Computer Security and Forensics notes

Computer Security & Forensics (University of Sheffield)

Computer Security and Forensics

Security Fundamentals & Access Control

- CIA triad
  - Confidentiality
    - Protecting info from being **disclosed** to unauthorized parties
  - Integrity
    - Protecting info from being **modified** by unauthorized parties
  - Availability
    - Ensuring info is **accessible** to authorized parties

- Identification and AAA
  - Subject (e.g. human); Access (e.g. read/write); Object (e.g. data, functional call)
  - Identification:- Associating identity with a subject
  - Authentication:- Verify validity of something (Identity claimed by system entity)
  - Authorisation:- Granting/denying the permission of a system entity to access an object
  - Access Control:- Controlling access of system entities to object based on access control policy

- Types of authentication
  - Something you **know**
  - Something you **have**
  - Something you **are**
  - **Current location**

  - Multi-factor authentication uses one or more of ^

- Access Control Models
  - RBAC (Role-Based Access Control)
    - Define ROLES {lecture, demonstrator, student}
    - Define USERS {achim, heidi, alice, bob}
    - Define Permission {write_lecture, read_lecture}
    - Define relation $UA \subset USER \times ROLES$ {(User, Role), …}
    - Define relation $PA \subset ROLES \times PERMISSION$ {(Role, Permission), …}

    - Hierarchic RBAC adds
      - Define role $RH \subset ROLES \times ROLES$ hierarchy
        - RH = {(lecturer, lecturer), (lecturer, demonstrator), (demonstrator, demonstrator), (demonstrator, student), (student, student)}

    - Instead of having two statements:
        - {(lecturer, read_lecture), (student, read_lecture)} = {(student, read_lecture)}
        - Because lecturer has permission of student

- Constraints
    - Throw in a predicate in PA
    - $(\texttt{student}, \text{is\_registered\_for\_comx501}(\textit{user}), \texttt{read\_comx501\_slides})$

- Multi-Level Access Control
    - State all entities that can read the file mentioned in a list {}

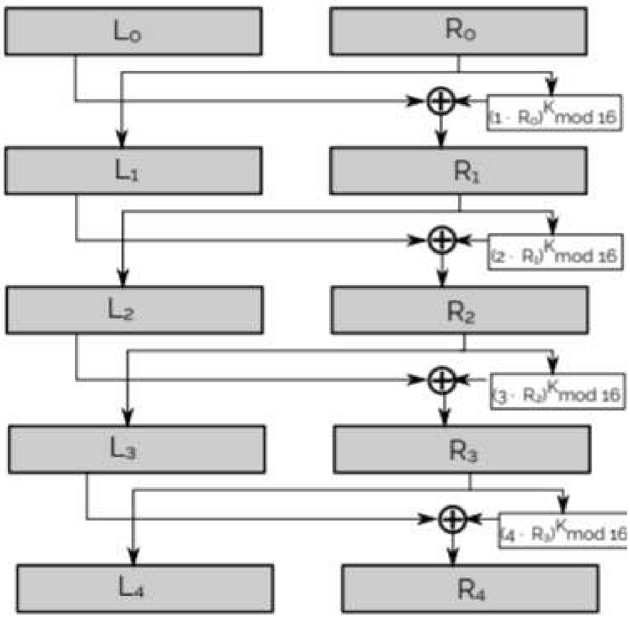
Cryptography & PKIs

- **Cipher:-** An algorithm performing encryption/decryption
- **Cryptanalysis:-** Deciphering encrypted message without key
- **Steganography:**-Hiding messages in other messages/images
- Transpositional cipher:- rearranges letters

- Work out large mods:-
  - Use formula to break down mods, then work your way back

$$b^e \mod n = \begin{cases} b^{e/2} \cdot b^{e/2} \mod n & \text{if } e \text{ is even} \\ b \cdot b^{(e-1)/2} \cdot b^{(e-1)/2} \mod n & \text{if } e \text{ is odd} \end{cases}$$

- Symmetric encryption
  - Same encrypt/Decrypt key
  - Tanspositional (T)
  - Substitution (S)
  - Examples: Composite cipher (S+T)), AES, Blowfish, ROT13 (S cipher)
  - DES (Memorize Diagram):
    - $f_i(x, K) = (i \cdot x)^K mod\,16$



- Asymmetric encryption:
  - Public key used for encrypting message/plain text/clear text
  - A key is pair of public and private key

        - Used as part of SSL/TLS
        - RSA
        - Public key encryption schemes
            - Key (n, e)
                - Encrypt: $c=m^e \bmod n$
                - Decrypt: $m=c^d \bmod n$

- No. of Symmetric keys = n(n-1)/2
- No of Asymmetric keys = 2n

- Digital signature
    - Needs an asymmetric key
    - Provides authentication and non-repudiation

- PKI:-
    - Combination of digital certificates, public-key cryptography and certificate authorities
    - Components
        - CA:-
            - Publishes certificate in directory
            - Maintains Certificate Revocation List (if CA gets stolen check problem sheet for e.g.)
        - Directory :-(Stores ^)
        - Registration Authority (RA):- Registers and issues certificate(binds identity to a key)
        - Clients
    - X.509
        - Check problem sheet
        - Intermediate CAs offer
            - Performance: root CA can use stronger key pair as less certificates need to be signed
            - Security: root CA certificate can be stored offline
            - Security: reduce the risk from a compromised CA certificate (less users will be affected)
            - Scalability: reduce work load (signing request) for a CA (i.e. organising CAs by country)

        - Transitive trust:- Bob trusts root CA, by checking signature along chain this trust is transitively extended until he can validate Alice.

- Direct trust:-
    - If all users subscribe to the same CA, then common trust of that CA
    - All users placed in the same directory for access by all users

- Hierarchical trust:-
    - Trust extends from root certificates
    - These certificates may certify themselves, or other certificates down the chain
    - Leaf certificate is verified by tracing back from its certifier until trusted root certificate is found

- Cross-certification:-
    - CAs exchange their public keys
    - ∴ A can obtain B's public key by chain of certificates

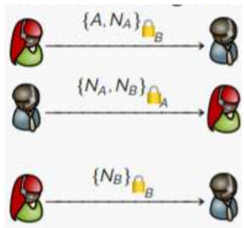
- Web of trust
    - Uses direct and hierarchical trust
    - PGP:-
        - Keys can be signed by any user
        - Keys may have multiple signatures (including self-signing)
        - No central infrastructure bc any use can act as CA
        - Can only validate users if recognise validator as trusted

Security Protocols

- Properties of a nonce
    - Freshness
    - Secret

- Make replay attack in authentication protocols harder by…
    - Nonce
    - Time-stamp
    - Monotonically increasing sequence number
    - Random number used no more than one

- NSPK
    - Lowe's attack
        - Eve speaks to Bob, pretending to be Alice
        - Fix
            - When Bob sends back the nonces, put in B



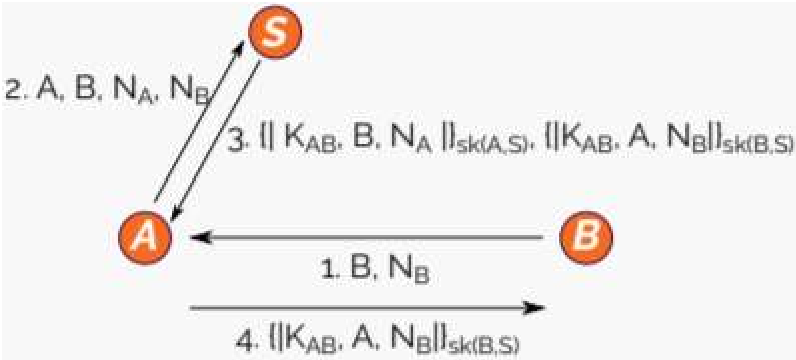
NSPK Protocol

Roles:	A, B or *Alice, Bob*
Agents:	a, b, i
Symmetric Keys:	$K, K_{AB}, \ldots;$ sk(A, S)
Symmetric Encryption:	$\{\|M\|\}_K$
Public Keys:	K, pk(A)
Private Keys:	inv(K), inv(pk(A))
Asymmetric Encryption:	$\{M\}_K$
Signing:	$\{M\}_{\mathrm{inv}\, K}$
Nonces:	$NA, N1$ fresh data items used for challenge/response. Sometimes, we may use subscripts, e.g. N_A, but it does not mean that principals can find out that N_A was generated by A
Timestamps:	T denotes time, e.g. used for key expiration.
Message concatenation:	M_1, M_2, M_3

Protocol Notation

- When analysing protocols, check using the following 4 security assumptions
    - The intruder is able to eavesdrop on all messages
        - E.g. Are you encrypting your messages?

- The intruder is able to intercept messages on the network and send message to anybody
  - E.g. Are you sending Roles with the session key?
- The intruder may be a legitimate protocol participant (an insider) or external party (outsider), or a combination of both
  - E.g. Are you sending Roles with the session key?
- The intruder is able to obtain the value of the session key used in an old/previous run of the protocol
  - E.g. Are you sending nonces with your messages?



Key-Establishment Protocol

- Attack types
  - Man-in-the-middle/parallel sessions
  - Replay/freshness:- reuse parts of previous messages
  - Masquerading:- pretend to be another principal
  - Reflection:- send transmitted information back to originator
  - Oracle:- take advantage of normal protocol responses as encryption and decryption 'services'
  - Binding:- use messages in different context/purpose than originally intended
  - Type flaw:- substitute a different type of message field

- Diffie Hellman Key-Exchange based on computing discrete logarithms

- Dolev-Yao Closure
    - Attackers can't decrypt all messages
    - But can; eavesdrop all messages, block all messages and decompose messages

    - Rules
        - Axiom:- Knowledge you already have
        - Composition:- Compute any public func with required axioms
        - DecSym:- Decrypt symmetric encryption (requires key)
        - DecAsym:- Decrypt asymmetric encryption “”
        - OpenSig:- Opens signature
            - If key is private(inv) it is a signature
        - Proj_i:- Chooses one element from a bracket, denoted by *i*
        - Algebra
        - Don't forget to put '$\in$ DY(M)' in every line

$$\frac{}{m \in \mathcal{DY}(M)} \text{Axiom } (m \in M) \qquad \frac{s \in \mathcal{DY}(M)}{t \in \mathcal{DY}(M)} \text{Algebra } (s \approx t)$$

$$\frac{t_1 \in \mathcal{DY}(M) \cdots t_n \in \mathcal{DY}(M)}{f(t_1, \ldots, f_n) \in \mathcal{DY}(M)} \text{Composition } (f \in \Sigma_p) \qquad \frac{\langle m_1, m_2 \rangle\rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)} \text{Proj}_i$$

$$\frac{\{|m|\}_k \in \mathcal{DY}(M) \quad k \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)} \text{DecSym} \qquad \frac{\{m\}_k \in \mathcal{DY}(M) \quad \text{inv}(k) \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)} \text{DecAsym}$$

$$\frac{\{m\}_{\text{inv}(k)} \in \mathcal{DY}(M)}{m \in \mathcal{DY}(M)} \text{OpenSig}$$

Application security

- CVSS (Common Vulnerability Scoring System)
  - AV (Access Vector)
    - Local(L)
      - Need physical access or local account
    - Adjacent Network(A)
      - Needs access to neighbouring network
    - Network(N)
      - Needs remote access

  - Access Complexity (AC)
    - High(H)
      - Specialised conditions must be fulfilled
    - Medium(M)
      - Some specialised conditions must be fulfilled (i.e. non-default)
    - Low(H)
      - No special conditions

  - Confidentiality (C)
    - None(N)
      - No impact on confidentiality
    - Partial(P)
      - Considerable information disclosure (but constrained)
    - Complete(C)
      - All information disclosed

- Integrity (I)
  - None(N)
    - No impact on integrity
  - Partial(P)
    - Modification of some data (but limited)
  - Complete(C)
    - Total loss of integrity

- Availability (A)
  - None(N)
    - No impact on availability
  - Partial(P)
    - Reduced performance or some loss of functionality
  - Complete(C)
    - Total loss of availability

- STRIDE
  - **S**poofing Identity
    - Can an attacker use a stolen device to authenticate the system using the victim's credentials?
  - **T**ampering with data
    - "" use an injection attack in the mobile app to modify local storage?
  - **R**epudiation
    - Can a user modify the data in the server's database without a trace/log?
  - **I**nformation Disclosure
    - "" eavesdrop the communication between the server and mobile app?
  - **D**enial of Service
    - Can users crash the server by uploading large amounts od data using the mobile app?
  - **E**levation of Privilege
    - How are users authenticated, i.e., is a non-authenticated user able to circumvent the authentication mechanism?

- Secure programming
  - SQL Injection
    - `WHERE userid = ' sdfssd '`**`or 1=1)`**`;`
    - Prevention
      - Validate any input that flows into SQL statement
      - Use prepared SQL statements (and use them correctly)
        - If you can't use prepared statement

            - Whitelist (specify allowed input)
            - NEVER Blacklist (specify forbidden characters)
        - Use stored procedures (and access rights on database tables)

    - Buffer overflow (Unlikely to come up
        - Can cause DOS
        - Prevention
            - Use counted versions of string functions
            - Use safe string libraries
            - Check loop termination and array boundaries
            - Use C++/STL containers instead of arrays
            - Check API and use it correctly

    - XSS
        - User input directly displayed in output webpage (e.g. a comment)
            - Can then send data back to his server i.e. document.cookie
        - '`.html_safe`' Says that the input is safe, not sanitise
        - Prevention
            - Sanitize any user input which might reach output statements (including those that write to database or save cookies)
            - Encode output using HTML encoding
                - So malicious links/JavaScript code stays uninterpreted by browser

- If a static analysis tool reports a finding (a weakness), the finding can be…
    - Exploitable **(true positive)**
    - Cannot be exploitable **(false positive)** (No weakness, but says there was one found)
        - A developer prefers 0 ^ as it avoid unnecessary effort
- If a static analysis tool does not report a finding, the code can be…
    - Secure **(true negative)**
    - Contain a vulnerability **(false negative)** (Weakness present, but says none were found)
        - A security expert prefers 0 ^ as it increases overall security risk

- SAST
    - Best when finding **generic defects** that are **visible in the code** (i.e. division if a division by 0 could occur)
        - E.g. buffer overflows
    - Risks
        - Wasting effort that could be used more wisely
        - Shipping insecure software

        - H.e. lower security risk than DAST
    - E.g. of limitations
        - Not all programming languages supported
        - Doesn’t cover all layers of software stack

- DAST
    - In a nutshell:
        - Fire up application
        - Feed with “strange input” (large random data, JavaScript, SQL)
        - Observe behaviour
    - Dangers
        - Break your IT landscape (e.g. mistyping an IP address)
        - Destroy/corrupt database
        - Violate compliance policies (granting access to data you’re not allowed to see)

- SAST vs DAST
    - SAST
        - Quicker, less configuration required, used on just source code
    - DAST
        - Access system from user p.o.v, thus check using broader range of vulnerabilities