Introduction

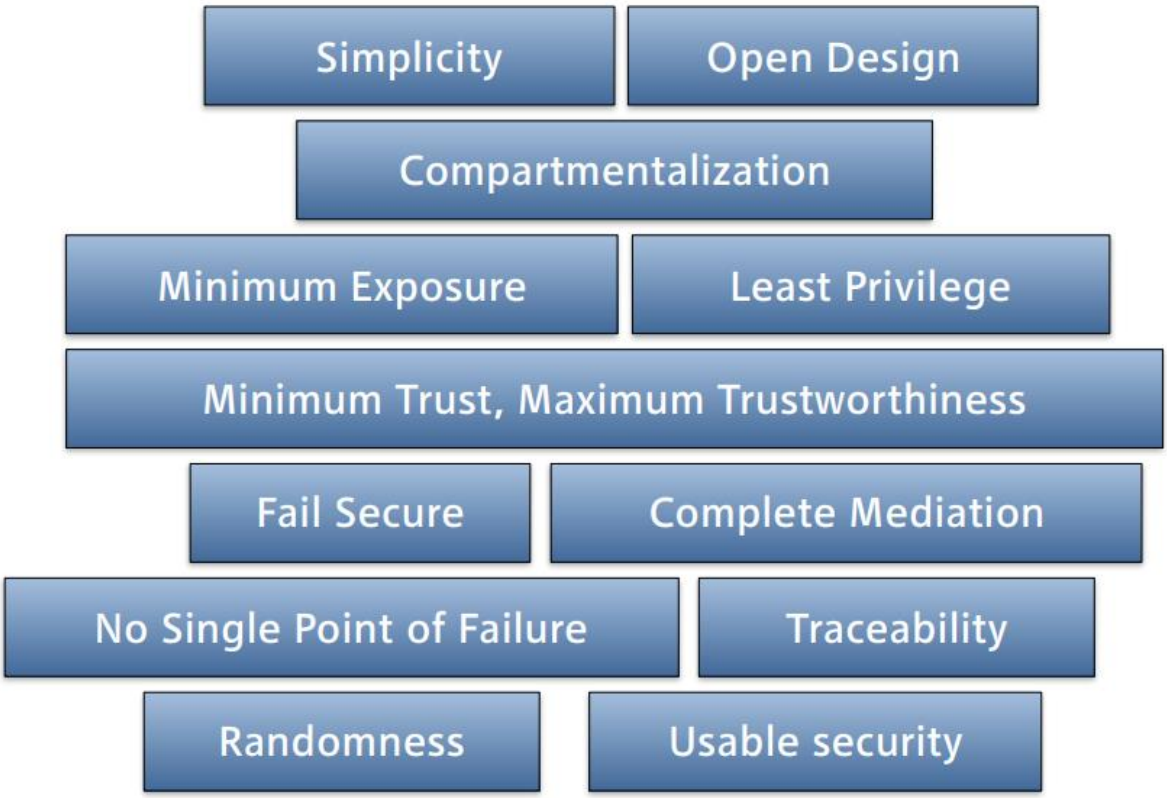
Terminology

Computer security is the protection, which afforded an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).

Terminology list

Data Confidentiality	Assures that private or confidential information is not made available or disclosed to unauthorized individuals
Privacy	assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed
Data integrity	Assures that information and programs are changed only in a specified and authorized manner
System integrity	Assures that a system performs its intended function in an unimpaired manner
Availability	Assures that systems work promptly and service is not denied to authorized users
Authenticity	The property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator
Accountability	The security goal that generates the requirement for actions of an entity to be traced uniquely to that entity
Adversary	An entity that attacks, or is a threat to, a system
Attack	An assault on system security that derives from an intelligent threat; a deliberate attempt to evade security services and violate security policy of a system
Countermeasure	An action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken
Risk	An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result
Security policy	A set of rules and practices to be followed in order protect sensitive and critical system resources
System resource (asset)	Something worthy to be protected
Threat	A potential violation of security
Vulnerability	Flaw or weakness in a system's design

Security Principles



System Security

Authentication

Identification: Whom you claim to be

Authentication: How you prove your ID

Credentials: Evidence used to prove the ID

Password Authentication

Password Authentication is the most common authentication method.
The basic idea is, the user has a secret password and the system checks password to authenticate user.
Issues:

- How is password stored?
- How does a system check passwords?
- How easy is it to guess a password?

A password is computed with a hash-function, which should have the following properties:

- One-way functions: Easy to compute output from input, infeasible to compute input from output
- Collision-resistance: Infeasible to find two different inputs that map to the same output

To compute the password, we have a cryptographic hash function `h: strings -> strings`. When we have given h(password) it is hard to find password. The user-password is stored as h(password). When user enters password the system computes h(password) and compares with entry in password file. Therefore, we have no plaintext passwords stored on disk.

Salt is a reandom Number per User, which inflicts on the computation of the hash-function. Without salt the attacker can pre-compute hashes of all common passwords once. On all UNIX machines, we have the same hash functions. Therefore, we have identical passwords hash to identical values. One table of hash values works for all password files.

With Salt, the attacker must compute hashes of all common passwords for each possible salt value (e.g., with 8-character salt: 248 precomputed hashes per possible password (→ file too large, storage problem)). Dictionary attack on known password file: You have to try all common strings for each salt found in file. Important: Salt like password hash are publicly readable.

When you use password-authentication you must be aware: People tend to reuse the same passwords in different places. If one site is compromised, the password can be stolen and used elsewhere an attacker can use stolen credentials in automated scripts on a large-scale (“Credential stuffing”).

Password managers allow complex, long, unique password per account. They store password in an encrypted file, so we have a potential single point of failure. People use multiple machines so we must

synchronize the password database. Managers that can securely recognize websites can help protect against phishing.

Biometrics

Use a person's physical characteristics (fingerprint, voice, face, keyboard timing, …). The advantage is, that it cannot be disclosed, lost, forgotten.
Disadvantages are

- Cost, installation, maintenance
- Reliability of comparison algorithms (False positive, False negative)
- Privacy?
- If forged, how do you revoke it?

Hardware Authentication Tokens

Specification by the FIDO (Fast IDentity Online) Alliance and is heavily supported by Google. It either generates one-time passwords or challenge-response authentication using a secure hardware key. The device may communicate via USB, NFC, etc and is integrated into Chrome & Firefox, supported by many services (e.g., Dropbox, GitHub, Facebook, …).

Advantages:

- You don't need to remember any password
- Password reuse is not an issue

Disadvantages:

- What if the device is lost?
- You have to bring device always with you
- Relatively expensive

Two-Factor Authentication (2FA)

Two-Factor-Authentication combines two of the three types of authentication. We use second factor to work around limitations of first. An important security requirement is, that both channels are independent from each other, so compromise of one channel does not compromise the other.

Examples:

- Online banking password plus generated TAN
- Online banking password plus mTAN (unless banking done on same phone that receives mTAN)
- Account password plus authenticator
- Account password plus hardware token, e.g., FIDO U2F (Universal 2nd Factor) token

Other Considerations & Takeaways

We must consider **Usability** (Hard-to-remember passwords? Carry a physical object all the time?), **Denial of service**, e.g. an attacker tries to authenticate as you, account locked after three failure and **Availability**, with Mechanism for password recovery.

Access Control

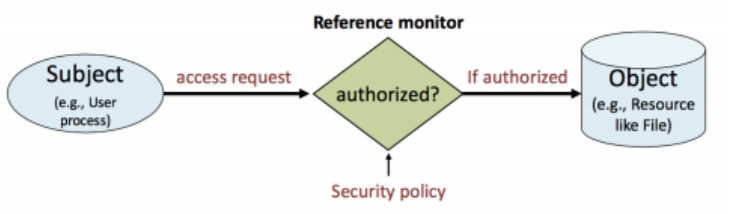
Authorizes requests by subjects to perform operations on objects.

Reference Monitor: Access enforcement mechanism (e.g. Part of the virtual file system in the OS)

Security Policy: Rules to describe the operations that a subject can perform on objects. It defines how the rules can be modified.

Reference Monitor

Assumption: System knows who the subject is.
A access requests pass through a gatekeeper.
The subject needs to fulfil well-defined properties.



Properties:

- *Complete Mediation*: System must not allow monitor to be bypassed
- *Tamperproof*: Monitor must be protected from compromise
- *Verifiable*: Monitor should be small enough to be analyzed. It should be verifiable that the system provides its security goals

Access Control Matrix

A matrix with objects (files) for the columns and subjects (users) for the rows. Cell(s,o) with s ∈ Subjects and o ∈ Objects defines the access rights of subject s for object o. Access rights are {-, read, write}.

Objects

Subjects	File 1	File 2	File 3	…	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
…					
User m	read	write	read	write	read

Access Control List: is Object-centered (for each file). It relies on authentication (need to know user).

Capabilities: is Subject-centered. Capability is unforgeable ticket (token) that defines the privileges of its holder.

Roles: More users can have the same poriviliges: They are in the same rule. Rules can have a hierarchy.

Access Control Types

A control-type can be **private** (e.g. Keys for your flat or login to your laptop) or **centrally-managed** (e.g. Keys for your office, Iris scan for high security rooms).

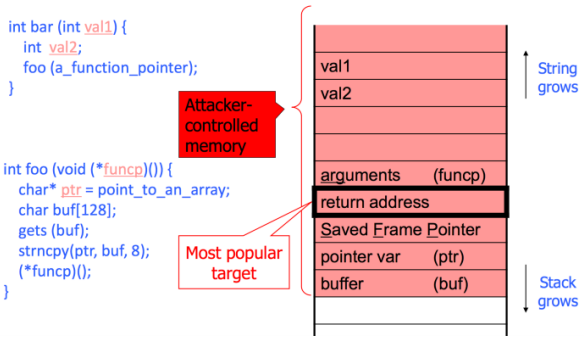
In a **Discretionary Access Control (DAC)** Subjects can freely delegate/revoke/modify access rights to objects for which they have certain access rights (e.g. that they own, for which they have a capability).

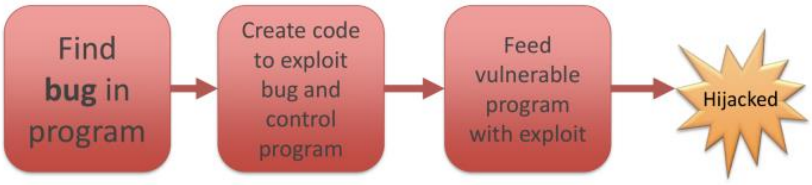
In a Mandatory Access Control (MAC) the security policy is set and modified centrally by trusted administrator. Users and untrusted processes cannot override the policy (e.g., delegate rights if not allowed by policy). Subjects and objects usually labeled with security attributes. Access rules describe allowed operations between labels. A transition rule describes how labels of subjects/objects can change.

Memory Attacks

Control-Flow Hijacking Attacks

The attacker's goal is it to take over target machine and execute arbitrary code on target by hijacking application control flow. The attack-pattern is always similar:





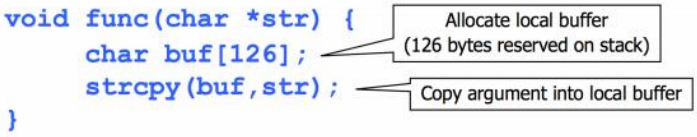
Programmers write their code in a higher language, which comes with some assumptions. These don't have always to be true:

- Basic statements are atomic. - Each basic statement compiled down to many instructions.
- Only one of the branches of an if statement can be taken. - There is no restriction on the target of a jump.
- Functions start at the beginning. - Can start executing in the middle of functions.
- A function executes from beginning to end. - A fragment of a function may be executed.
- And, when done, they return to their call site. - Returns can go to any program instruction.
- Only the code in the program can be executed - Dead code can be executed

Buffer Overflow

A common assumption is that a target buffer is large enough for source data.

When this function is invoked, a new frame (activation record) is pushed onto the stack. Then the Memory pointed to by str is copied onto stack. If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations.



When the buffer contains an attacker-created string, and the function exists, the code in the buffer will be executed and giving attacker a shell.

Basic Stack Code Injection

The executable attack code is stored on stack, inside the buffer containing attacker's string. The stack memory is supposed to contain only data, but this is not checked. For the basic stack-smashing attack, overflow portion of the buffer must contain correct address of attack code in the RET position. The value in the RET position must point to the beginning of attack assembly code in the buffer (Otherwise application will crash with segmentation violation). Attacker must correctly guess in which stack position his buffer will be when the function is called. NOP sled (sequence of "no-operation"

instructions) at the beginning of attack code. RET position must not be precise, enough to the the NOP sled.

Attack #1: Return address:

You can change the return address to point to the attack code. After the function returns, control is transferred to the attack code. Otherwise you can use existing instructions in the code segment such as system(), exec(), etc. as the attack code.

This is caused by no-bounds-checking. Many C-library-Functions (like strcpy) don't check the input size.

C uses function pointers for callbacks: if pointer to F is stored in memory location P, then another function G can call F as (*P)(…).

Attack #2: Pointer Variables

Change a function pointer to point to attack code. Any memory, on or off the stack, can be modified by a statement that stores a value into the compromised pointer.

An example is a off-by-one-overflow: With a 1-Byte-Overflow you cannot change RET but you can change saved pointer to previous stack frame.

Attack #3: Frame-Pointer:

Change the caller's saved frame pointer to point to attacker-controlled memory. Caller's return address will be read from this memory

Memory Defence

Memory Exploits

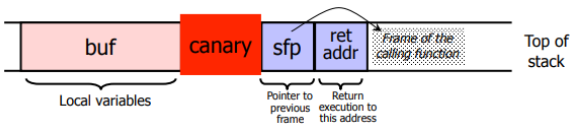
The buffer is a data storage area inside computer memory (stack or heap). It is intended to hold a pre-defined amount of data. The simplest exploit is it to supply executable code (shellcode) as "data", trick victim's machine into executing it. The code will self-propagate or give attacker control over machine.

Overwriting adjacent memory locations can modify other variables or modiy the control flow data such as return addresses and pointers to previous stack frames.

In the worst case, the attacker will execute arbitrary code with the privileges of the attacked process.

Causes for Memory Exploits can be "reading data from stdin or the network", "Copying/merging data", "Bugs in boundary check (off-by-one)" or "Appending strings".

Stack Canaries



We can have embed "canaries" (aka stack cookies) in stack frames and verify their integrity prior to function return. Any overflow of local variables will damage the canary. This makes it hard to exploit a buffer overflow.

When we use a **terminator canary** (e.g. "\0", newline, linefeed, EOF) , string-functions like strcpy won't copy beyond "\0". A disadvantage is that the canary is known and therefore can be overwritten the attacker with the known value.

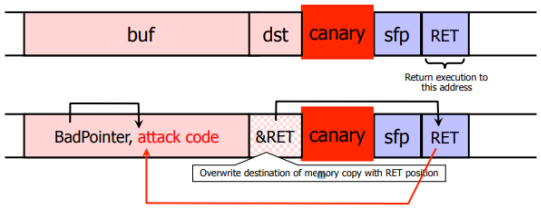
A **random canary** is random string chosen at program start and is stored in a global variable padded by unmapped pages. A attacker can't guess what the value of canary will be and any try to exploit bugs to read off RAM will cause a segmentation fault. A sisadvantage is that the attacker still can learn the canary if he knows where to read it from.

A **random XOR canary** is random string chosen at program start. It is XOR scrambled using all or part of the control data to protect (return pointer, etc.). It can be used to detect attacks in which attacker is able to modify return address without overwriting the canary. A disadvantage is that it has the same vulnerability as random canaries, but the attacker also needs to know the control data and the scrambling algorithm.

Cons for the stack-canaries are:

- It requires code recompilation.
- Checking the canary integrity prior to every function return causes a performance penalty
- The StackGuard can be defeated: single memory copy where the attacker controls both the source and the destination are sufficient.

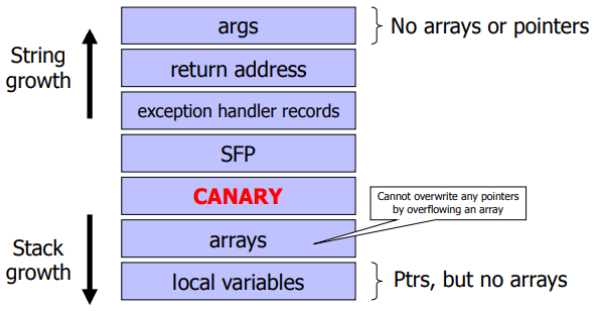
To defeat the StackGuard we suppose a programm contains *dest=buf[0] where the attacker controls both dst and buf. Example: dst is a local pointer variable. Therefore we overwrite pointer to make it point to RET. The strcpy()-function will write into RET without touching canary.



Stack Canaries have some limitations. They do not prevent heap-based buffer overflows. They only protect against contiguous buffer overflows and won't detect if an exploit writes to arbitrary address directly. They give us no protection if attack happens before function returns. The canary won't detect if exploit overwrites an argument function pointer that gets called before function returns or an exception handler that gets invoked before function returns. The canary alone offers no protection for local pointers, because they are allocated before the canary. This is bad in particular for function pointers, but not only. But they are still good as a first barrier of defence.

ProPolice Stack-Smashing Protection (SSP)

In the ProPolice Stack-Smashing Protection (SSP) we rearrange the stack layout.



But with the rearrangement there is still data which can be overwritten. One of them are other string buffers in the vulnerable function. We can

also overwrite any data stored on the stack like exception handling records or pointers to virtual method tables.

W⊕X / DEP

We mark all writeable memory locations as non-executable (W⊕X means Write XOR execute). An example is Microsoft's DEP (Data Execution Prevention). This blocks most (not all) code injection exploits. It has hardware-support and is widely deployed.

But it does have some Issues. Some applications, like JavaScript, Flash or Lisp, require an executable stack. The JVM makes all its memory RWX (readable, writable, executable). Therefore, someone can spray attack code over memory containing Java objects, pass control to them. An attack can start by "returning" into a memory mapping routine and make the page containing attack code writeable.

With the usage of W⊕X we can still corrupt stack or function pointers or critical data on the heap. As long as "saved EIP" (Extended Instruction Pointer) points into existing code, W⊕X protection will not block control transfer.

Return-to-Libc Exploits

We can overwrite a saved EIP with the address of any library routine, arrange memory to look like arguments. This does not look like a huge threat at a first glance because an attacker cannot execute arbitrary code (especially if system() is not available). An overwritten saved EIP need not point to the beginning of a library routine! If any existing instruction in the code image is fine, the system will execute the sequence starting from this instruction.

We can call any function in libc (sh, system, etc.). The trick is that programs that use functions from a shared library, like printf from libc, will link the entire library into their address space at run time. We can use this technique to run arbitrary code rather than just calling functions we have available to us. We do this by returning to gadgets, which are short sequences of instructions ending in a ret.

Address Space Layout Randomization (ASLR)

The problem is the lack of diversity. Classic memory exploits need to know the (virtual) address to hijack control, so it needs the address of attack code in the buffer and the address of a standard kernel library routine. But some addresses are used on many machines.

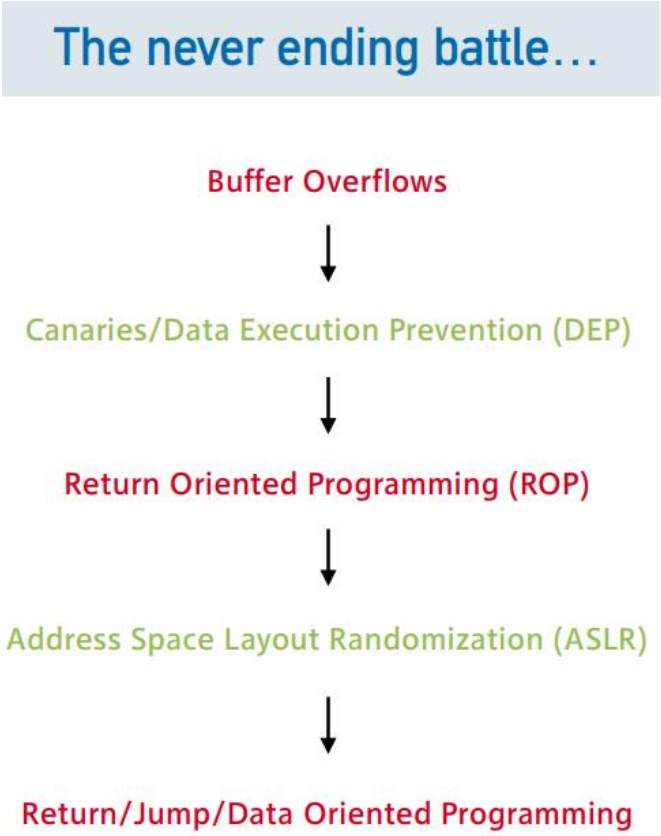
Therefore, we need to introduce artificial diversity: We need to make stack addresses, addresses of library routines, etc. unpredictable and different from machine to machine.

We randomly choose base address of stack, heap, code segment, location of Global Offset Table. The randomization can be done at compile- or link-time or by rewriting existing binaries. We randomly pad stack frames and malloc'ed areas. Other randomization methods are to randomize system call IDs or to even the instruction set. They are implemented in today's operating systems. The effectiveness depends on the amount of randomness. It is crucial to keep pointer information secret: if the attacker can read, we are screwed!

In case of the **Base-address-Randomization** only the base address is randomized. The layouts of stack and library table remain the same. The relative distances between memory objects are not changed by

base address randomization. To attack, it's enough to guess the base shift. A 16-bit value can be guessed by brute force. If address is wrong, target will simply crash.

ASLR makes ROP harder, but not impossible. One can guess the position of the target function (or gadget) by computing the offset from a function whose address is known.



Browser Security

Web-applications work in 4 Steps: The user requests a webpage with dynamic content, the web application queries a database, the database data is used by application to generate page content and the received data is rendered by the client's browser.

Each browser window or frame loads the content, renders the pages (It processes the HTML and scripts to display the page. This may involve images, subframes, etc.) and responds to events. Events can be user actions (onClick, onMouseOver), rendering (onload, onUnload) or timing (setTimeout(), clearTimeout()).

JavaScript in web-pages

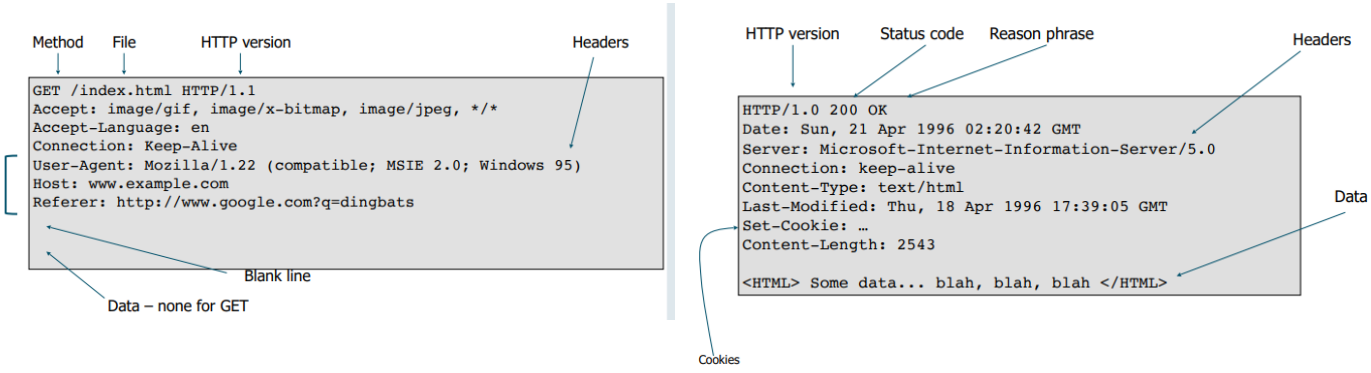
JavaScript can be embedded in HTML page as <script> element. The JavaScript can be written directly inside <script> element (<script> alert("Hello World!") </script>) or can be linked to a fie as src attribute of the <script> element (<script type="text/JavaScript" src="functions.js" ></script>). It can also be inserted as event-handler attribute (<a href=http://www.yahoo.com onmouseover="alert('hi');">) or as Pseudo-URL referenced by a link (Click me).

JavaScript can interact with the HTML page and the browser through the DOM and the BOM. The **Document Object Model** (DOM) is a Object-oriented representation of the hierarchical HTML structure. It has properties like (document.alinkColor, document.URL, document.forms[ ], document.links[ ], …) and Methods like (document.write(document.referrer)) which can change the content of the page. The **Browser Object Model (BOM)** contains Window, Frames[], History, Location, Navigator (type and version of browser).

HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is simple, widely used, stateless and unencrypted and uses Port 80. The Hypertext Transfer Protocol Secure (HTTPS) is the successor of HTTP and is encrypted by SSL/TLS and uses Port 443.

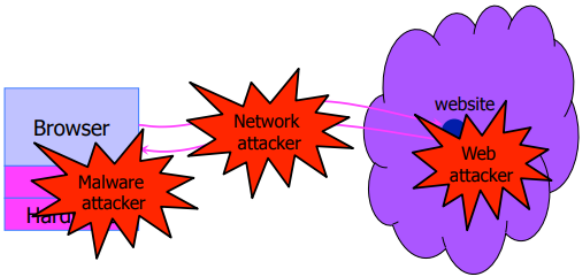
The Uniform Resource Locator (URL) is the global identifier of network-retrievable documents. Special characters are encoded as hex (e.g. %0A = newline, %20 or + = space, %2B = +).



Web application security problems

Web security is problematic because of the delusive simplicity for creating web apps, the lack of security awareness, Time- & resource limits during the development and a rapid increase in code complexity.
Many company's security focus shifts towards web. The security perimeter moves from network to application layer. Web applications intentionally expose functionality to the Internet while being connected to internal servers (e.g. database servers). Controlling network traffic via port blocking is no longer effective on application layer (port 80/443 problem).

A **Web attacker** controls a malicious website (attacker.com) and can even obtain an SSL/TLS certificate for his site. A user visits attacker.com (via e.g. Phishing email, enticing content, search results, placed by an ad network, blind luck …). A attacker has no other access to user machine!



A **Network attacker** can be a passive newtwor attaker where he is a wireless eavesdropper or a actice one, wehere he uses evil Wi-Fi router, DNS poisoning,…

A **Malware attacker** sends malicious code which executes directly on victim's computer. To infect a victim's computer, he can exploit software bugs (e.g., buffer overflow) or convince user to install malicious content himself.

Common web attacks

Common web attacks are on the client-side Cross-site Scripting (XSS), Cross-site Request Forgery (CSRF) and on the server-side SQL injection, Path traversal and Remote code injection.

Same Origin Policy

The foal of the Same Origin Policy is it, to safely execute JavaScript code provided by a remote website by enforcing isolation from resources provided by other websites. We want to have no direct file access, limited access to OS, network, browser data, content that came from other websites.

"A service can only read properties of documents and windows from the same protocol, domain, and port." – Definition of SOP

The SOP is often misunderstood. It often simply stated as "same origin policy: This usually just refers to "can script from origin A access content from origin B"?
Full policy of current browsers is complex. It evolved Evolved via "penetrate-and-patch" and has different features evolved in slightly different policies.

The common scripting and cookie policies are that a script has access to DOM considers protocol, domain and port, cookie reading reading considers protocol, domain, path considers protocol, domain, path and cookie writing considers domain.

Historically there were inconsistencies across different browsers.

Same Origin Policy (SOP) for DOM:

Origin A can access origin B's DOM if A and B have same **(protocol, domain, port)**

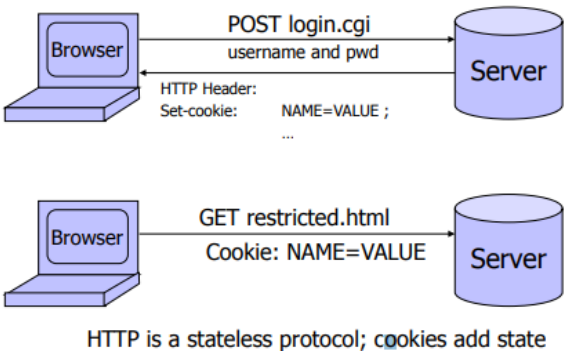
Same Origin Policy (SOP) for cookies:

Generally, based on **([protocol], domain, path)**

optional

Cookies

A cookie is a file created by a website to store information in the browser. When it's used for authentication, the cookie proves that the client previously authenticated correctly. Furthermore it can be used for personalization, where it helps the website recognize the user from a previous visit and can also be used for Tracking, where it follows the user from site to site and learns his/her browsing behavior, preferences, … .



A cookie is set by the http-response and is deleted by by setting "expires" to date in past. The default scope is the domain and path of the setting URL.

The SOP for writing cookies is for the domain: any domain suffix of URL-hostname, except top-level domain (TLD) and for the path is anything.

The SPO for reading cookies is that the browse sends all cookies in URL scope where the cookie-domain is domain-suffix of URL-domain, the cookie-path is prefix of URL-path and the protocol is =HTTPS if cookie is "secure".

The SOP for JavaScript in the browser is that it has the dame domain scoping rules as for sending cookies to the server. Javascript can set and delete cookies via DOM.

A HttpOnlyCookie is a Cookie which is sent over HTTP(s), but is not accessible to scripts. It cannot be read vie document.cookie and also blocks access from XMLHttpRequest headers. This helps to prevent cookie theft via XSS but does not stop most other risks of XSS bugs.

Frames

A window may contain frames from different sources. A **frame** is a rigid division as part of frameset and an **iframe** is a floating inline frame.

We use frames to get modularity, isolation and robustness. Because of the modulation we can bring together content from multiple sources and aggregate them on the client. We can delegate screen area to content from another source. Our browser provides isolation based on frames because different frames can represent different principals and they cannot script each other. A Frame can only draw on its own rectangle. Robustness assures, that the parent may work even if frame is broken.

Browser security policies for frames

Each frame of a page has an origin and each frame can access objects from its own origin (Read/write DOM, cookies and local-storage). Frames cannot access objects associated with other origins.

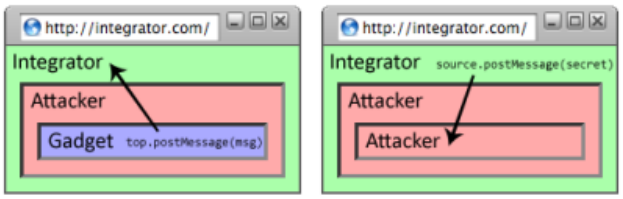
Today's browsers all have descendant policy.

Cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) says that the Access-Control-Allow-Origin is a list of domains. The typical usage is Access-Control-Allow-Origin: *.

Message eavesdropping

We assume a descendant frame navigation policy. The attacker embeds the integrator in its own web page and when the integrator call postMessage on gadget, attacker replaces it with its own gadget. So the attacker gets the secret. The source is a reference to the frame which sent the message. The attacker replaces the child frame with his own and gets the secret message.



SQL Injection

The serve- side of a web application runs on a web-server (application-server). It takes inputs from remote users via web-server and interacts with back-end database servers. It prepares and outputs results for users in dynamically generated HTML-pages with content from many different sources.

SQL (short for Structured Query Language) is a widely used database query language with a syntax (mostly) independent of a vendor.

An SQL-injection is a input validation vulnerability, where an untrusted user input in an SQL query is sent to backend database without sanitizing the data. This is bad, because data can be misinterpreted as a command and it can alter the intended effect of the command or the query.

Example for an SQL-Injection:

```
$user = "admin";
$pass = "' OR '1'='1";

$sql = "SELECT * FROM UserTable
                WHERE   User = 'admin' AND
                      Passwd = '' OR '1'='1'";
```

```
$sql = "SELECT * FROM UserTable
                WHERE (User = 'admin' AND Passwd = '')
                       OR
                      '1'='1'";
```

→ '1' = '1' -> evaluetes to TRUE

With the usage of **'UNOIN SELECT'** we can get information from another table and the results of the two tables get combined. The empty table from the first query is displayed together with the entire contents of the second query. Table after the UNION must have the same amount of columns as the original SELECT.

Second-order SQL injection

We can use data, which is stored in the database, to conduct an SQL injection later. For example user manages to set username to admin' --. This vulnerability could occur if input validation and escaping are applied inconsistently (Some Web applications only validate inputs coming from the Web server but not inputs coming from the back-end DB). The solution is to treat all parameters as dangerous.

Solution: Prepared Statements

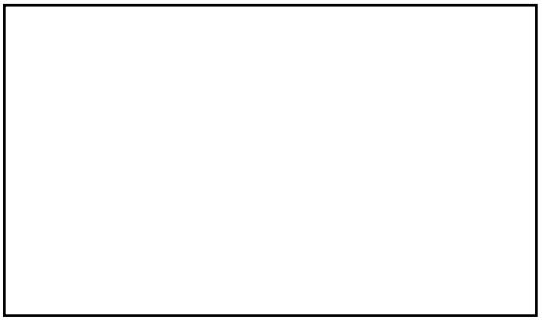
An SQL-onjection is a problem because of insufficient validation of untrusted data. You can never trust a user input and you need to validate input data according to specified requirements (e.g. only dots and numbers in birthdays). Consequences can be an information leakage, a data manipulation or sabotage.

In the most injection attacks, data are interpreted as code. This this changes the semantics of a query or command generated by the application. You need to bind variables so that placeholders are guaranteed to be data and not code. Prepared Statements allow creation of static queries with bind variables and preserve the structure of intended queries.

Cross Site Request Forgery (CSRF)

A user logs into bank.com and forgets to sign off, so the session cookie remains in browser state. Then the user visits a malicious website containing the malicious code and the browser sends cookie, payment request fulfilled. The problem is, that the cookie authentication is not sufficient when side effects (i.e., attacker forging requests) can happen.

```
<form  name=BillPayForm
 action=http://bank.com/BillPay.php>
 <input  name=recipient  value=badguy> …
 <script> document.BillPayForm.submit(); </script>
```



It happens in 4 steps:

1. User established session with server
2. User visits attack-server
3. User receives malicious website
4. Website sends forged request to victim server

Traditional CSRF Defenses

Traditional CSRF defences are

- a secret validation token (`<input type=hidden'value=23a3af01b>`),
- referer validation (`http://www.facebook.com/home.php`)
- or a custom HTTP header (`X-Requested-By: XMLHttpRequest`)

Secret validation tokens

A synchronizer token pattern is a secret, random number embedded by the web application in all HTML forms and verified on the server side. It is hard to keep it secret against malicious scripts reading the webpage.

A cookie-to-header token (Javascript) is a token, which is set on login with a random number. The javascript reads the value and copies it into the custom HTTP header. The server then checks the token.

Referer validation

A referrer validation can be lenient, where the header is optional or strict, where the header is required. It cannot always be strict, because the referrer header ca be surpressed with the following examples:

- Stripped by the organization's network filter
- Stripped by the browser for HTTPS → HTTP transitions
- Stripped by the local machine
- User preference in browser
- Buggy browser

Web applications can't afford to block these users but referer are rarely suppressed over HTTPS.

Custom header

A custom header is a XMLHttpRequest for same-origin requests and therefore cannot come from the attacker. The browser prevents sites from sending custom HTTP headers to other sites, but can send to themselves. A website can use setRequestHeader within origin. Google Web Toolkit recommends that developers defend against CSRF by attaching a X-CSRF-Cookie (the value does not matter, the header suffices). A similar mechanism is implemented in AJAX toolkits (XRequested-With header).

When using custom headers there are no secrets required.

SameSite Cookie

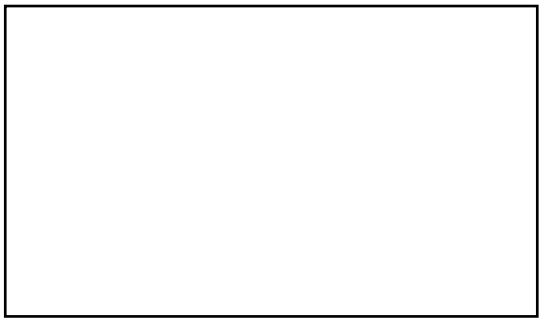
A SameSite cooke is a cookie with an attribute, which prevents the cookie from being sent along with cross-domain requests. The strict-flag says, that the site must be as you expect it, but this could introduce compatibility issues. The lax flag says that the cookie is sent even in case of cross-domain requests, but then there must be a change in top-level navigation and therefore the user realizes it.

Cross Site Scripting (XSS)

An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application. Methods of XSS are reflective XSS, Stored XSS and other variants, such as DOM-based attacks.

Reflected XSS (type 1)

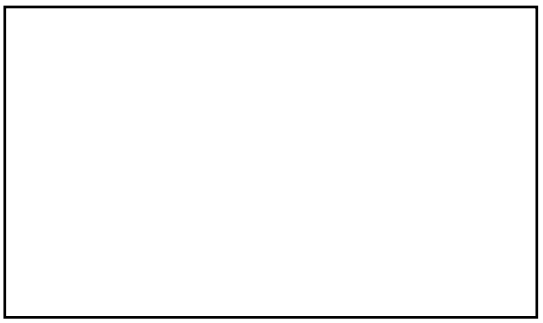
By reflective XSS the attack script is reflected back to the user as part of a page from the victim site. The user is tricked into visiting an honest website, but a bug in website code causes it to echo to the user's browser an attack script. The origin of this script is now the website itself. The script can manipulate website contents (DOM) to show bogus information, request sensitive data, control form fields on this page and linked pages and can cause user's browser to attack other websites. This violates the "spirit" of the same origin policy, but not the letter.



The malicious scripts lurk in the user-created content like social sites, blogs, forums and wikis. When a visitor loads the page, the website displays the content and the visitor's browser executes the script. Many sites try to filter out scripts from user content, but it's difficult.

Stored XSS (type 2)

By stored XSS the attacker stores the malicious code in a resource managed by the web application, such as a database. A popular method is to store the information in images. Suppose oic.jpg on web server contains HTML. A request for results in a positive HTTP response like seen. E.g. internet explorer will render this as a HTML, despite the content-type. This can happen on photo-sharing sites, where a attacker uploads an image with a script.



```
HTTP/1.1  200 OK
...
Content-Type:  image/jpeg
<html>  fooled ya  </html>
```

DOM-Based XSS

By a DOM-based XSS attack the attacker's code is never sent to the server (What Is written after the # will not be sent to the server). It is instead written by the browser directly in the DOM and rendered there, so the server-side detection techniques do not work. The problem is that an input from the user (source) goes to an execution point (sink).

Preventing cross-site scripting

To protect yourself, you need to ensure that your app validates all headers, cookies, query strings, form fields, and hidden fields against a rigorous specification of what should be allowed. Do not attempt to identify active content and remove, filter or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content. You should define

your policy ‘positive’ this means what is allowed and not ‘negative’, this means what is forbidden. This is due to the fact, that negative policies are often incomplete.

To prevent cross-site scripting, and user input and client-side data must be preprocessed before it is used inside the HTML. You need to remove or encode (X)HTML special characters (e.g. ‘ becomes &039; or & becomes &). The prevention of injection of scripts into HTML is hard. Blocking < and > is not enough, because of scripts in event handlers, stylesheets and encoded inputs. You should beware of filter evasion tricks (XSS Cheat Sheet).

Going Beyond Scripting

In a post-XSS-world we would have many browser mechanisms to stop script injection like Content Security Policy, Add-ons like NoScript, Built-in XSS filters in IE and Chrome, Client-side APIs like toStaticHTML() and many server-side defences. But attackers can do damage by injecting non-script HTML mark-up elements, … .

One of the goals of XSS attacks is to steal user data, such as cookies. But the attacker could steal the secret token used to counter CSRF attacks and the attacker might be able to get it in the url.

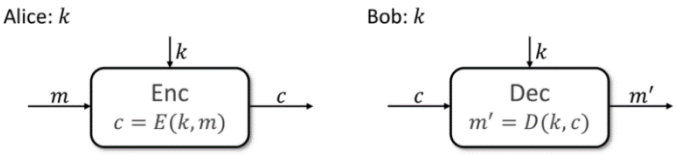
But a attacker could still reroute existing forms. The <form< tag can’t be nested, and the top-level occurrence of this markup always takes precedence over subsequent appearances. The attacker can change the URL to which any form will be submitted, simply by injecting an additional form definition in the preceding portion of the document. This is very nice with autofilled forms.

Cryptography

Symmetric Cryptography

Streamciphers

In a symmetric cryptography both communication-partners (Alice and Bob) have the same key k.



Histroricly ciphers are already used in the bible, where they rotate the letters (e.g. ROT 13). These are very unsecure with 26 possible keys and with guessing cracked very easily.

The Vigenere Cipher is a cipher from the 16th century and it's key is a randomly chosen string of certin length n. It has 26^n possibilities which is not very secure.

Ciphers

Definition (Symmetric Encryption Scheme):

A symmetric encryption scheme with key space K, message space M, and ciphertext space C is a triple of algorithms (Gen,Enc,Dec):

- The randomized key generation algorithm Gen takes no input and returns a key $k \in K$
- The (often randomized) encryption algorithm Enc takes a key $k \in K$ and a message $m \in M$ and returns a ciphertext $c \in C$.
- The deterministic decryption algorithm Dec takes a key $k \in K$ and a ciphertext $c \in C$ and returns a plaintext $m \in M$ or a distinguished error symbol.

Correctness:

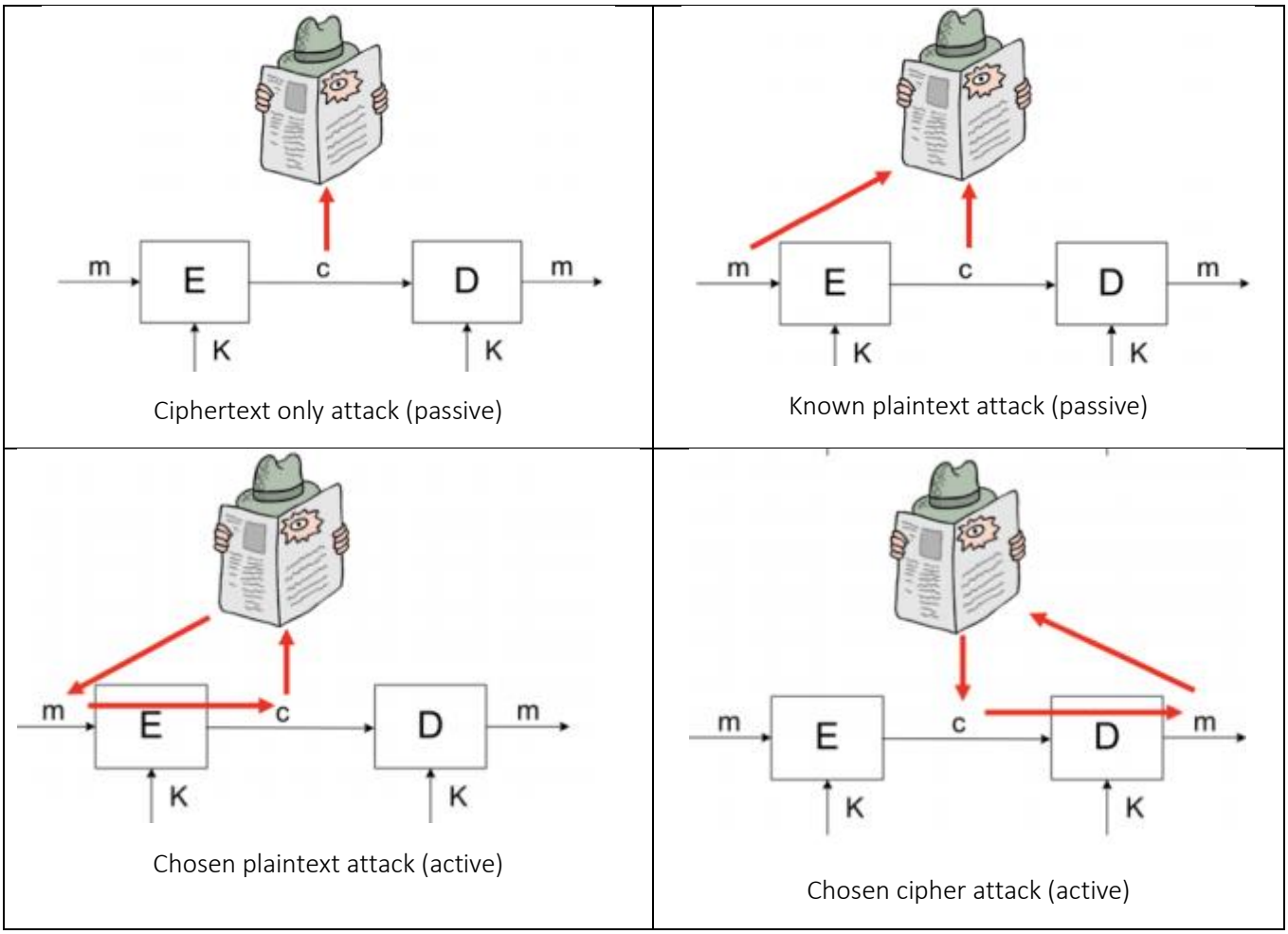
The above algorithms have to satisfy the following property: for any key $k \in K$, any message $m \in M$, and any $c \leftarrow Enc(k, m)$, we have that $Dec(k, c)=m$.

Security of Ciphers

There are more types of adversary success. The **total break** means, that an attacker finds the key, an **usiversal break means**, that the attacker finds an equivalent method to being able to decrypt with key. Under **an partial break** we understand when an attacker successfully decrypts only selected ciphertexts, but those completely and under **partial information** we understand that an attacker is able to successfully lerarn partial information about single plaintexts (individual bits, checksum,…).

The security of ciphers is part of the information theoretic security. The basic idea is to define, that a ciphertext reveals no information about its plaintext.

The first three are unacceptable, the fourth seems strong, but is on the save side.



Ciphertext only attack (passive)

Known plaintext attack (passive)

Chosen plaintext attack (active)

Chosen cipher attack (active)

The security of ciphers is a part of information theoretic security. Ithe dasic idea is to define, that a ciphertext reveals no information about its plaintext. The definition of information theorectic security states:

"An encryption scheme is information theoretically secure,
if "no information" about its plaintext is revealed."

The intuition of this is, that the adversary learns nothing about the plaintext after seeing the ciphertext that he didn't know already. The convention is, that M, C, K are random variables denoting the value of the message, ciphertext and key respectively (for a given probability distribution).

Perfect Secrecy: An encryption scheme (Gen, Enc, Dec) over a message space is perfectly secret if for every probability distribution over M, every message $m \in M$ and every ciphertext $c \in C$:

$$Pr[M = m \mid C = c] = Pr[M = m]$$

Lemma: Perfect Indistinguishability

An encryption scheme (Gen,Enc,Dec) over a message space $\mathcal{M}$ is perfectly secret if and only if every probability distribution over $\mathcal{M}$, every message $m_0, m_1 \in \mathcal{M}$ and every ciphertext $c \in \mathcal{C}$

$$Pr[C=c \mid M=m_0] = Pr[C=c \mid M=m_1]$$

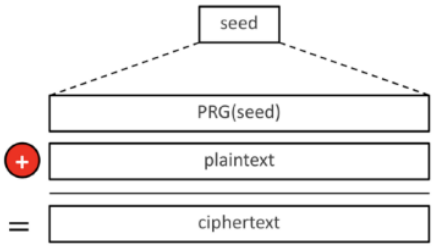
The intuition fo this lemma is, that given a ciphertext c, no adversary can tell if the ciphertext contains m_0 or m_1 (for any m_0 and m_1). When the ciphertext is perfectly secure, no ciphertext-only attack is possible, but other attacks might be.

One-time Pad (OTP)

One-time pad is the first proven-secure cipher. It has a secret key $k \in \{0,1\}^n$ whtih a randem bit string as long as the message. The functions are: $Enc(k,m) = k \oplus m$ and $Dec(k,c) = k \oplus c$. It has very fast decryption and encryption, but the problem is, that the key is as long as the message and we have to generate lots of randomness. (Proof on slides 30-31).

Stream Ciphers

The problem with OTP ciphers is, that the key is as long as the plaintext. Therefore, we replace the random with a pseudorandom and the secret key is now a seed. The pseudorandom generator (PRG) expands a small seed of random bits into a large amount of somewhat random bits.



Perfect secrecy of sream cipers is impossible, because $|K| < |M|$. To increase the secrecy of the steam ciphers, wen need to assure, that the PRGs are unpredictable. We don't only need the hiding of the seed, but also not allowing to look forward which randomness will be created through the seed. The unpredictability is very important, because many messages have known prefixes (like fixed header of an e-mail), and therefore we can predict the cipher and leak the plaintext. But lot's of PRGs don't satisfy this.

Getting true randomness is hard. Weak options are (analog) to throw a coin or (digital) it derives from load/system parameters. Stronger is it, to use different physical processes that are expected to be random like thermal noise, air pollution, … . These randomness from all inputs is XORed and hashed to remove the bias. This works well, but too slow for some purposes.

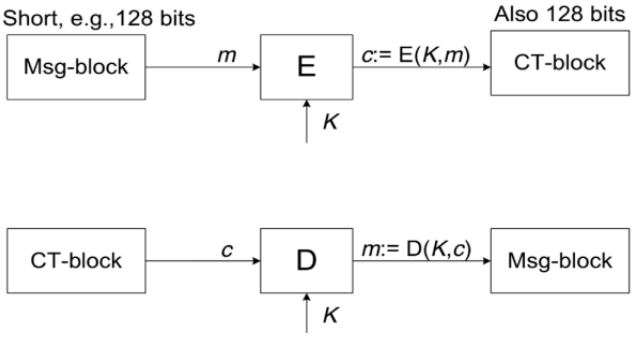
In the encryption it is important, that one key is only used once, because otherwise the vulnerability grows with every usage.

How to use a Stream Cipher: We assume that we already have a strong (but slow) cipher for encrypting small blocks and we have a PRG. We pick a random seed and transfer it in a secure way. Then we use the PRG to produce a pseudorandom stream and we use this stream to encrypt messages.

$$E(k,m) = \underbrace{E^*(k,seed)}_{\text{Strong cipher}} \;||\; \underbrace{m \oplus PRG(seed)}_{\text{Fast cipher}}$$

Blockciphers

A block cipher is a deterministic algorithm operating on a fixed-length group of bits called block (as opposed to a stream if bits bit by bit).



Electronic Codebook (ECB)

Electronic codeblock has a parallelizable encryption, a parallelizable decryption and a random read access. Every block of ECB will be encrypted by the same cipher. A problem is, that it can reveal patterns, but al least it is self-synchronising, because if one bit fails, the complete block fails.

Cipherblock Chaining (CBC)

In Cipherblock chaining the initial value is randomly chosen and the output as well. It is very often used, but has a main problem, that it es sequential, and therefore the encryption is not parallelizable (the decryption is).It is self-synchronising after two blocks, if the block-length is ok.

Cipher Feedback (CFB)

Cipher feedback is similar to stream ciphers. The encryption cannot be parallelized and it is also self-synchronizing after two blocks, if the block-length is ok.

Data Encryption Standard (DES)

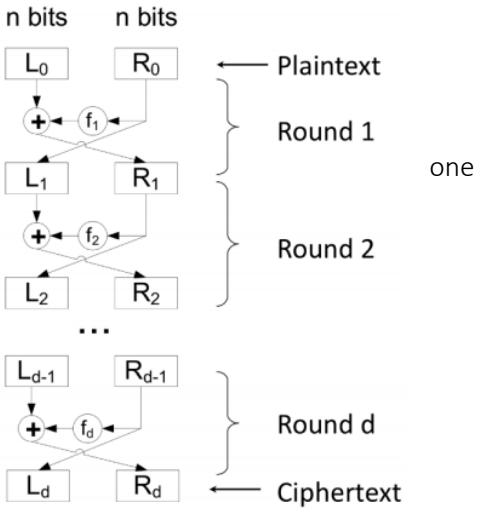
The idea of DES is to encrypt in many rounds.

$f_1, \dots, f_d : \{0,1\}^n \to \{0,1\}^n$ (for DES: n = 32 , d=16)

It consists out of d Feistel networks which is a one-to-one map.

For example DES has an initial permutation, 16 rounds of Feistel networks and the inverse of the initial permutation to get the ciphertext.

(Attacks DES are on slides 72-85).



Advanced Encryption Standard (AED)

Explained on slides 88 – 93. Must be worked through by hand.

Macs and Hashes

The goal of meesage integrity is that Alice generates tag t for message m and bob verifies the tag. The goal is it, that the attacker cannot change the message i.e. the attacker cannot generate any valid pair (m, t).

Message Authentication Codes (Macs)

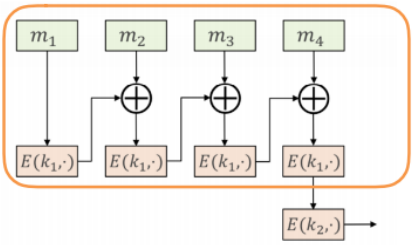
Definition:

A message authentication code with message space M, key space K, and a tag space T is a triple of algorithms (Gen, Sig, Ver):

- The randomized key generation algorithm Gen takes no input and returns a key $k \in K$.
- The (often randomized) signing algorithm Sig takes a key $k \in K$ and a message $m \in M$ and returns a tag $t \in C$.
- The deterministic verification algorithm Ver takes a key $k \in K$ and a message $m \in M$ and returns a bit $b \in \{0,1\}$.

Let $E: K \times X \to Y$ be an encryption function. We define CBC-MAC I^E as follows:

Raw-CBC



Where I^E is a function $K^2 \times X^L \to Y$.

The raw CBC-MAC is an incecure MAC, so we need another, last encryption. The chosen message attack is that adv picks random one-block-length message $m \in X_n$and requests tag for message m and gets $t \coloneqq Enc(k, m)$. Adv outputs t as MAC forgery on two-block-length message $(m \,||\, t \oplus m)$.

Hash-functions

Let $H: M \to T$ be a non-keyed hash function. A collision for H is a tuple (m_1, m_2) with $H(m_1) = H(m_2) \wedge m_1 \neq m_2$. A collision resistant hash function (CRHF) is a hash function H, where there is no efficient algorithm known that finds collisions for H in suitable time.

More Details to Hash functions on sildes 103-109

Asymmetric Cryptography

Asymmetric cryptography is slower than symmetric, but it is based on security proofs with well-defined assumptions instead of heuristics. In asymmetric cryptography you have one key for every user instead of one key for every pair of users. Therefore every user is responsible for his/her own secret key instead of two parties being responsible for one key.

Public-key encryption

A public key encryption scheme with key space K, message space M, and ciphertext space C is a triple of algorithms (Gen, Enc, Dec):

- The randomized key generation algorithm Gen takes no input and returns a key-pair $pk, sk \in K$.

- The (often randomized) encryption algorithm Enc takes a public key pk and a message $m \in M$ and returns a ciphertext $c \in C$.
- The deterministic decryption algorithm Dec takes a key $sk \in K$ and a ciphertext $c \in C$ and retruns a plaintext $m \in M$ or a distinguished error symbol.

To the **correctness**: The above algorithms have to satisfy the following property: for any eky-pair $(pk, sk) \in K$, any message $m \in M$, and any $c \leftarrow Enc(pk, m)$, we have that $Dec(sk, c) = m$.

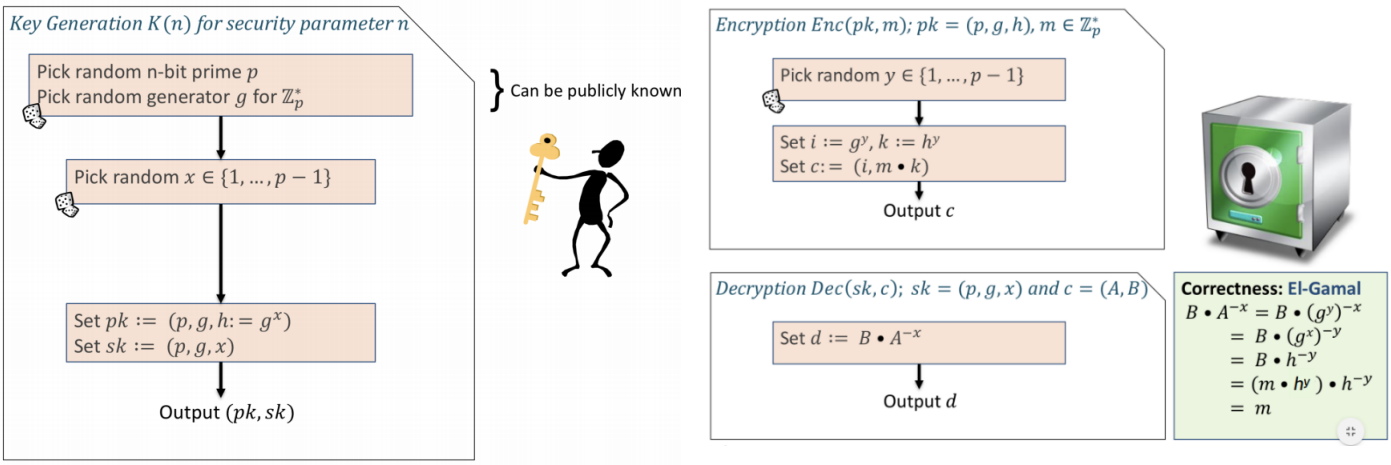
Number Theory Basics for the El-Gamal Encryption Scheme

Let N be a positive integer and p be a prime number in $\mathbb{Z}_n = \{0, 1, .., N-1\}$. There we can do addition and multiplication modulo N. With the usage of the modulo arithmetic we can solve the greatest common devisor by using the Extended Euclidian Algorithm. For Example see slide 10. The Modular Inversion of an element x in the modulo arithmetic is y, when y fulfils $x \cdot y = 1$. Then y is notated as x^{-1}.

$\mathbb{Z}_p$ is a cyclic group, that is $\exists g \in \mathbb{Z}_p^*$ such that $\{1, g, g^2, \dots\} = \mathbb{Z}_p^*$. An example for p=7 is g=3. But not every element is a generator.

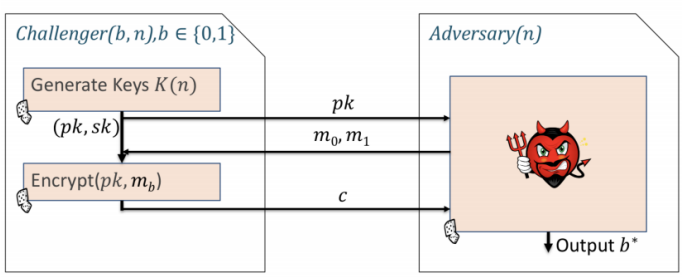
Order of g: The set $\{1, g, g^2, \dots\}$ is called the set generated by g, denoted <g>. The order of $g \in \mathbb{Z}_p^*$ is the size of <g>, denoted by $ord_p(g) = |< g >|$. It's the smallest $a > 0$, such that $g^a = 1$ in $\mathbb{Z}_p^*$.

You can create intractable problems with primes using Dlog. Fix a prime $p > 2$ and g in $\mathbb{Z}_p^*$ of order q. Now consider the function $x \to g^x$ in $\mathbb{Z}_p^*$. Now consider the inverse function $Dlog_g(g^x) = x$ where $x \in \{0, .., q-2\}$.



CPA-Security

Let $PE = (K, E, D)$ be a public-key encryption scheme and A an adversary. Define $Exp_{PE,A}^{CPA}(b)$ as seen on the picture,



Definition:

A sequence of pubic-key encryption schemes PE has indistinguishable ciphertexts under chosen-plaintext attack (CPA) if for all polynomially bounded (in the security parameter n) adversaries A

$$Adv_{PE,A}^{CPA} = \left|\Pr\left[Exp_{PE,A}^{CPA}(0) = 1\right] - \Pr\left[Exp_{PE,A}^{CPA}(1) = 1\right]\right|$$

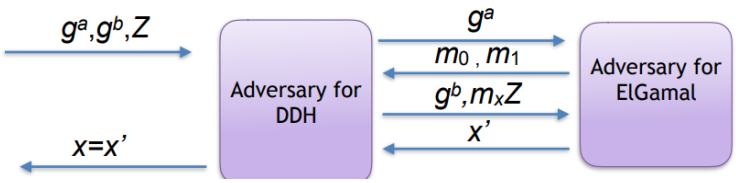
is negligible.

When we look at the CPA-Security of ElGemal we see that the following holds: (**IND-CPA of ElGamal**):

ElGamal has indistinguishable ciphertexts under CPA if the Decisional Diffie-Hellman assumption holds in G

Decisional Diffie-Hellman Assumption (DDH): Given a group G with $\sim 2n$ elements and a random $g \in G$, no polynomially bounded adversary (in n) can distinguish (g^x, g^y, g^z) and (g^x, g^y, g^z) for x, y, z random in $\{1, \dots, |G|\}$.

When we use ElGemal together with DDH (ElGemal => DDH) we can suppose some algorithm A breaks ElGemal. When given any public key, A produces plaintext m_0 and m_1 whose encryptions it can distinguish with advantage Adv.



Now we will use A to break DDH. We decide, given (g^a, g^b, Z) whether $Z = g^{ab} \bmod p$ or not. Give $y = g^a \bmod p$ to A as the public key. This produces m_0 and m_1. Toss a coin for bit x and give A the ciphertext $(g^b, m_x\, Z) \bmod p$. This is a valid ElGemal encryption of m_x iff $Z = g^{ab} \bmod p$. Then A receives $(g^b, m_x Z) \bmod p$. This is as well a valid ElGemal encryption of m_x iff $Z = g^{ab} \bmod p$. A outputs his guess of bit x'. If A guessed correctly ($x = x'$), we say that $Z = g^{ab} \bmod p$ otherwise we say Z is random. The advantage of breaking ElGemal carries over to DDH if $Z = g^{ab} \bmod p$, otherwise A cannot do any better than guessing with probability $1/2$. The second component of the ciphertext is a random group element and the ciphertext is neither an encryption of m_0 or if m_1. Hence the advantage of deciding DDH is half of the advantage of breaking ElGemal which is still not negligible.

Number Theory Basics for the RSA Encryption Scheme

Assumption Factorization: For two prime numbers p, q it is easy to compute $N = p \cdot q$. But given N it is hard to compute p and q.

Euler's ϕ function: For $N \in \mathbb{N}$, we define $\phi(n) := |\mathbb{Z}_N^*|$. Therefore $\phi(p) = p - 1$ and $\phi(N) = (p-1)(q-1)$.

Euler's theorem: For $x \in \mathbb{Z}_N^*$, it holds that $x^{\phi(N)} = 1 \; mod \; N$.

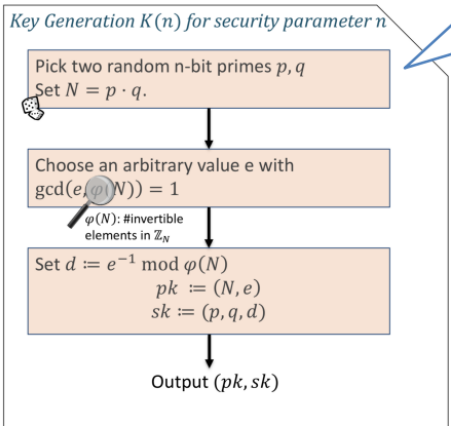
E'th root: Let p be a prime and $c \in \mathbb{Z}_p$. $x \in \mathbb{Z}_p$ such that $x^e = c$ in $\mathbb{Z}_p$ is called the e'th root of c.

More to the principals and the e'th root on slides 31 to 40.

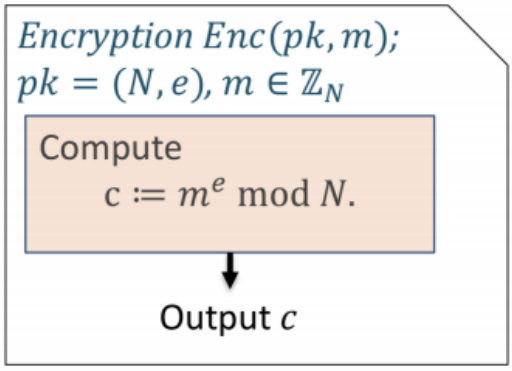
RSA Trapdoor Permutation

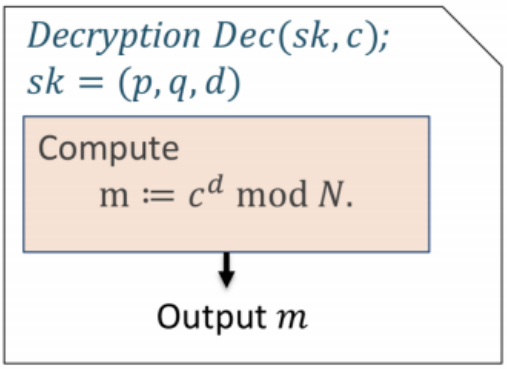
When we pick two random n-bit primes p and q they should be similar in magnitude but differ in length by a few digits to make factoring harder. Tis step cannot be publicly known, n is typically 1024 or 2048.

In the second step is no need for randomness, but you must choose carefully.



These two pictures show the naïve use of the RSA scheme.





A problem is, that this is not even secure against passive attacks. This is deterministic: $m = m' \Rightarrow Enc(pk, m) = Enc(pk, m')$. Therefore, it is not CPA secure! Moreover, the Jacobi symbol of m can be computed from $Enc(pk, m)$.

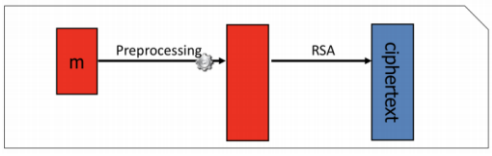
On the other hand RSA is correct. Therefore, for N, e, d as constructed in RSA definition and for arbitrary $m \in \mathbb{Z}$, it holds: $m^{ed} = m \; mod \; N$.

To improve the RSA performance we want to speed up the RSA encryption. We can use the minimal possible value for e: $e = 3$. (2 is not possible, because of $gcd(2, \phi(N)) = 2$). A recommended value for e is $e = 65537 = 2^{16} + 1$. It has a runtime for an encryption of 17 modular multiplications and therefore a fast encryption but a slow decription.

In the RSA encryption d is not allowed to be small. We have found out, that given $d < N^{0.292}$ we can easily discover d. If d is recoverable when $d < N^{0.5}$ is still a mystery.

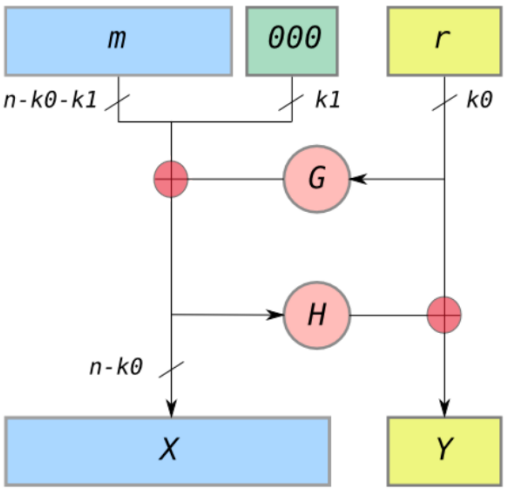
Optimal Asymmetric Encryption Padding

Because naïve RSA is still insecure, we need to make it better by introducing pre-processing.



The optimal asymmetric encryption padding is an encoding/decidoding principal. To encode:

1. the messages are padded with k_1 zeros to be $n - k_0$ bits in length
2. r is a randomly generated k_0-bit string
3. G expands the k_0 bits of r to $n - k_0$ bits.
4. $X = m00..0 \oplus G(r)$
5. H reduces the $n - k_0$ bits of X to k_0 bits
6. $Y = r \oplus H(x)$
7. The output is $X \,||Y$ where X is shown in the diragram as the most left block and Y as the rightmost block.



To decode:

1. Recover the random string as $r = Y \oplus H(X)$
2. Recover the message as $m00..0 = X \oplus G(r)$

Recommended key-lengths are:

Year	Minimum of Strength	Public key length
2010	80 bits	1024 bits
2011-2030	112 bits	2048 bits
> 2030	128 bits	3072 bits
>> 2030	192 bits	7680 bits

Digital Signatures

Only the secret key allows for creating signatures. Everybody can verify the validity of signatures using the respective public key. Signatures serve as undisputable evidence that the respective person signed the message.

Definition: A digital signature scheme with key space $\mathcal{K}$, message space $\mathcal{M}$, and tag space $\mathcal{T}$ is a triple of algorithms (Gen, Sig, Ver):

- The randomized key generation algorithm Gen takes no and returns a key-pair $(pk, sk) \in \mathcal{K}$.
- The (randomized ore stateful) signing algorithm Sig takes a secret key sk and a message $m \in \mathcal{M}$ and returns a tag $t \in \mathcal{T}$.
- The deterministic verification algorithm Ver takes a public key pk, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$ and returns a bit $\{0, 1\}$.

Correctness: The above algorithms have to satisfy the following property: for any key-pair $(pk, sk) \in \mathcal{K}$, any message $m \in \mathcal{M}_{pk}$, and any tag $t \leftarrow Sig(sk, m)$, we hate that $Ver(pj, m, t) = 1$.

When we compare MACs with Digital signatures, we can see the advantage, that we have public verifiability. Everybody can verify the signature and the integrity proofs can be trusted. Also we have a non-repudiation, so the signer cannot deny haing signed a document. With MACs, the key is kept secret, even if it is revealed to the judge, the judge does not know, whether it is the real one or one

faked a posteriori and even the judge knows somehow the right key, the judge does not know which one of the weo parties has signed the document.

Cryptographic Protocols

The web is a untrusted world, where everyone can read and write the messages in transit on the network.

Cryptography helps to secure our messages but unfortunately, cryptography is not enough. An attacker can circumvent cryptography and break the security goals of the protocol by simply intercepting, duplicating and sending back the messages in transit on the network, without the need to break the encryption scheme. In the following, we assume that cryptography is a fully reliable black box and focus on how cryptography is used.

For securing a communication we need to identify our security goals, determine the threat model (which enemy di we have to defend ourselves from) and protect the messages in transit on the network accordingly. Messages are protected by cryptography, but the way this is done depends on the security goals and the attacker's capabilities.

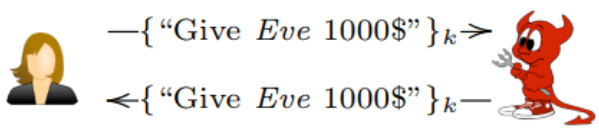
Some security goals are:

- Secrecy: Only the authorized recipient should be able to learn the message.
- Integrity: The recipient should be able to determine whether the message has been altered during transmission or not
- Authenticity: Where we have the non-injective and the injective agreement. The non-injective agreement states that the recipient of an authorized request should be able to verify the identity of the requester and both should agree on their respective roles. On the other hand, the injective agreement states the same as the non-injective agreement with the addition that the recipient should be able to verify the freshness of the authentication request.

Even though attacks are often surprising and hard to predict, they can be roughly classified according to the kind of interaction between the attacker and the protocol sessions.

Interleaving attack

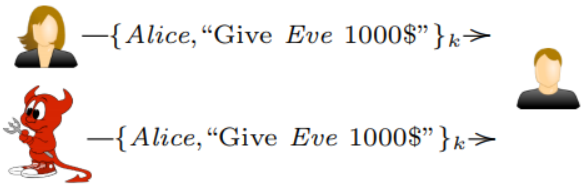
A message generated in a protocol session is exploited by the attacker to interact with another simultaneously ongoing session.



The reflection attack is a special type of an interleaving attack. In an reflection att ack a message is sent back to its generator. This works when the victim is playing multiple roles in the protocol, possibly in different sessions. The attacker intercepts the transfer request and sends it back to Alice, who recognizes the message as generated by Bob and performs the transfer. The symmetric nature fo the encryption key k does not allow Alice to verify whether the ciphertext has been generated by Bob or by herself.

A solution is to break the symmetry of the cryptographic scheme by inserting the originator's identifier (or the intended receiver's one).

Replay attack



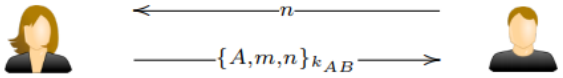
At a reply attack, the same message is duplicated and sent several times to the intended recipient. A possible solution is to insert a timestamp t for guaranteeing the freshness of the message. The authentication request is accepted only if it has been recently generated and no authentication request with the same timestamp has previously been accepted. This involves a global clock and brings synchronisation issues. Another solution is to exploit a challenge-response nonce handshake. A nonce is a randomly generated number n used in a single protocol session and is then discarded. An authentication request is accepted only if no authentication request with the same nonce has previously been accepted.

Challenge-response handshakes

Challenge-response nonce handshakes are very common in cryptographic protocols and they can be implemented in different ways: Plain-Cipher (PC), Cipher-Plain (CP) and Cipher-Cipher (CC).

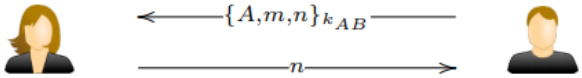
The common idea is, that principals prove their identities by encrypting / decrypting the challenges and the responses. The security properties provided by the three nonce handshakes are however different.

PC handshake



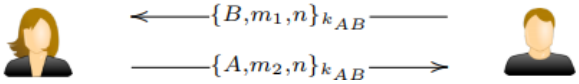
This protocol is also known as ISP two-pass unilateral authentication protocol. It garantees an injective agreement (A authenicates with B). The response migth be signed with Alice's private key and the identifier A replaced by B. Otherwise the reciever is not specified.

CP handshake



Bob authenticates with Alice receiving message m. Only Alice can decrypt the message. The second message can be sent as a receipt acknowledgement. This guarantees a non-injective agreement (B authenticates with A) and a injective agreement (A authenticates with B). The challenge might be encrypted by Alice's public key (and the identifier A replaced by B). But then the Non-injective agreement would not be guaranteed, since the challenge might originate from the attacker.

CC handshake

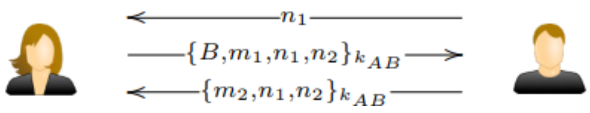


Bob authenticates with Alice receiving message m_1 and sending message m_2. Therefore, this protocol guarantees a non-injective agreement (B authenticates with A) and an injective agreement (A authenticates with B). We cannot replace A or B in the first message, because otherwise a reflection-attack would be possible. The challenge might be encrypted with Alice's public key and the

response might be encrypted with Bob's public key. The resulting protocol would not guarantee the non-injective agreement, because the challenge might originate from the attacker.

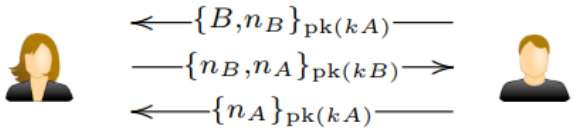
Mutual authentication protocols

Nonce handshakes can be combined I order to allow principals to authenticate each other. This protocol is also known as ISO three-pass authentication protocol and it is composed of a PC handshake and a CC handshake. It guarantees an injective agreement in both directions.

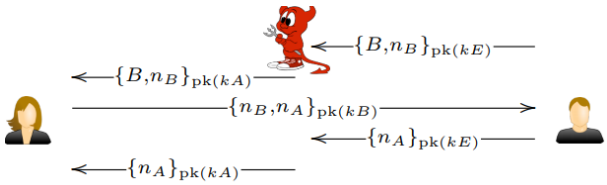


Needham-Schroeder Protocol

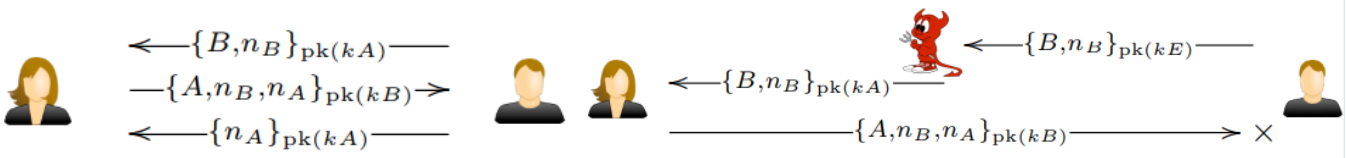
In the Needham-Schroeder-Protocol $pk(kA)$ and $pk(kB)$ are Alice and Bob's public keys respectively. This protocol was proposed in 1978 and the aim is to guarantee the secret and the authentication of the two nonces, which are then used for generating a symmetric session-key shared between Alice and Bob. All messages are encrypted with two CC handshakes.



Unfortunately, this protocol is not secure and an attack was discovered in 1996. There is a men-in-the-middle attack (MITM) possible. The adversary E steps into the communication path and simply relays (possibly without changing) the messages between legitimate parties A and B, itself acting as a part of the communication. A believes B is authenticating with her, while B is authenticating with E. In the end, E learns the two nonses and can build the session key that A uses to talk with B.



The supposed fix is adding Allice's identifier in the second ciphertext. Then B rejects the second ciphertext, as it does not come from E.



Prudent Engineering Practice

Principle 1: Every message should say what it means: the interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content, though if there is a suitable formalism available that is good too.

Principle 3: If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

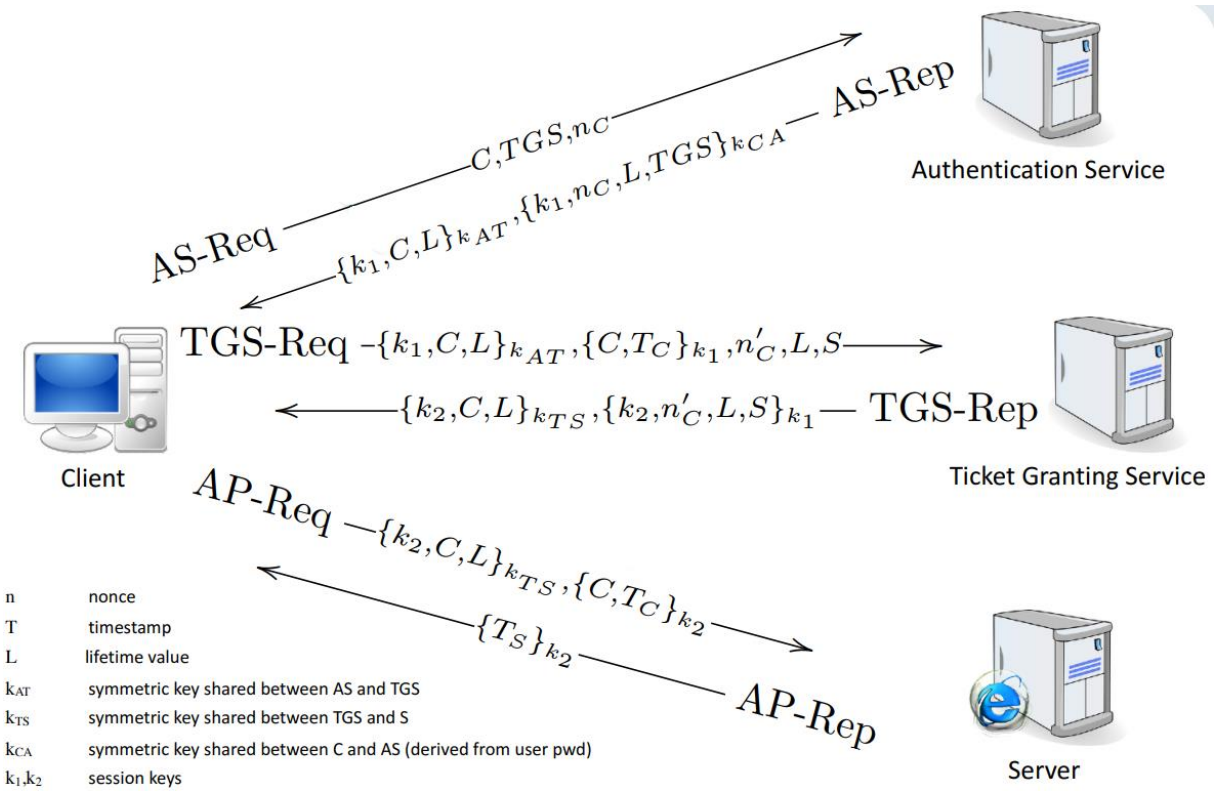
There are in sum 11 principals, but 1 and 3 are the most important ones.

Examles on slides 30 – 33

Kerberos

Kerberos is the holy grail of system administrators. It is a secure single sign-on solution. A user authenticates only once (a day) and can transparently access an arbitrary number of services. It is the default authentication mechanism for Active Directory (since Win2000) and probabli the mist widely used authentication protocol for end users in their company LAN.

[1] A client authenticates with an authentication-server and gets a 'ticket granting ticket' (e.g. once per day). [2] Then the client forwards this ticket-granting-ticket (TGT) to the ticket-granting-service and gets a 'service ticket'. [3] Finally the client forwards the service ticket (ST) to the server and can access the service.



Kerberos is frequently used to implement the login service. Until Kerberos 4.4 the underlying protocol composed the first tow message exchanges: The user types the password and the system checks that the second ciphertext in AS-Rep can be decrypted with the key k_{CA} derived from the user-provided password. The system also checks if the nonce in the ciphertext is fresh.

Attacks on Kerberos

The first attack on Kerberos was **KDC Spoofing**. The first attack was discovered in 2000. The attacker types the fake password corresponding to kE, intercepts AS-Req, and forges AS-Rep. Then the attacker can login in place of a honest user under the following (fairly weak) assumptions that the attacker has

physical access to the machine and the attacker can manipulate network traffic. The obvious solution (implemented in Windows) is to run also the next step of the protocol. A fictious service S for every machine in the network is created on the KDC and the key k_{TS} is stored on the corresponding machine. The client checks that the two ciphertextx in TGS-Rep can be decrypted and that the nonce is fresh.

The second attack, discovered in 2008, is the **pass-the-ticket-attack**. It consists of two phases. In the first we sniff the service ticket and in the second we make a men-in-the-middle-attack.

Details about this attack slides 41 – 42 and following attacks slides 43-47

SSL/TLS

The Secure Sockets Layer and the Transport Layer Security are two protocols with similar protocol design but different crypto algorithms. They are the de facto standard if internet security. “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”. It is deployed in every web-browser, also in VoIP, payment systems, distributed systems and more.

SSL/TLS guarantees end-to-end secure communication in the presence of network attackers, where attackers completely owns the network: controls Wi-Fi, DNS, routers, his own websites, can listen to any packet, modify packets in transit, inject his own packets into the network.

The scenario is that you are reading your email from an internet calé connected via a routed Wi-Fi access point to a dodgy ISP in a hostile authoritarian country.

SSL consists of two protocols. The first one is a handshake protocol, where it uses a public-key cryptography to establish several shares secret keys between the client and the server. The second one is the record protocol. This uses the secret keys established in the handshake protocol to protect confidentially, integrity, and authenticity of data exchange between the client and the server.

TLS also uses a handshake protocol and a record protocol.