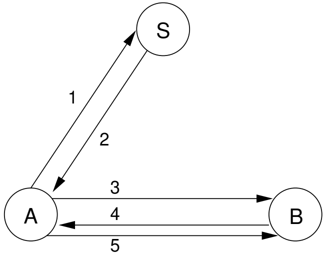
$$K_{AB} = y_A^{x_B} \bmod p = (g^{x_A} \bmod p)^{x_B} \bmod p = g^{x_A x_B} \bmod p$$

  - A and B now own a shared key K_{AB}.

Diffie-Hellman Key Exchange (cont.)

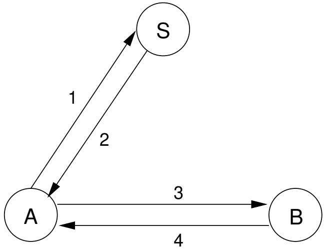
- A number g is a primitive root of $\mathbb{Z}_p = \{0, \ldots, p-1\}$ if the sequence $g^1 \bmod p, g^2 \bmod p, \ldots, g^{p-1} \bmod p$ produces the numbers $0, \ldots, p-2$ in any permutation.
- p should be choosen such that $(p-1)/2$ is prime as well.
- p should have a length of at least 2048 bits.
- Diffie-Hellman is not perfect: An attacker can play "man in the middle" (MIM) by claiming B's identity to A and A's identity to B.

Needham-Schroeder Protocol



$$\begin{array}{lll} \text{Msg 1:} & A \rightarrow S: & A, B, N_a \\ \text{Msg 2:} & S \rightarrow A: & \{N_a, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\ \text{Msg 3:} & A \rightarrow B: & \{K_{AB}, A\}_{K_{BS}} \\ \text{Msg 4:} & B \rightarrow A: & \{N_b\}_{K_{AB}} \\ \text{Msg 5:} & A \rightarrow B: & \{N_b - 1\}_{K_{AB}} \end{array}$$

Kerberos Protocol



$$\begin{array}{lll} \text{Msg 1:} & A \rightarrow S: & A, B \\ \text{Msg 2:} & S \rightarrow A: & \{T_s, L, K_{AB}, B, \{T_s, L, K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\ \text{Msg 3:} & A \rightarrow B: & \{T_s, L, K_{AB}, A\}_{K_{BS}}, \{A, T_a\}_{K_{AB}} \\ \text{Msg 4:} & B \rightarrow A: & \{T_a + 1\}_{K_{AB}} \end{array}$$

BAN Logic

- Idea: Use a formal logic to reason about authentication protocols.
- Answer questions such as:
  - What can be achieve with the protocol?
  - Does a given protocol have stronger prerequisites than some other protocol?
  - Does a protocol do something which is not needed?
  - Is a protocol minimal regarding the number of messages exchanged?
- The Burrows-Abadi-Needham (BAN) logic was a first attempt to provide a formalism for authentication protocol analysis.
- The spi calculus, an extension of the pi calculus, was introduced later to analyze cryptographic protocols.

Using BAN Logic

- Steps to use BAN logic:
  1. Idealize the protocol in the language of the formal BAN logic.
  2. Define your initial security assumptions in the language of BAN logic.
  3. Use the productions and rules of the logic to deduce new predicates.
  4. Interpret the statements you've proved by this process. Have you reached your goals?
  5. Remove unnecessary elements from the protocol, and repeat (optional).
- BAN logic does not prove correctness of the protocol; but it helps to find subtle errors.

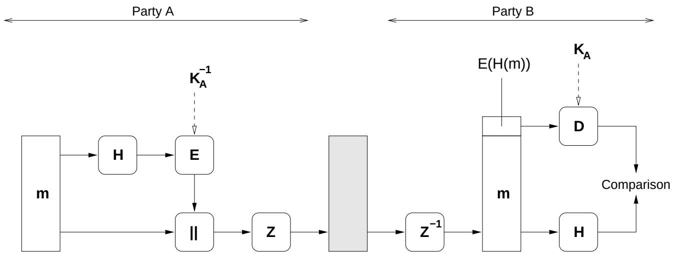
Part: Secure Communication Protocols

Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP)

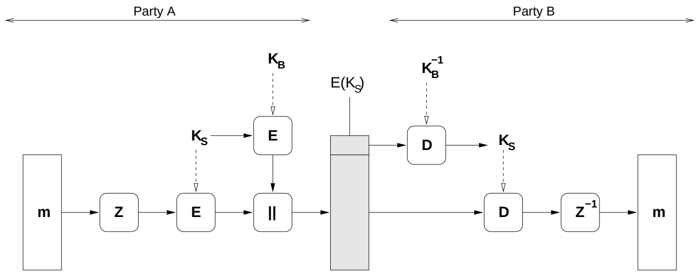
- PGP was developed by Philip Zimmerman in 1991
- PGP got famous because it demonstrated why patent laws and export laws in a globalized connected world need new interpretations.
- In order to export his PGP implementation, Philip Zimmerman did publish the code as a book.
- There are nowadays several independent PGP implementations.
- The underlying PGP specification is now called OpenPGP (RFC 4880).
- An alternative to PGP is S/MIME, which relies centralized trust via X.509 certificates, while PGP relies on a decentralized web of trust.

PGP Signatures



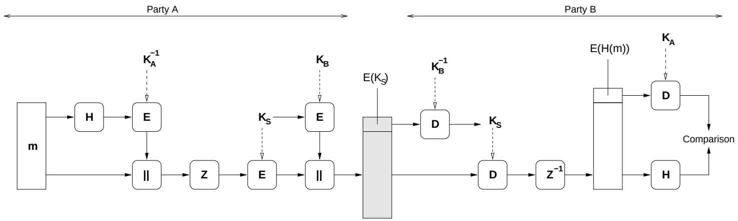
- A computes $c = Z(E_{K_A^{-1}}(H(m))||m)$
- B computes $Z^{-1}(c)$, splits the message and checks the signature by computing $D_{K_A}(E_{K_A^{-1}}(H(m)))$ and then comparing it with the hash $H(m)$.

PGP Confidentiality



- *A* encrypts the message using the key K_s generated by the sender and appended to the encrypted message.
- The key K_s is protected by encrypting it with the public key K_B.

PGP Signatures and Confidentiality



- Signature and confidentiality can be combined as shown above.
- PGP uses in addition Radix-64 encoding (a variant of base-64 encoding) to ensure that messages can be represented using the ASCII character set.
- PGP supports segmentation/reassembly functions for very large messages.

PGP Key Management

- Keys are maintained in so called key rings:
  - one key ring for public keys
  - one key ring for private keys
- Keys are identified by their fingerprints.
- Key generation utilizes various sources of random information (`/dev/random` if available) and symmetric encryption algorithms to generate good key material.
- So called "key signing parties" are used to sign keys of others and to establish a "web of trust" in order to avoid centralized certification authorities.

PGP Private Key Ring

Timestamp	Key ID	Public Key	Encrypted Private Key	User ID
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_i	$K_i \bmod 2^{64}$	K_i	$E_{H(P_i)}(K_i^{-1})$	User_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Private keys are encrypted using $E_{H(P_i)}()$, which is a symmetric encryption function using a key which is derived from a hash value computed over a user supplied passphrase P_i.
- The Key ID is taken from the last 64 bits of the key K_i.

PGP Public Key Ring

Timestamp	Key ID	Public Key	Owner Trust	User ID	Signatures	Sig. Trust(s)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_i	$K_i \bmod 2^{64}$	K_i	otrust_i	User_i	...	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Keys in the public key ring can be signed by multiple parties. Every signature has an associated trust level:
  1. undefined trust
  2. usually not trusted
  3. usually trusted
  4. always trusted
- Computing a trust level for new keys which are signed by others (trusting others when they sign keys).

Transport Layer Security (TLS)

Transport Layer Security

- Transport Layer Security (TLS), formerly known as Secure Socket Layer (SSL), was created by Netscape to secure data transfers on the Web (i.e., to enable commerce on the Web)
- As a user-space implementation, TLS can be shipped as part of applications (Web browsers) and does not require operating system support
- TLS uses X.509 certificates to authenticate servers and clients (although TLS layer client authentication is not often used on the Web)
- TLS is widely used to secure application protocols running over TCP (e.g., http, smtp, ftp, telnet, imap, . . .)
- A datagram version of TLS called DTLS can be used with protocols running over UDP (dns, . . .)

History of TLS and SSL

Name	Organization	Published	Wire Version
SSL 1.0	Netscape	unpublished	1.0
SSL 2.0	Netscape	1995	2.0
SSL 3.0	Netscape	1996	3.0
TLS 1.0	IETF	1999	3.1
TLS 1.1	IETF	2006	3.2
TLS 1.2	IETF	2008	3.3
TLS 1.3	IETF	2018	3.3 + supported_versions

TLS Protocols

- The *Handshake Protocol* authenticates the communicating parties, negotiates cryptographic modes and parameters, and establishes shared keying material.
- The *Alert Protocol* communicates alerts such as closure alerts and error alerts.
- The *Record Protocol* uses the parameters established by the handshake protocol to protect traffic between the communicating peers.
- The Record Protocol is the lowest internal layer of TLS and it carries the handshake and alert protocol messages as well as application data.

TLS Record Protocol

Record Protocol

The record protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, adds a message authentication code, and encrypts and transmits the result. Received data is decrypted, verified, decompressed, reassembled, and then delivered to higher-level clients.

- The record layer is used by the handshake protocol, the change cipher spec protocol (only TLS 1.2), the alert protocol, and the application data protocol.
- The fragmentation and reassembly provided does not preserve application message boundaries.

TLS Handshake Protocol

Handshake Protocol

- Exchange messages to agree on algorithms, exchange random numbers, and check for session resumption.
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret.
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generate a master secret from the premaster secret and the exchanged random numbers.
- Provide security parameters to the record layer.
- Allow client and server to verify that the peer has calculated the same security parameters and that the handshake completed without tampering by an attacker.

TLS Change Cipher Spec Protocol

Change Cipher Spec Protocol

The change cipher spec protocol is used to signal transitions in ciphering strategies.

- The protocol consists of a single ChangeCipherSpec message.
- This message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys.
- This protocol does not exist anymore in TLS 1.3.

TLS Alert Protocol

Alert Protocol

The alert protocol is used to signal exceptions (warnings, errors) that occured during the processing of TLS protocol messages.

- The alert protocol is used to properly close a TLS connection by exchanging `close_notify` alert messages.
- The closure exchange allows to detect truncation attacks.

Secure Shell (SSH)

Secure Shell (SSH)

- SSH provides a secure connection through which user authentication and several inner protocols can be run.
- The general architecture of SSH is defined in RFC 4251.
- SSH was initially developed by Tatu Ylonen at the Helsinki University of Technology in 1995, who later founded SSH Communications Security.
- SSH was quickly adopted as a replacement for insecure remote login protocols such as telnet or rlogin/rsh.
- Several commercial and open source implementations are available running on almost all platforms.
- SSH is a Proposed Standard protocol of the IETF since 2006.

SSH Protocol Layers

1. The **Transport Layer Protocol** provides server authentication, confidentiality, and integrity with perfect forward secrecy
2. The **User Authentication Protocol** authenticates the client-side user to the server
3. The **Connection Protocol** multiplexes the encrypted data stream into several logical channels

$\Rightarrow$ SSH authentication is not symmetric!

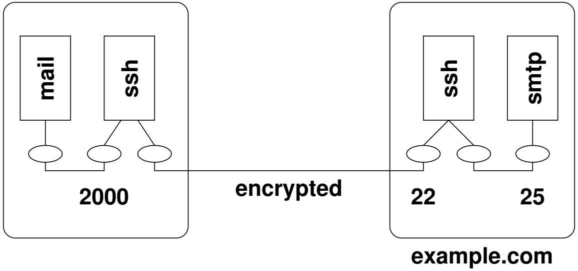
$\Rightarrow$ The SSH protocol is designed for clarity, not necessarily for efficiency (shows its academic roots)

SSH Keys, Passwords, and Passphrases

- **Host key**:
  - Every server must have a public/private host key pair.
  - Host keys are used for server authentication.
  - Host keys are typically identified by their fingerprint.
- **User key**:
  - Users may have their own public/private key pairs, optionally used to authenticate users.
- **User password**:
  - Remote accounts may use passwords to authenticate users.
- **Passphrase**:
  - The storage of a user's private key may be protected by a passphrase.

SSH Features: TCP Forwarding

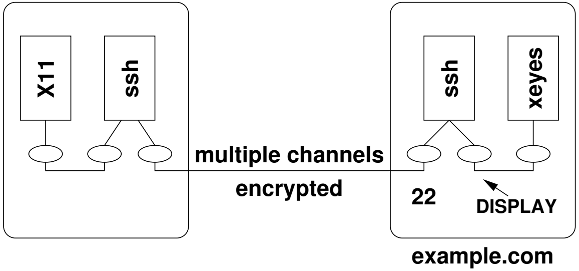
```
ssh -f joe@example.com -L 2000:example.com:25 -N
```



- TCP forwarding allows users to tunnel unencrypted traffic through an encrypted SSH connection.

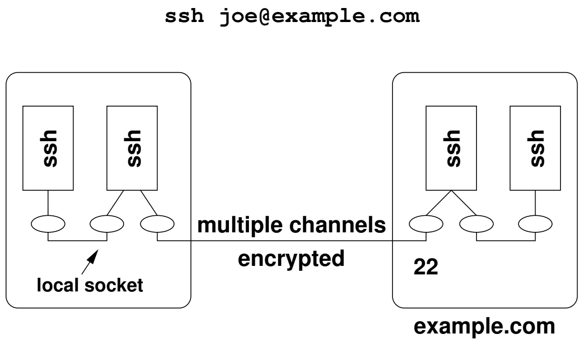
SSH Features: X11 Forwarding

```
ssh -X joe@example.com
```



- X11 forwarding is a special application of TCP forwarding allowing X11 clients on remote machines to access the local X11 server (managing the display and the keyboard/mouse).

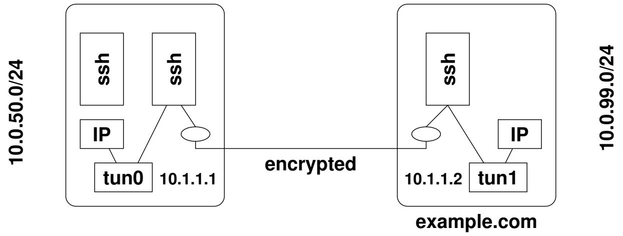
SSH Features: Connection Sharing



- New SSH connections hook as a new channel into an existing SSH connection, reducing session startup times (speeding up shell features such as tab expansion).

SSH Features: IP Tunneling

```
ssh -f -w 0:1 example.com
```

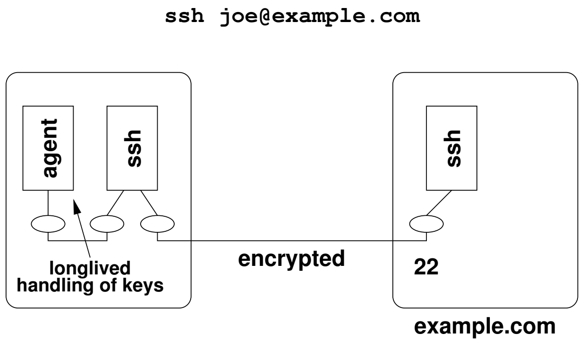


```
ifconfig tun0 10.1.1.1 10.1.1.2 \
        netmask 255.255.255.255
route add 10.0.99.0/24 10.1.1.2
```

```
ifconfig tun0 10.1.1.2 10.1.1.1 \
        netmask 255.255.255.255
route add 10.0.50.0/24 10.1.1.1
```

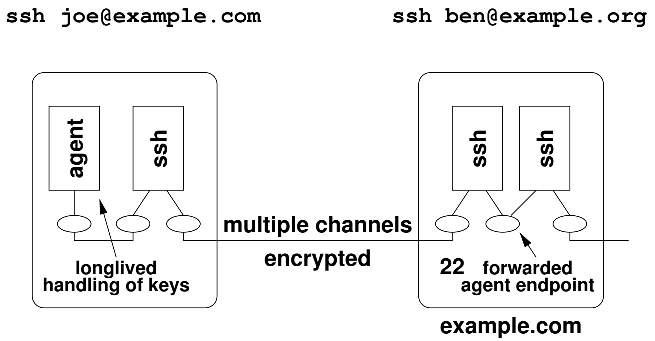
- Tunnel IP packets over an SSH connection by inserting tunnel interfaces into the kernels and by configuring IP forwarding.

SSH Features: SSH Agent



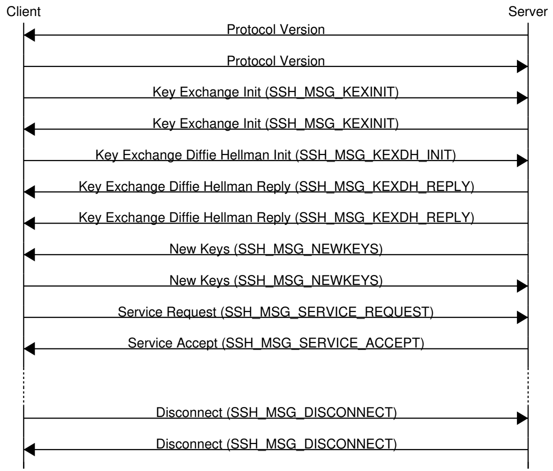
- Maintains client credentials during a login session so that credentials can be reused by different SSH invocations without further user interaction.

SSH Features: SSH Agent Forwarding



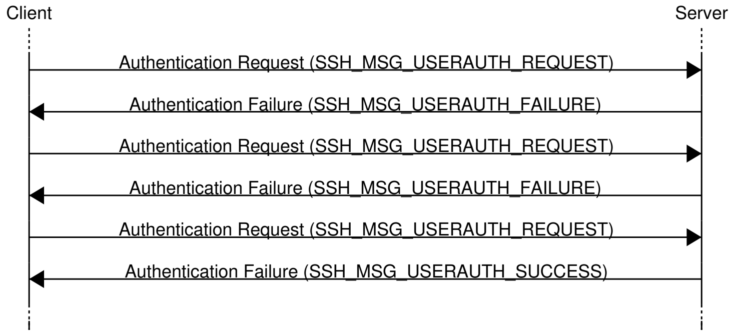
- An SSH server emulates an SSH Agent and forwards requests to the SSH Agent of its client, creating a chain of SSH Agent delegations.

SSH Transport Protocol



Client
Server
Protocol Version
Protocol Version
Key Exchange Init (SSH_MSG_KEXINIT)
Key Exchange Init (SSH_MSG_KEXINIT)
Key Exchange Diffie Hellman Init (SSH_MSG_KEXDH_INIT)
Key Exchange Diffie Hellman Reply (SSH_MSG_KEXDH_REPLY)
Key Exchange Diffie Hellman Reply (SSH_MSG_KEXDH_REPLY)
New Keys (SSH_MSG_NEWKEYS)
New Keys (SSH_MSG_NEWKEYS)
Service Request (SSH_MSG_SERVICE_REQUEST)
Service Accept (SSH_MSG_SERVICE_ACCEPT)
Disconnect (SSH_MSG_DISCONNECT)
Disconnect (SSH_MSG_DISCONNECT)

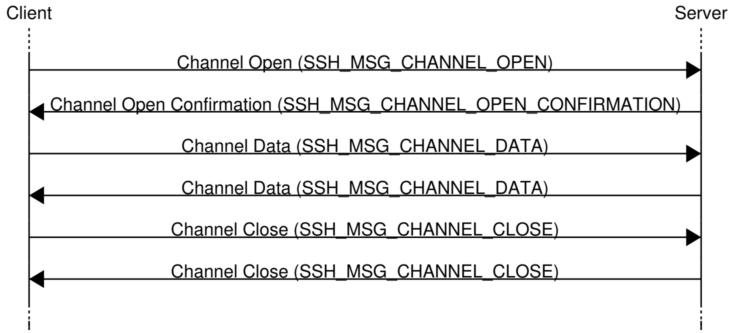
SSH User Authentication



Client
Server
Authentication Request (SSH_MSG_USERAUTH_REQUEST)
Authentication Failure (SSH_MSG_USERAUTH_FAILURE)
Authentication Request (SSH_MSG_USERAUTH_REQUEST)
Authentication Failure (SSH_MSG_USERAUTH_FAILURE)
Authentication Request (SSH_MSG_USERAUTH_REQUEST)
Authentication Failure (SSH_MSG_USERAUTH_SUCCESS)

- The user authentication protocol iterates through a list of mechanisms until either authentication was successful or all mechanisms have failed.

SSH Connection Protocol


Client
Server
Channel Open (SSH_MSG_CHANNEL_OPEN)
Channel Open Confirmation (SSH_MSG_CHANNEL_OPEN_CONFIRMATION)
Channel Data (SSH_MSG_CHANNEL_DATA)
Channel Data (SSH_MSG_CHANNEL_DATA)
Channel Close (SSH_MSG_CHANNEL_CLOSE)
Channel Close (SSH_MSG_CHANNEL_CLOSE)

- The connection protocol has additional messages to handle control flow, error messages (equivalent of stderr), and end-of-file indicators.

OpenSSH Privilege Separation

- Privilege separation is a technique in which a program is divided into parts which are limited to the specific privileges they require in order to perform a specific task.
- OpenSSH is using two processes: one running with special privileges and one running under normal user privileges
- The process with special privileges carries out all operations requiring special permissions.
- The process with normal user privileges performs the bulk of the computation not requiring special rights.
- Bugs in the code running with normal user privileges do not give special access rights to an attacker.

DNS Security Extensions (DNSSEC)

Part: Information Hiding and Privacy

Steganography and Watermarks

Information Hiding

Definition (information hiding)

Information hiding aims at concealing the very existence of some kind of information for some specific purpose.

- Information hiding itself does not aim at protecting message content
- Encryption protects message content but by itself does not hide the existence of a message
- Information hiding techniques are often used together with encryption in order to both hide the existence of messages and to protect messages in case their existence is uncovered

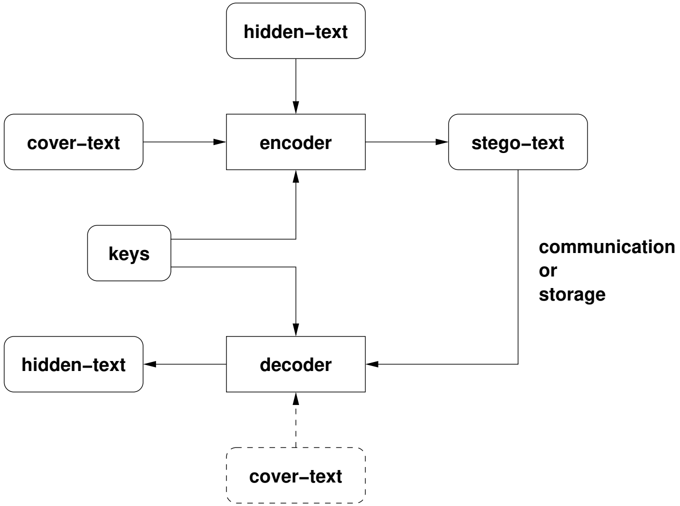
Steganography

Definition (steganography)

Steganography is the embedding of some information (hidden-text) within digital media (cover-text) so that the resulting digital media (stego-text) looks unchanged (imperceptible) to a human/machine.

- Information hiding explores the fact that there are often (almost) unused or redundant bits in digital media that can be used to carry hidden digital information.
- The challenge is to identify (almost) unused or redundant bits and to encode hidden digital information in them in such a way that the existence of hidden information is difficult to observe.

Steganography Workflow



Types of Cover Media

- Information can be hidden in various cover media types:
  - Image files
  - Audio files
  - Video files
  - Text files
  - Software (e.g., executable files, source code)
  - Network traffic (e.g., covert channels)
  - Storage devices (e.g., steganographic file systems)
  - Events (e.g., timing covert channels, signaling covert channels)
  - ...
- Media types of large size usually make it easier to hide information.
- Robust steganographic methods may survive some typical modifications of stego-texts (e.g., cropping or recoding of images).

Watermarking

Definition (watermarking)

Watermarking is the embedding some information (watermark) within a digital media (cover-text) so that the resulting digital media looks unchanged (imperceptible) to a human/machine.

- Watermarking:
  - The hidden information itself is not important.
  - The watermark says something about the cover-text.
- Steganography:
  - The cover-text is not important, it only conveys the hidden information.
  - The hidden text is the valuable information, and it is independent of cover-text.

Classification of Steganographic Algorithms

- fragile vs. robust
    - Fragile: Modifiations of stego-text likly destroys hidden text.
    - Robust: Hidden text is likely to survive modifications of the stego-text.
- blind vs. semi-blind vs. non-blind
    - Blind requires the original cover-text for detection / extraction.
    - Semi-blind needs some information from the embedding but not the whole cover-text
    - Non-blind does not need any information for detection / extraction.
- pure vs. symmetric (secret key) vs. asymmetric (public key)
    - Pure needs no key for detection / extraction.
    - Secret key needs a symmetric key for embedding and extraction.
    - Public key needs a secret key for embedding and a public key for extraction.

Example: LSB-based Image Steganography

- Idea:
  - Some image formats encode a pixel using three 8-bit color values (red, green, blue).
  - Changes in the least-significant bits (LSB) are difficult for humans to see.
- Approach:
  - Use a key to select some least-significant bits of an image to embed hidden information.
  - Encode the information multiple times to achieve some robustness against noise.
- Problem:
  - Existence of hidden information may be revealed if the statistical properties of least-significant bits change.
  - Fragile against noise such as compression, resizing, cropping, rotating or simply additive white Gaussian noise.

Example: DCT-based Image Steganography

- Idea:
  - Some image formats (e.g., JPEG) use discrete cosine transforms (DCT) to encode image data.
  - The manipulation happens in the frequency domain instead of the spatial domain and this reduces visual attacks against the JPEG image format.
- Approach:
  - Replace the least-significant bits of some of the discrete cosine transform coefficients.
  - Use a key to select some DCT coefficients of an image to embed hidden information.
- Problem:
  - Existence of hidden information may be revealed if the statistical properties of the DCT coefficients are changed.
  - This risk may be reduced by using an pseudo-random number generator to select coefficients.

Covert Channels

Covert Channels

- Covert channels represent unforeseen communication methods that break security policies. Network covert channels transfer information through networks in ways that hide the fact that communication takes place (hidden information transfer).
- Covert channels embed information in
  - header fields of protocol data units (protocol messages)
  - the size of protocol data units
  - the timing of protocol data units (e.g., inter-arrival times)
- We are not considering here covert channels that are constructed by exchanging steganographic objects in application messages.

Covert Channel Patterns

P1 Size Modulation Pattern
The covert channel uses the size of a header field or of a protocol message to encode hidden information.

P2 Sequence Pattern
The covert channel alters the sequence of header fields to encode hidden information.

P3 Add Redundancy Pattern
The covert channel creates new space within a given header field or within a message to carry hidden information.

P4 PDU Corruption/Loss Pattern
The covert channel generates corrupted protocol messages that contain hidden data or it actively utilizes packet loss to signal hidden information.

Covert Channel Patterns

P5 Random Value Pattern
The covert channel embeds hidden data in a header field containing a “random” value.

P6 Value Modulation Pattern
The covert channel selects one of values a header field can contain to encode a hidden message.

P7 Reserved/Unused Pattern
The covert channel encodes hidden data into a reserved or unused header field.

P8 Inter-arrival Time Pattern
The covert channel alters timing intervals between protocol messages (inter-arrival times) to encode hidden data.

Covert Channel Patterns

P9 Rate Pattern
The covert channel sender alters the data rate of a traffic flow from itself or a third party to the covert channel receiver.

P10 Protocol Message Order Pattern
The covert channel encodes data using a synthetic protocol message order for a given number of protocol messages flowing between covert sender and receiver.

P11 Re-Transmission Pattern
A covert channel re-transmits previously sent or received protocol messages.

Anonymization Terminology

Anonymity

Definition (anonymity)

Anonymity of a subject from an attacker's perspective means that the attacker cannot sufficiently identify the subject within a set of subjects, the anonymity set.

- All other things being equal, anonymity is the stronger, the larger the respective anonymity set is and the more evenly distributed the sending or receiving, respectively, of the subjects within that set is.
- Robustness of anonymity characterizes how stable the quantity of anonymity is against changes in the particular setting, e.g., a stronger attacker or different probability distributions.

Unlinkability and Linkability

Definition (unlinkability)

Unlinkability of two or more items of interest (IOIs) (e.g., subjects, messages, actions, . . .) from an attacker's perspective means that within the system, the attacker cannot sufficiently distinguish whether these IOIs are related or not.

Definition (linkability)

Linkability of two or more items of interest (IOIs) (e.g., subjects, messages, actions, . . .) from an attacker's perspective means that within the system, the attacker can sufficiently distinguish whether these IOIs are related or not.

Undetectability and Unobservability

Definition (undetectability)

Undetectability of an item of interest (IOI) from an attacker's perspective means that the attacker cannot sufficiently distinguish whether it exists or not.

Definition (unobservability)

Unobservability of an item of interest (IOI) means

- undetectability of the IOI against all subjects uninvolved in it and
- anonymity of the subject(s) involved in the IOI even against the other subject(s) involved in that IOI.

Relationships

With respect to the same attacker, the following relationships hold:

- unobservability $\Rightarrow$ anonymity
- sender unobservability $\Rightarrow$ sender anonymity
- recipient unobservability $\Rightarrow$ recipient anonymity
- relationship unobservability $\Rightarrow$ relationship anonymity

We also have:

- sender anonymity $\Rightarrow$ relationship anonymity
- recipient anonymity $\Rightarrow$ relationship anonymity
- sender unobservability $\Rightarrow$ relationship unobservability
- recipient unobservability $\Rightarrow$ relationship unobservability

Pseudonymity

Definition (pseudonym)

A *pseudonym* is an identifier of a subject other than one of the subject's real names. The subject, which the pseudonym refers to, is the holder of the pseudonym.

Definition (pseudonymity)

A subject is *pseudonymous* if a pseudonym is used as identifier instead of one of its real names. *Pseudonymity* is the use of pseudonyms as identifiers.

Identifiability and Identity

Definition (identifiability)

Identifiability of a subject from an attacker's perspective means that the attacker can sufficiently identify the subject within a set of subjects, the identifiability set.

Definition (identity)

An identity is any subset of attribute values of an individual person that sufficiently identifies this individual person within any set of persons. So usually there is no such thing as "the identity", but several of them.

Identity Management

Definition (identity management)

Identity management means managing various partial identities (usually denoted by pseudonyms) of an individual person, i.e., administration of identity attributes including the development and choice of the partial identity and pseudonym to be (re-)used in a specific context or role.

- A partial identity is a subset of attribute values of a complete identity, where a complete identity is the union of all attribute values of all identities of this person.
- A pseudonym might be an identifier for a partial identity.

Mixes and Onion Routing

Mix Networks

- A mix network uses special proxies called mixes to send data from a source to a destination.
- The mixes filter, collect, recode, and reorder messages in order to hide conversations. Basic operations of a mix:
  1. Removal of duplicate messages (an attacker may inject duplicate message to infer something about a mix).
  2. Collection of messages in order to create an ideally large anonymity set.
  3. Recoding of messages so that incoming and outgoing messages cannot be linked.
  4. Reordering of messages so that order information cannot be used to link incoming and outgoing messages.
  5. Padding of messages so that message sizes do not reveal information to link incoming and outgoing messages.

Onion Routing

- A message m it sent from the source S to the destination T via an overlay network consisting of the intermediate routers $R_1, R_2, \ldots, R_n$, called a circuit.
- A message is cryptographically wrapped multiple times such that every router R unwraps one layer and thereby learns to which router the message needs to be forwarded next.
- To preserve the anonymity of the sender, no node in the circuit is able to tell whether the node before it is the originator or another intermediary like itself.
- Likewise, no node in the circuit is able to tell how many other nodes are in the circuit and only the final node, the "exit node", is able to determine its own location in the chain.

Tor

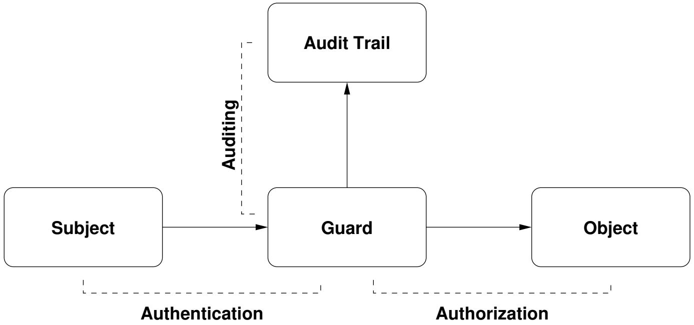
- Tor is an anonymization network operated by volunteers supporting the Tor project.
- Every Tor router has a long-term identity key and a short-term onion key.
- The identity key is used to sign TLS certificates and the onion key is used to decrypt messages to setup circuits and ephemeral keys.
- TLS is used to protect communication between onion routers.
- Directory servers provide access to signed state information provided by Tor routers.
- Applications build circuits based on information provided by directory servers.

Part: System Security

Authentication, Authorization, Auditing, Isolation

- Authentication
  - Who is requesting an action?
- Authorization
  - Is a principal allowed to execute an action on this object?
- Auditing
  - Record evidence for decision being made in an audit-trail.
- Isolation
  - Isolate system components from each other to create sandboxes.

Lampson Model



- This basic model works well for modeling static access control systems.
- Dynamic access control systems allowing dynamic changes to the access control policy are difficult to model with this approach.

Isolation

- Isolation is a fundamental technique to increase the robustness of computing systems and to reduce their attack surface.
- Isolation can be achieved in many different layers of a computing system:
  - Physical (e.g., preventing physical access to compute clouds)
  - Hardware (e.g., trusted execution environments, memory management units)
  - Virtualization (e.g., virtual machines, containers)
  - Operating System (e.g., processes, file systems)
  - Network (e.g., virtual LANs, virtual private networks)
  - Applications (e.g., transaction isolation in databases)
- Isolation should be a concern of every system design.
- Isolation also concerns the deployment of computing systems.

Trusted Computing

Trusted Computing Base

Definition (trusted computing base)

The *trusted computing base* of a computer system is the set of hard- and software components that are critical to achieve the systems' security properties.

- The components of a trusted computing base are designed such that when other parts of a system are attacked, the device will not misbehave.
- Trusted computing bases should be small in order to be able to verify their correctness.
- Trusted computing bases should be tamper-resistant.
- Trusted computing bases typically involve special hardware components.

Trusted Computing Security Goals

- *Isolation*: Separation of essential security critical functions and associated data (keys) from the general computing system.
- *Attestation*: Proving to an authorized party that a specific component is in a certain state.
- *Sealing*: Wrapping of code and data such that it can only be unwrapped and used under certain circumstances.
- *Code Confidentiality*: Ensures that sensitive code and static data cannot be obtained by untrusted hardware or software.
- *Side-Channel Resistance*: Ensures that untrusted components are not able to deduce information about the internal state of a trusted computing component.
- *Memory Protection*: Protects the integrity and authenticity of data sent over system buses or stored in (external) memory from physical attacks.

Trusted Platform Module (TPM)

- A Trusted Platform Module (TPM) is a dedicated micro-controller designed to secure hardware through integrated cryptographic operations and key storage.
- The TPM 1.2 specification was published in 2011:
  - Co-processor capable of generating good random numbers, storing keys, performing cryptographic operations, and providing the basis for attestation.
  - Limited protection against physical attacks.
- The TPM 2.0 specification was published in 2014.
  - Support of a larger set of cryptographic algorithms and more storage space for attestation purposes.
- The TPM specifications have been created by the Trusted Computing Group (a consortium of vendors with large influence of Microsoft on TPM 2.0).

Trusted Execution Environment (TEE)

Definition (trusted and rich execution environment)

A *trusted execution environment* (TEE) is a secure area of a processor providing isolated execution, integrity of trusted applications, as well as confidentiality of trusted application resources. A *rich execution environment* (REE) is the non-secure area of a processor where an untrusted operating system executes.

- REE resources are accessible from the TEE
- TEE resources are accessible from the REE only if explicitly allowed.
- The TEE specifications have been created by the GlobalPlatform (another industry consortium).

TrustZone Cortex-A (ARM)

- The ARM processor architecture has an internal communication interface called the Advanced eXtensible Interface (AXI).
- ARM's TrustZone extends the AXI bus with a Non-Secure (NS) bit.
- The NS bit conveys whether the processor works in secure mode or in normal mode.
- The processor is normally executing in either secure or normal mode.
- To perform a context switch (between modes), the processor transits through a monitor mode.
- The monitor mode saves the state of the current world and restores the state of the world being switched to.
- Interrupts may trap the processor into monitor mode if the interrupt needs to be handled in a different mode.

TrustZone Cortex-M (ARM)

- The Cortex-M design follows the Cortex-A design by having the processor execute in either secure or normal mode.
- Instructions read from secure memory will be executed in the secure mode of the processor and instructions read from non-secure memory will be executed in normal mode.
- Cortex-M replaces the monitor mode of the Cortex-A design with a faster mechanism to call secure code via multiple secure function entry points (supported by the machine instructions `SG`, `BXNS`, `BLXNS`).
- The Cortex-M design supports multiple separate call stacks and the memory space is separated into secure and non-secure sections.
- Interrupts can be configured to be handled in secure or non-secure mode.

Security Guard Extension (SGX, Intel)

- SGX places the protected parts of an application in so called enclaves that can be seen as a protected module within the address space of a user space process.
- SGX enabled CPUs ensure that non-enclaved code, including the operating system and potentially the hypervisor, cannot access enclave pages.
- A memory region called the Processor Reserved Memory (PRM) contains the Enclave Page Cache (EPC) and is protected by the CPU against non-enclave accesses.
- The content of enclaves is loaded when enclaves are created and measurements are taken to ensure that the content loaded is correct.
- The measurement result obtained during enclave creation may be used for (remote) attestation purposes.
- Entering an enclave is realized like a system call and supported by special machine instructions (`EENTER`, `EEXIT`, `ERESUME`).

Authentication

Authentication

Definition (authentication)

Authentication is the process of verifying a claim that a system entity or system resource has a certain attribute value.

- An authentication process consists of two basic steps:
  1. Identification step: Presenting the claimed attribute value (e.g., a user identifier) to the authentication subsystem.
  2. Verification step: Presenting or generating authentication information (e.g., a value signed with a private key) that acts as evidence to prove the binding between the attribute and that for which it is claimed.
- Security services frequently depend on authentication of the identity of users, but authentication may involve any type of attribute that is recognized by a system.

Authentication Factors

- Something you know (knowledge factors)
    - Your password, first own music album, personal identification number, ...
- Something you have (possession factors)
    - Your mobile device, security token, software token, ...
- Something you are (static biometrics)
    - Your fingerprint, retina, face, ...
- Something you do (dynamic biometrics)
    - Your voice, signature, typing rhythm, ...

- Multi-factor authentication uses multiple factors to authenticate a user.
- Two-factor authentication is increasingly used these days.

Password Authentication

Definition (password authentication)

A *password* is a secret data value, usually a character string, that is presented to a system by a user to authenticate the user's identity.

- Never ever store passwords in cleartext on a server (or elsewhere).
- A common approach is to store $H(s\|p)$ where H is a cryptographic hash function, p is the password, and s is a random value (the salt).
- The salt ensures that multiple occurrences of the same password does not lead to the same hash values.

Challenge-Response Authentication

Definition (challenge-response authentication)

Challenge-response authentication is an authentication process that verifies an identity by requiring correct authentication information to be provided in response to a challenge. In a computer system, the authentication information is usually a value that is required to be computed in response to an unpredictable challenge value, but it might be just a password.

- Password authentication can be seen as a special case of a challenge-response authentication process.
- In some protocols the server sends a challenge to the client in the form of a random value and the client responds with the value return by a cryptographic hash function computed over the random value and a password (shared with the server).

One-Time Password Authentication

- Let $H^n(m)$ denote the repeated application, n-times, of the function H to m.
- **Initialization**: Given a passphrase p and a salt s, computes $k = H^{n+1}(s\|p)$ and the authentication server remembers k and n.
- **Challenge**: The authentication server sends the name of the hash function H, the salt s, and the current value of n to the user.
- **Response**: The user computes $q = H^n(s\|p)$ and sends the value q back to the server.
- **Verification**: The server computes $H(q) = H(H^n(s\|p)) = H^{n+1}(s\|p)$ and checks whether it matches k. If it matches, the server sets $k = q$ and n is decremented. If n becomes 0, a new initialization must be performed.

Token Authentication

Definition (token authentication)

Token authentication verifies the claim of an identity by proving the possession of a (hardware) token.

- Smart cards are credit-card sized devices containing one or more chips that perform the functions of a computer's central processor, memory, and input/output interface.
- A smart token is a device that conforms to the definition of a smart card except that rather than having the standard dimensions of a credit card, the token is packaged in some other form, such as a military dog tag or a door key.
- Mobile devices are sometimes uses as a token in today's multi-factor authentication systems.

Biometric Authentication

Definition (biometric authentication)

Biometric authentication is a method of generating authentication information for a person by digitizing measurements of a physical or behavioral characteristic, such as a fingerprint, hand shape, retina pattern, voiceprint, handwriting style, or face.

- Sensors that read biometric data must be designed such that they can detect fake copies of biometric data. Fingerprint sensors, for example, try to detect blood flows.

Authorization

Subjects, Objects, Rights

- Subjects (S): set of active objects
  - processes, users, ...
- Objects (O): set of protected entities
  - files, directories, ...
  - memory, devices, sockets, ...
  - processes, memory, ...
- Rights (R): set of operations a subject can perform on an object
  - create, read, write, delete ...
  - execute ...

Lampson's Access Control Matrix

Definition (access control matrix)

An *access control matrix* M consists of subjects $s_i \in S$, which are row headings, and objects $o_j \in O$, which are column headings. The access rights $r_{i,j} \in R^*$ of subject s_i when accessing object o_j are given by the value in the cell $r_{i,j} = M[s_i, o_j]$.

- Another way to look at access control rights is that the access rights $r \in R^*$ are defined by a function $M : (S \times O) \to R^*$.
- Since the access control matrix can be huge, it is necessary to find ways to express it in a format that is lowering the cost for maintaining it.

Access Control Lists

Definition (access control list)

An access control list represents a column of the access control matrix. Given a set of subjects S and a set of rights R, an access control list of an object $o \in O$ is a set of tuples of $S \times R^*$.

- Example: The inode of a traditional Unix file system (the object) stores the information whether a user or a group (the subject) or all users have read/write/execute permissions (the rights).
- Example: A database system stores for each database (the object) information about which operations (the rights) users (the subjects) can perform on the database.

Capabilities

Definition (capabilities)

A capability represents a row of the access control matrix. Given a set of objects o and a set of rights R, a capability of a subject s is a set of tuples of $O \times R^*$.

- Example: An open Unix file descriptor can be seen as a capability. Once opened, the file can be used regardless whether the file is deleted or whether access rights are changed. The capability (the open file descriptor) can be transferred to child processes. (Note that passing capabilities to child processes is not meaningful for all capabilities.)

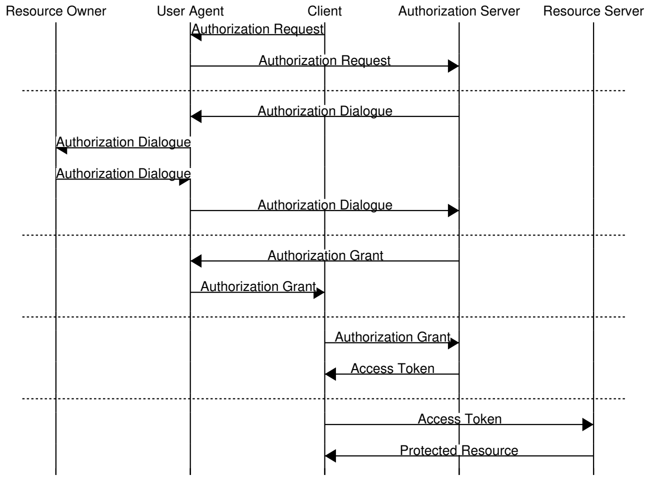
Access Control Lists vs. Capabilities

- Both are theoretically equivalent (since both at the end can represent the same access control matrix).
- Capabilities tend to be more efficient if the common question is “Given a subject, what objects can it access and how?”.
- Access control lists tend to be more efficient if the common question is “Given an object, what subjects can access it and how?”.
- Access control lists tend to be more popular because they are more efficient when an authorization decision needs to be made.
- Systems often use a mixture of both approaches.

Discretionary, Mandatory, Role-based Access Control

- Discretionary Access Control (DAC)
  - Subjects (with certain permissions, i.e., ownership) can define access control rules to allow or deny access to an object
  - It is the at the subject's discretion what to allow to whom.
- Mandatory Access Control (MAC)
  - System mechanisms control access to an object and an individual subject cannot alter the access rights.
  - What is allowed is mandated by the (security administrator of the) system.
- Role-based Access Control (RAC)
  - Subjects are first mapped to a set of roles that they have.
  - Mandatory access control rules are defined for roles instead of subjects.

API Authorization (OAuth 2.0)



Auditing

Auditing

Definition (auditing)

Auditing is the process of collecting information about security-related events in an audit log, also called an audit trail.

- Audit logs are necessary for performing forensic investigation and for identifying and tracking ongoing attacks.
- Examples of security-related events that are typically logged are (failed) login attempts, failed attempts to obtain additional privileges, information about who accesses a system when, unusual failures of security protocols etc.
- Unix systems use logging daemons to receive, filter, forward, and store system logs originating from the kernel and background daemons.

Audit Log Processing

- Audit logs can become very large and a common approach is to rotate logs periodically (say every day) and to keep only a history of the last *N* days or weeks.
- Audit logs often consist of semi-structured information, which automated processing of logged information challenging.
- Audit logs often contain a lot of noise (information about events that are not security-related in a given deployment) and finding relevant information often becomes a search for an unknown needle in a haystack.
- There are tools that automatically filter logged messages and generate reports summarizing events that were not classified as expected and harmless.
- Maintaining good filter rules takes effort and obviously filter rules must be maintained in such a way that an attacker cannot modify them.