OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Solve the exercises

Secure Programming

Authentication

Prof. Dr. Christoph Skornia
christoph.skornia@oth-regensburg.de

Authentication

Authentication is:

Proof that you are the one that you claim!

This can be done through:

1. Things you know

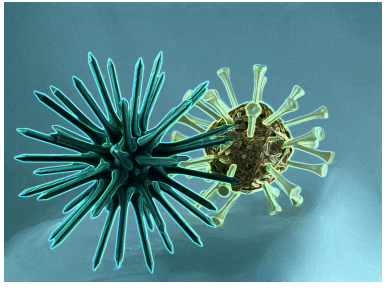
   Examples: passwords, PIN numbers, or passphrases, answers to personal questions

2. Things you have

   Examples: ATM cards are common physical tokens that are often implicitly used for authentication

3. Things you are

   Examples: fingerprints or voice analysis.



Authentication

Important requirements for authentication:

- ❑ Practicality of deployment DNA-Test is secure but inefficient practicality
- ❑ Usability
- ❑ Use across applications
- ❑ Patents
- ❑ Efficiency
- ❑ Common mechanism
- ❑ Economy of expression DNA gives way more info than necessary
- ❑ Security
- ❑ Recoverability from randomness problems
- ❑ Forward secrecy (communication is still secure, even if you lose private key

Authentication

Popular technologies:

"pillars of authentication"

- Password — never store password in clear-text; rather a hashed version ("no-way-back") — when processing them use key derivation-function
  - Unix crypt() or md5crypt
  - Password Based Key Derivation Functions (PBKDF2)
  - One-Time-Password systems (S/KEY or OPIE)
  - Challenge Response Authentication Mechanism (CRAM) — use pw only in CRAM if possible
  - Digest-Auth (RFC2617)
- PKI — Public Key Infrastructure
  - SSL certificate based checking
  - Public Key Exchange
- Directory Based Mechanisms — especially if company has active directory
  - LDAP — Lightweight Directory-Access-Protocol
  - Kerberos (RFC 1510) (incl. authorization)
- Biometric authentication

Authentication

Passwords should be:

- ❑ secret
- ❑ hard to guess
- ❑ easy to remember!!!

⇨ check passwords you accept (http://www.passwordmeter.com/)

- ❑ best way to check: use password cracker (e.g. cracklib)

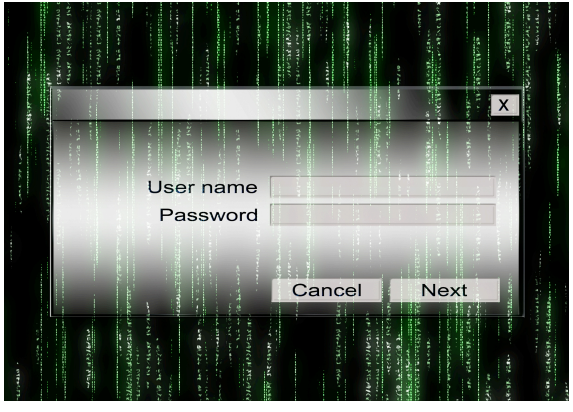
```
char *FascistCheck(char *pw, char *dictpath);
```

Authentication

Getting Passwords:

don't just use a regular input function

- Unix: `char *getpass(const char *prompt);`
- Windows: use `EDIT` controls with `ES_PASSWORD and EM_SETPASSWORDCHAR`



Encrypting Passwords:

- either regular crypt or more advanced hashing mechanisms with salt

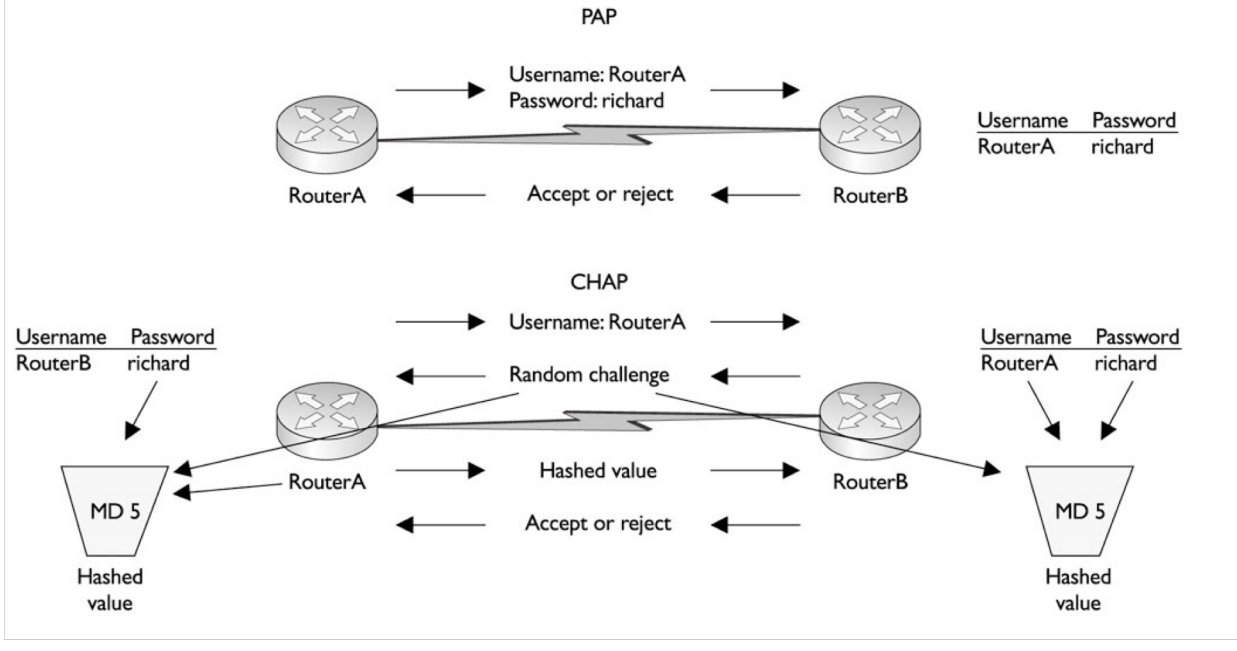
never any reason to store pws in cleartext

Authentication

Checking Password:

- Local: check against local pw-database (encrypted…)
- Remote: 1. don't send unencrypted passwords over the net!!!

challenge - authentication / ...

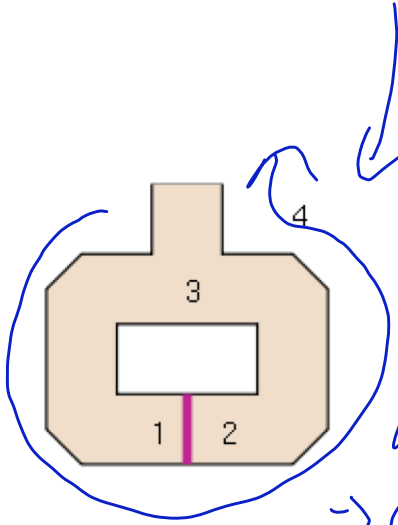


Authentication

- Remote:

2. If possible agree on encryption prior to password exchange (RSA or DH)

3. In case validator is not trusted fully: Use Zero knowledge mechanisms (e.g. Fiat-Shamir-Feige):



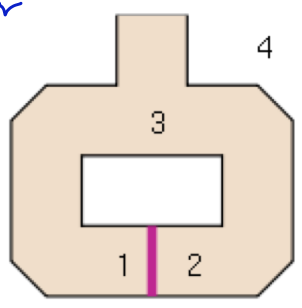
- Ali Baba

if you come out the opposite side, you can assume that the person had the key to the door
→ can prove it without knowledge about the lock itself: „zero-knowledge"

Authentication

Feige-Fiat-Shamir identification

Victor shouldn't know anything about the secret, other than the fact that Peggy knows the secret



- Choose two prime integers p and q
- compute $N = pq$ and publish it
- Create secret numbers $s_1 ... s_k$ with $gcd(s_i, N) = 1$
- compute $v_i = s_i^2 \bmod N$ and send it to Victor

verification: no knowledge about original number after s^2 : $\mathbb{Z}/N\mathbb{Z}$ — $s^2 \bmod N \to s \bmod N$ hard to find s

prove you have s with just s^2

proof

1. Peggy chooses a random integer r, a random sign $s \in \{-1, 1\}$ computes $x = s \cdot r^2 \bmod N$ and sends it to Victor
2. Victor chooses numbers $a_1, ..., a_k$ where $a_i \in \{0, 1\}$ and sends the numbers to Peggy
3. Peggy computes $y = r s_1^{a_1} \cdots s_k^{a_k} \bmod N$ and sends it to Victor
4. Victor checks that $y^2 = \pm x v_1^{a_1} \cdot \cdot v_k^{a_k} \bmod N$

s_1, s_2, s_3

$(a_1, a_2, a_3) = (0, 0, 1)$

$r \cdot s_1^0 \cdot s_2^0 \cdot s_3^1 = r \cdot s_3$ → doesn't know s

v.l.l. natural sequence

Feige, Uriel; Fiat, Amos; Shamir, Adi (1988). "Zero-knowledge proofs of identity" Journal of Cryptology 1 (2): 77–94.

Authentication

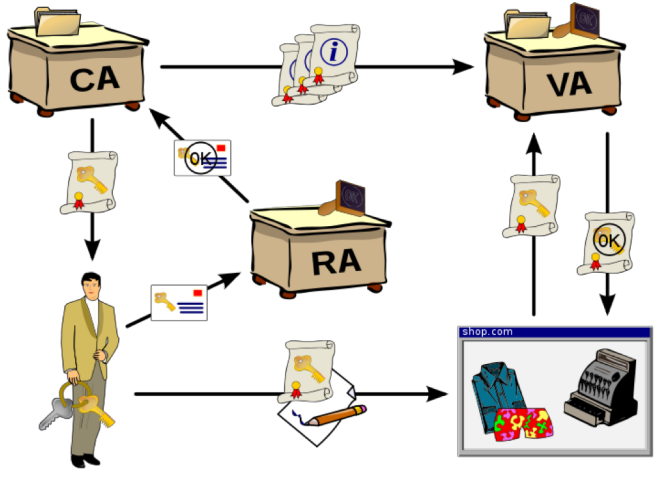
Directory Services:

- A directory service has basically nothing to do with authentication
- However a directory services stores personal information
- Reasonable to store also authentication credentials (encrypted password, public key, encrypted private key) in directory services.
- Popular Services:
  - Active Directory
  - Novell eDirectorsy (NDS) ~ an Uni ; NRW- Verwaltung
  - Apache DS
  - NT-Domain
  - Open LDAP
- Most Popular Protocol: LDAP (Lightweight Directory Access Protocol)

```
int ldap_search_ext(
    LDAP *ld,
    char *base,
    int scope,
    char *filter,
    char *attrs[],
    int attrsonly,
    LDAPControl **serverctrls,
    LDAPControl **clientctrls,
    struct timeval *timeout,
    int sizelimit,
    int *msgidp );
```

Authentication

P(ublic)K(ey)I(nfrastruktur)



- Certification Authority (CA):
  - issues and signs certificates
  - creats, updates and publishes CRLs
  - offers Online Certificate Status Protocol (OCSP) for clients
- Registration Authority (RA):
  - guarantees that public key belongs to a specific person or entity
- Directory Service: distributes certificates and CRLs
- Validation Authority (VA):
  - real-time validation of certificates

registrant for a certain timeframe

also possible to have a server that checks if certificat is revoked; single point of failure!

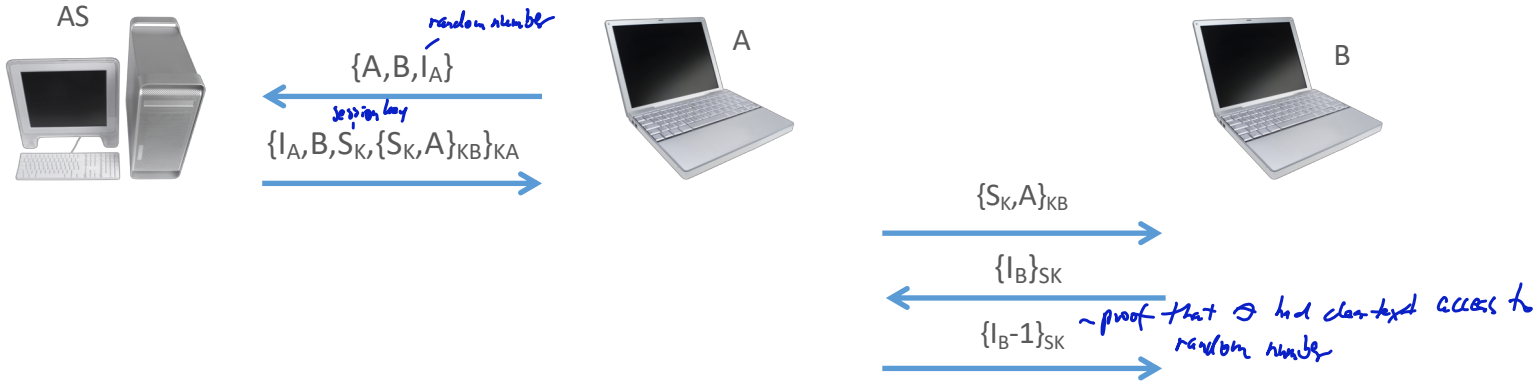
List of revoked certificates
↳ not scalable!

anecdotal: no revoking for most certificates
because faked certificates are better than a possible failure/compromising of revoking mechanism

Authentication

Needham-Schroeder Protokoll (symmetric):

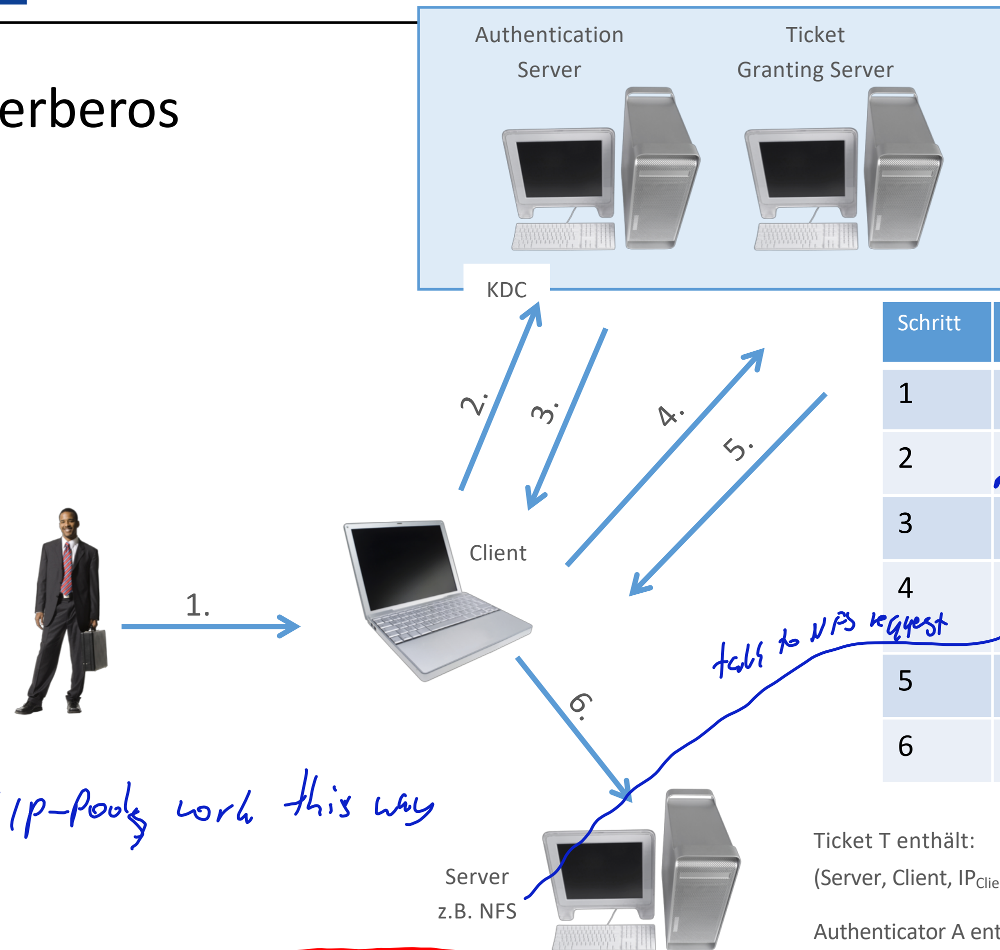
- Components:
  - authentication server AS,
  - K_A, K_B: secrect keys (only AS and A or B know)
  - Nonce I_A
  - symmetric key S_K



Authentication & Authorization

Always related to time; Authentication isn't

- Kerberos



Schritt	Inhalt
1	Joe, Passwort
2	Joe, TGS, $Nonce_1$, $\{Joe, Time\}_{KJoe}$
3	$\{K_{Joe,TGS}, Nonce_1\}_{KJoe}$, $\{T_{Joe,TGS}\}_{KTGS}$
4	$\{A_{Joe}\}_{KJoe,TGS}$, $\{T_{Joe,TGS}\}_{KTGS}$, NFS, $Nonce_2$
5	$\{K_{Joe,NFS}, Nonce_2\}_{KJoe,TGS}$, $\{T_{joe,NFS}\}_{KNFS}$
6	$\{A_{Joe}\}_{Kjoe,NFS}$, $\{T_{Joe,NFS}\}_{KNFS}$

Ticket T enthält:

(Server, Client, IP_{Client}, aktuelle Zeit, Lebenszeit des Tickets, $K(ey)_{c,s}$)

Authenticator A enthält:

(Client, IP_{Client}, aktuelle Zeit)

Authentication & Authorization

Fin

How about federated accounts? Single Sign On crossing company borders?

Examples:

- Microsoft account
- Google Account
- Twitter
- LinkedIn
- PayPal
- Foursquare
- Amazon

Authentication & Authorization

Example:

Access Control in open service-platforms

- SAML: XML-based framework to exchange security-information in distributed environments
- Single-Sign-on for different administrative domains

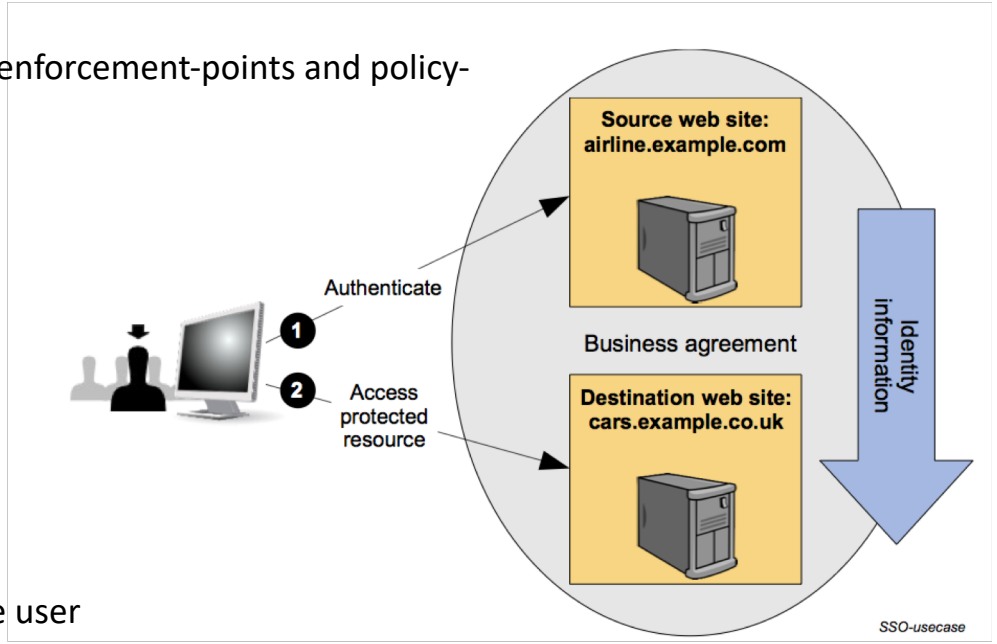
- SAML V2.0 (Security Assertion Markup Language)

  OASIS-Standard (Organization for the Advancement of Structured Information Standards)

  - **assertions** are given by an issuer to a
    - authentication assertion: subject is authenticated
    - authorization decision assertion: subject is authorized
    - attribute assertion: subject has specific attributes

Authentication & Authorization

- SAML-Protocol
  - used to exchange information between policy-enforcement-points and policy-decision-points
  - Q&A message-format: SAML-request/reply

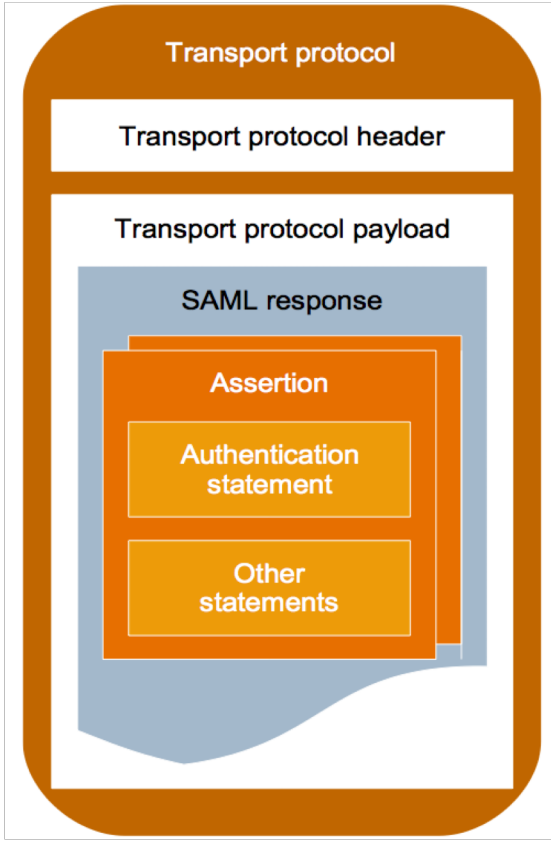


- Example
  1. user logs on at AirlineInc.com
  2. user wants to access: CarRentalInc
  3. AirlineInc.com asserts CarRentalzu to know the user
  4. CarRental trusts the assertion of AirlineInc.com
  5. no need to logon at CarRentalInc again

Authentication & Authorization

- Authentication Assertion
  - The issuing authority asserts that:
  - subject S has itself authenticated at time T with method M

```
 1: <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 2:   Version="2.0"
 3:   IssueInstant="2005-01-31T12:00:00Z">
 4:   <saml:Issuer Format=urn:oasis:names:SAML:2.0:nameid-format:entity>
 5:     http://idp.example.org
 6:   </saml:Issuer>
 7:   <saml:Subject>
 8:     <saml:NameID
 9:       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
10:         j.doe@example.com
11:     </saml:NameID>
12:   </saml:Subject>
13:   <saml:Conditions
14:     NotBefore="2005-01-31T12:00:00Z"
15:     NotOnOrAfter="2005-01-31T12:10:00Z">
16:   </saml:Conditions>
17:   <saml:AuthnStatement
18:     AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
19:     <saml:AuthnContext>
20:       <saml:AuthnContextClassRef>
21:         urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
22:       </saml:AuthnContextClassRef>
23:     </saml:AuthnContext>
24:   </saml:AuthnStatement>
25: </saml:Assertion>
```



Authentication & Authorization

- Attribute Assertion
  - Issuing authority asserts, that:
    - subject S is associated with the attributes A, B, …
    - With values “a”, “b”, “c”…
  - Single attributes are specified through attribute statements

```
12:      Name="LastName">
13:      <saml:AttributeValue
14:        xsi:type="xs:string">Doe</saml:AttributeValue>
15:    </saml:Attribute>
16:    <saml:Attribute
17:      NameFormat="http://smithco.com/attr-formats"
18:      Name="CreditLimit">
19:      xmlns:smithco="http://www.smithco.com/smithco-schema.xsd"
20:      <saml:AttributeValue xsi:type="smithco:type">
21:        <smithco:amount currency="USD">500.00</smithco:amount>
22:      </saml:AttributeValue>
23:    </saml:Attribute>
24: </saml:AttributeStatement>
```

Authentication & Authorization

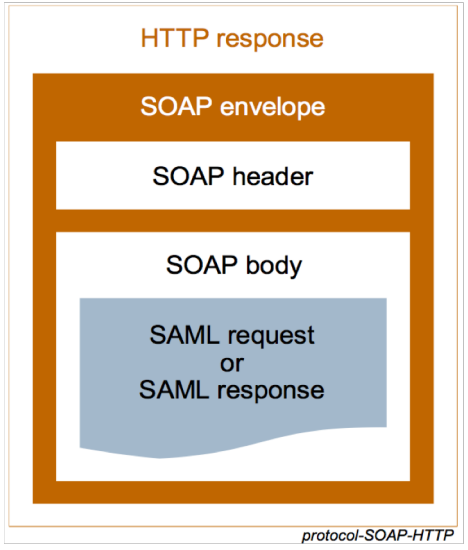
- Authorization Decision Assertion
  - Issuing authority decides if a request is granted:
    - Request of subject S to access A
    - on ressource R
    - under condition E

```
<saml:Assertion …>
        <saml:Conditions …/>
        <saml:AuthorizationStatement
                Decision="Permit"
                Resource="http://jonesco.com/rpt_12345.htm">
                <saml:Subject>
                        <saml:NameIdentifier
                                SecurityDomain="smithco.com"
                                Name="joeuser" />
                </saml:Subject>
        </saml:AuthorizationStatement>
</saml:Assertion>
```

Authentication & Authorization

- SOAP-based exchange of assertions
  - SAML-authentication-request embedded in SOAP-message (line 4)
  - line 5: request
  - Specification of expected response: e.g. line 7: force new authentication

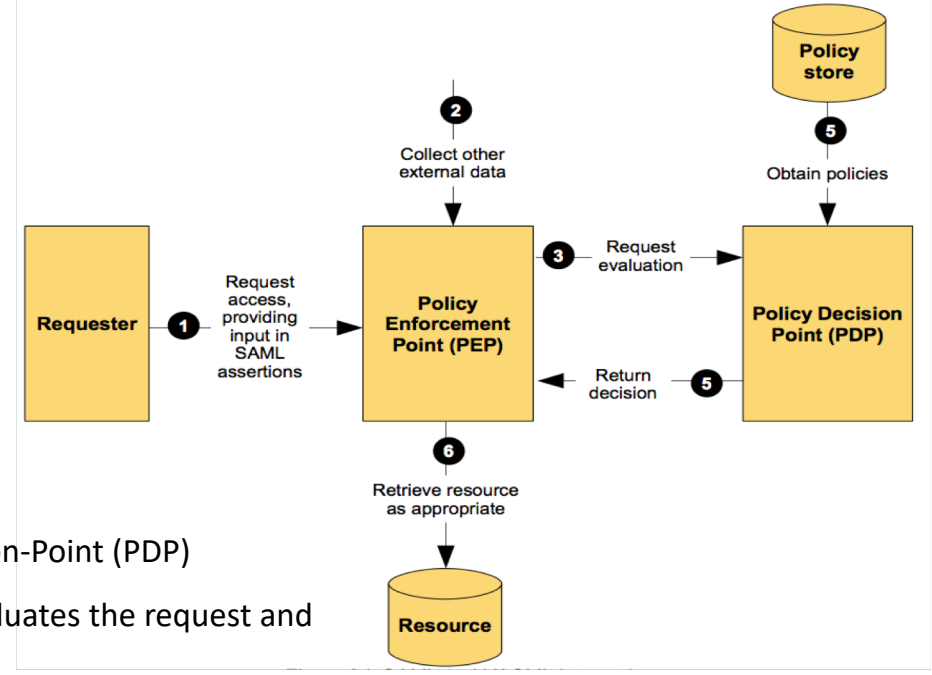
```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <env:Envelope
 3:   xmlns:env="http://www.w3.org/2003/05/soap/envelope/">
 4:   <env:Body>
 5:     <samlp:AuthnRequest
 6:       xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
 7:       ForceAuthn="true"
 8:       AssertionConsumerServiceURL="https://www.sp.example.com/SSO"
 9:       AttributeConsumingServiceIndex="0" ProviderName="CarRentalInc.com"
10:       ID="abe567de6"
11:       Version="2.0"
12:       IssueInstant="2005-01-31T12:00:00Z"
13:       Destination="https://www.idp.example.com/" >
14:       <saml:Subject
15:         xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
16:         <saml:NameID
17:           Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
18:             j.doe@acompany.com
19:         </saml:NameID>
20:       </saml:Subject>
21:     </samlp:AuthnRequest>
22:   </env:Body>
23: </env:Envelope>
```



Authentication & Authorization

XACML (eXtensible Access Control Markup Language)

- OASIS-Spezification (http://www.oasis-open.org)
- XML-based language, to specify access policies
- generic schema:



1. XACML Policy-Enforcement-Point (PEP) gets access request
2. PEP collects SAML-Assertions about requestor, time/date, location and properties or resource …
3. PEP hands over collected information to the Policy-Decision-Point (PDP)
4. PDP is looking up matching entries in the policy-store, evaluates the request and manages potential conflicts
5. PDP sends response to PEP: permit, deny, not applicable, indeterminate (e.g. request not complete)
6. PEP enforces decision (allow or deny access)

Authentication & Authorization

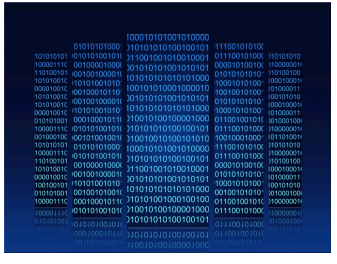
Alternatives: OpenID and OAuth

	OpenID	OAuth	SAML
Dates from	2005	2006	2001
Current version	OpenID 2.0	OAuth 2.0	SAML 2.0
Main purpose	Single sign-on for consumers authentication only	API authorization between applications	Single sign-on for enterprise users authentication and authorization
Protocols used	XRDS, HTTP	JSON, HTTP	SAM, XML, HTTP, SOAP
No. of related CVEs	24	3	17

How about Passwords in Source Code?

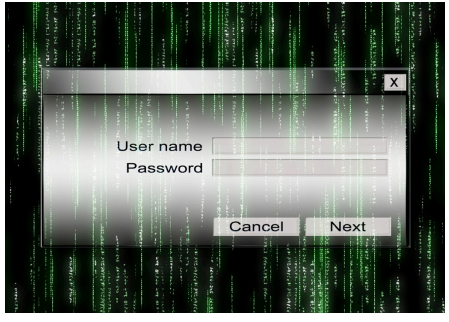
```
...
private static String passwd = "mYv3rYSECr3tPWD";
...
db = MySQLdb.connect(host="db.server.com", user="admin", passwd="NOBODYwillEVERguess", db="sales")
...
String url = "jdbc:mysql://" + serverName + "/access?user=webclient&password=ILoveJuliet";
...
./remove_temp_files.sh --machine=appserv$i --rootpassword="*d3H%sS-W"; done
...
```

- Any secret information has to be kept out of source code!
  - Why?
    - its all about lifetime and access
      1. changing secret information (e.g. password change) gets difficult
      2. easy to attack by anyone who has the binary (and the source code anyway…)



How about Passwords in Source Code?

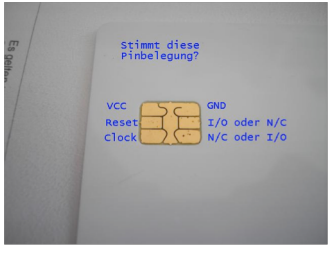
- Alternatives?
  1. Ask the user who is starting the binary!
     - remember to use a secure way to ask (see authentication…)
     - **absolutely best way**
     - not always applicable (server-jobs, scripts without user interaction)
  2. store secret in a separate file or database and secure that
     - Advantage over securing secrets in code?
       - easy to change (or even to remove if not needed)
       - no CVS or other uncontrolled duplication
     - Better but of course not good!



How about Passwords in Source Code?

- Alternatives?

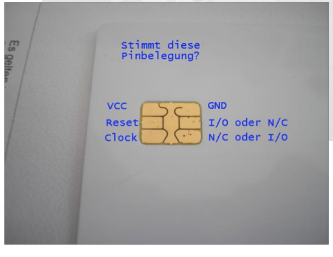
3. Encrypt
   - if more secret information is needed than can be asked from a user: encrypt and ask the user or OS for the key (PBKDF2)!
   - use hardware key containers (e.g. Smartcards) with PKCS#11
   - if OS is trustworthy this can be quite OK with very limited user interaction (through startup of OS)
   - storing secrets in RAM encrypted is better than storing in clear even if the key is also stored in RAM! Why?



How about Passwords in Source Code?

- Alternatives?

4. Use existing external mechanisms for authentication
   - LDAP or other directory services
   - RADIUS
   - Kerberos
   - PKI (PKCS#11)
5. Use already existing credentials
   - in SSO environments, there might be already credentials or tickets available which be be used to retrieve needed information
   - good solution, if applicable



How about Passwords in Source Code?

- Alternatives?

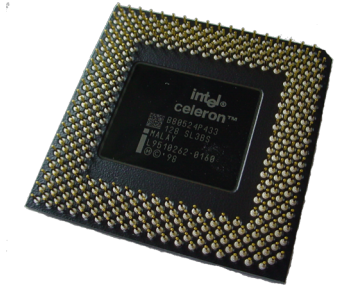
6. Use hashing wherever useful
    - Example: If your code needs to verify credentials, there is no need to store the credentials itself, a salted hash would be enough
    - Advantage: Hashing is irreversible..

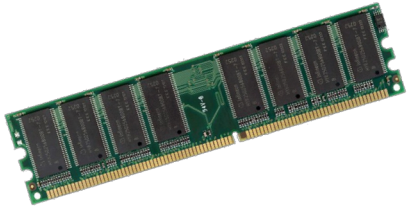
How about Passwords in Source Code?

- Conclusion:
  - For any case there are better alternatives than having secrets in source code!
  - Evaluate your use case and choose the best solution
  - Remember:
    - If your OS is broken, everything is broken!!!

Secrets during execution

- Where are secrets during execution?
- CPU?
  - If you want to use the secret, you will need it sooner or later in the CPU
- RAM?
  - Very very likely you want store the secret in RAM to reuse it or to get it to CPU from RAM
- Hard disk?
  - depends, whether you are asking the secret from the user
  - if RAM is paged out to HD





Secrets during execution

- Who can get access to a secret in:
- CPU?
  - OS (e.g. pipe process through debugger, virtual cpu etc.)
- RAM?
  - OS (of course)
  - other processes of same user or with higher privileges (if allowed by OS)
- Hard disk?
  - OS (of course)
  - other processes of same user or with higher privileges
  - anyone who has physical access to the disk (persistent memory)

If your OS is broken, everything is broken!!!

Secrets during execution

- Who can get access to a secret **after** execution:
- CPU?
  - nope, everything gone
- RAM?
  - depends if secret is still there… freeing memory is not enough, overwriting is necessary
  - OS and other processes
- Hard disk?
  - depends if secret is still there…
  - OS and other processes
  - anyone who has physical access to the disk (persistent memory)

Secrets during execution

- Who can get access to a secret **after system shutdown**:
- CPU?
  - nope, everything gone
- RAM?
  - nope, everything gone… usually!
  - exception: cold boot attack
- Hard disk?
  - depends if secret is still there…
  - by default pagefile is not deleted by OS (at least don't rely on it)

Secrets during execution

- Consequences:
  - make sure to delete memory securely after secrets are not needed anymore, but latest before process terminates.

```
int get_and_verify_password(char *real_password) {
    int  result;
    char *user_password[64];

    get_password_from_user_somehow(user_password, sizeof(user_password));
    result = !strcmp(user_password, real_password);
    memset(user_password, 0, strlen(user_password));

    return result;
}
```

  - Better:
    - C, C++

```
volatile void *guaranteed_memset(volatile void *dst, int c, size_t len) {
    volatile char *buf;

    for (buf = (volatile char *)dst;  len;  buf[--len] = c);
    return dst;
}}
```

    - Windows:

```
PVOID SecureZeroMemory(
        _In_  PVOID ptr,
        _In_  SIZE_T cnt
        );
```

Secrets during execution

- Consequences:
  - make sure to avoid paging of secret information
    - Unix: Lock address space of process or address-range

```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

    - Windows:

```
BOOL WINAPI VirtualLock(
     _In_  LPVOID lpAddress,
     _In_  SIZE_T dwSize
     );
```

Secrets during execution

- Consequences:
  - use further OS-based memory and process protection technologies…
  - use advanced technologies like TRESOR or aes-amnesia to move encryption to the cpu-cache without keys in the RAM
- Baseline:
  - protecting a secrets on a running system is difficult
  - DRM-technologies try since years and found out to need Locked-Down-Systems…

If your OS is broken, everything is broken!!!