Distributed Systems
Principles and Paradigms

Christoph Dorn

Distributed Systems Group,
Vienna University of Technology

c.dorn@infosys.tuwien.ac.at
`http://www.infosys.tuwien.ac.at/staff/dorn`

Slides adapted from Maarten van Steen, VU Amsterdam, steen@cs.vu.nl

Chapter 09: Security

Contents

Chapter
01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Security
10: Distributed Object-Based Systems
11: Distributed File Systems
12: Distributed Web-Based Systems
13: Distributed Coordination-Based Systems

Overview

- Introduction
- Secure channels
- Access control
- Security management

Security: Dependability revisited

Basics

A *component* provides *services* to *clients*. To provide services, the component may require the services from other components ⇒ a component may depend on some other component.

Property	Description
Availability	Accessible and usable upon demand for authorized entities
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Maintainable	Easy repair of a failure
Integrity	No accidental or malicious alterations of information have been performed (even by authorized entities)

Security: Dependability revisited

Observation

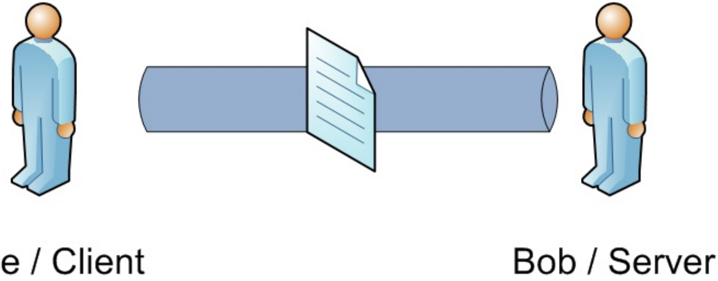
In distributed systems, security is the combination of availability, integrity, and confidentiality.

Property	Description
Confidentiality	No unauthorized disclosure of information
Availability	Accessible and usable upon demand for authorized entities
Integrity	No accidental or malicious alterations of information have been performed (even by authorized entities)

Security threats

The players

- Subject: Entity capable of issuing a request for a service as provided by objects
- Channel: The carrier of requests and replies for services offered to subjects
- Object: Entity providing services to subjects.



Security threats

The threats

Threat	Channel	Object/Server
Interruption	Preventing message transfer	Denial of service
Inspection	Reading the content of transferred messages	Reading the data contained in an object/server
Modification	Changing message content	Changing an object/server's encapsulated data
Fabrication	Inserting messages	Spoofing an object/server

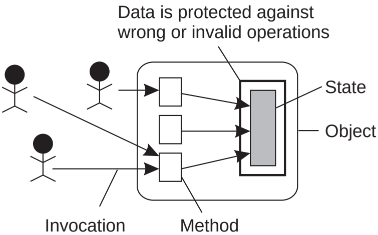
Security mechanisms

Issue

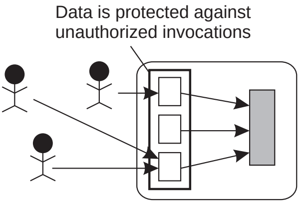
To protect against security threats, we have a number of security mechanisms at our disposal:

- Encryption: Transform data into something that an attacker cannot understand (confidentiality). It is also used to check whether something has been modified (integrity).
- Authentication: Verify the claim that a subject says it is *S*: verifying the identity of a subject. (**Who is accessing**/ requesting?)
- Authorization: Determining whether a subject is permitted to make use of certain services. (**Who is allowed** to access/request a service/)
- Auditing: Trace which subjects accessed what, and in which way. Useful only if it can help catch an attacker. (Attackers will try to avoid leaving traces)

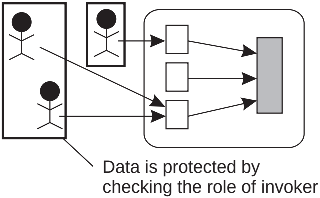
Design issue: Focus of control



(a)



(b)

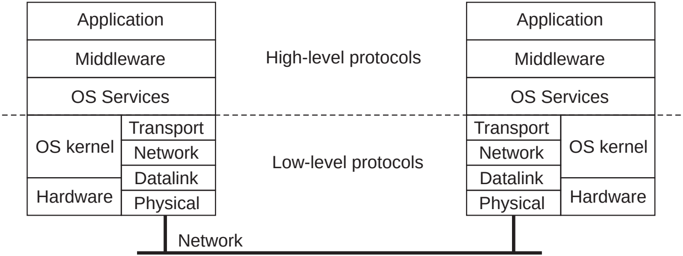


(c)

Design issue: Layering of mechanisms

Issue

At which logical level are we going to implement security mechanisms?



Design issue: Layering of mechanisms

Trusted Computing Base

Typically: security at lower layers often more convenient, BUT

Important

Whether security mechanisms are actually used is related to the trust a user has in those mechanisms. (Do you trust the network layer between your smart phone and your email server?) No trust ⇒ implement your own mechanisms (at higher levels). (Now, you need to trust SSL/TLS ...)

Fundamental Laws of Security - 1

Important

The security of any distributed system is exactly as good as its weakest component/link.

Fundamental Laws of Security - 2

© Randy Glasbergen
glasbergen.com

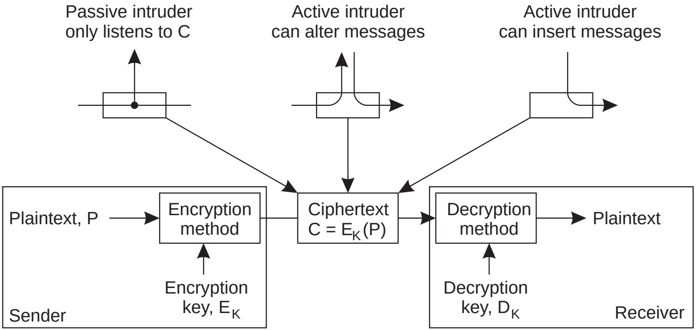
"I've done my best to make your user name and password as secure as possible...but you still move your lips when yo

Fundamental Laws of Security - 3

Observation

The security of your system needs to depend on technical and mathematical facts, and never on hidden information.

Cryptography



Cryptography

Methods

- Symmetric system: Use a single key to (1) encrypt and (2) decrypt. Requires that sender and receiver share the secret key. (e.g., DES, AES) $P = D_K(E_K(P))$
- Asymmetric system: Use different keys for encryption and decryption, of which one is private (K_A^-), and the other public (K_A^+). (e.g., RSA) $P = D_{K\,D}(E_{K\,E}(P))$
- Hashing system: Only encrypt data and produce a fixed-length digest. There is no decryption; only comparison is possible. (e.g., MD5, SHA-1)

Cryptography

Use Cases

- Symmetric system: Encryption (prevention of interception)
- Asymmetric system: Authentication (prevention of fabrication)
- Hashing system: Integrity (prevention of modification)

Cryptographic functions

Essence

Make the encryption method E public, but let the encryption as a whole be parameterized by means of a key S (Same for decryption)

- One-way function: Given some output m_{out} of E_S, it is (analytically or) computationally infeasible to find $m_{in} : E_S(m_{in}) = m_{out}$
- Example: given $E_S = Shakespeare$ and $m_{out} = MacBeth$ infeasible to find/define the environment m_{in} that let Shakespeare's mind to produce MacBeth

Cryptographic functions

Essence

- Weak collision resistance: Given the pair $\langle m, E_S(m)\rangle$, it is computationally infeasible to find an $m^* \neq m$ such that $E_S(m^*) = E_S(m)$
- Example: $m =$ *mouse is afraid of* and $E_S(m) =$ *cat* unable to find $m^* =$ *dog is afraid of* and $E_S(m^*) =$ *cat*
- Strong collision resistance: It is computationally infeasible to find any two different inputs m^* and m such that $E_S(m^*) = E_S(m)$
- Example: unable to find $m =$ *dog Rex is afraid of* and $m^* =$ *dog Struppi is afraid of* and $E_S(m^*) = E_S(m) =$ *cat*

Cryptographic functions

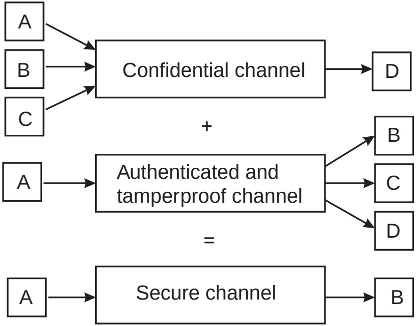
Essence (cnt'd)

- One-way key: Given an encrypted message m_{out}, message m_{in}, and encryption function E, it is analytically and computationally infeasible to find a key K such that $m_{out} = E_K(m_{in})$
- Weak key collision resistance: Given a triplet $\langle m, K, E \rangle$, it is computationally infeasible to find an $K^* \neq K$ such that $E_{K^*}(m) = E_K(m)$
- Strong key collision resistance: It is computationally infeasible to find any two different keys K and K^* such that for all m: $E_K(m^*) = E_K(m)$

Secure channels

- Authentication
- Message Integrity and confidentiality

Secure channels



What’s a secure channel

- Both parties know who is on the other side (authenticated).
- Both parties know that messages cannot be tampered with (integrity).
- Both parties know messages cannot leak away (confidentiality).

Authentication versus integrity

Important

Authentication and data integrity rely on each other: Consider an active attack by Trudy on the communication from Alice to Bob.

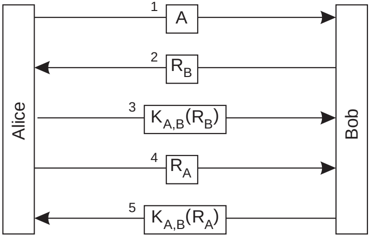
Authentication without integrity

Alice's message is authenticated, and intercepted by Trudy, who tampers with its content, but leaves the authentication part as is. Authentication has become meaningless.

Integrity without authentication

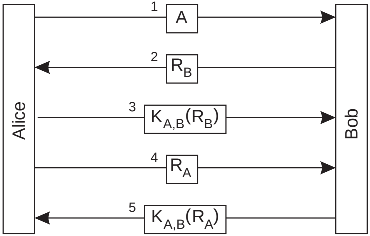
Trudy intercepts a message from Alice, and then makes Bob believe that the content was really sent by Alice. Integrity has become meaningless.

Authentication: Secret (shared) keys



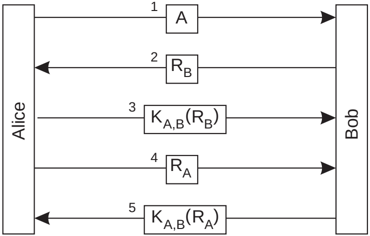
1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret (shared) keys



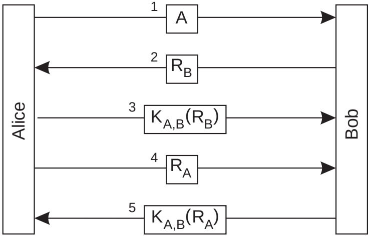
1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret (shared) keys



1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret (shared) keys



1: Alice sends ID to Bob

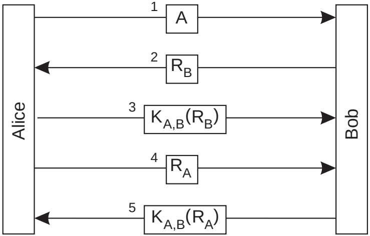
2: Bob sends challenge R_B to Alice

3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.

4: Alice sends challenge R_A to Bob

5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret (shared) keys



1: Alice sends ID to Bob

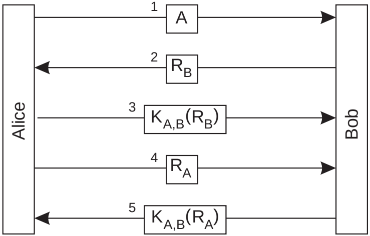
2: Bob sends challenge R_B to Alice

3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.

4: Alice sends challenge R_A to Bob

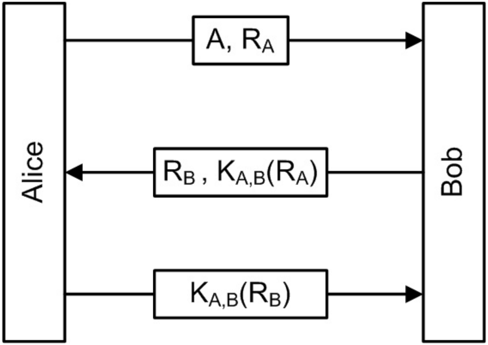
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

Authentication: Secret (shared) keys



1: Alice sends ID to Bob
2: Bob sends challenge R_B to Alice
3: Alice encrypts R_B with shared key $K_{A,B}$. Bob now knows he is talking to Alice.
4: Alice sends challenge R_A to Bob
5: Bob encrypts R_A with $K_{A,B}$. Alice now knows that she is talking to Bob.

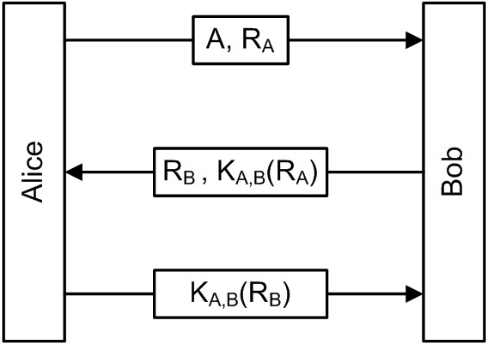
Authentication: Secret keys



Improvement

Combine steps 1&4, and 2&5. Price to pay: correctness.

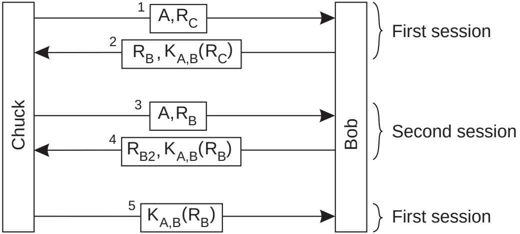
Authentication: Secret keys



Improvement

Combine steps 1&4, and 2&5. Price to pay: correctness.

Authentication: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C

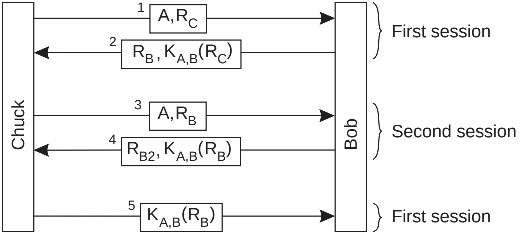
2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C

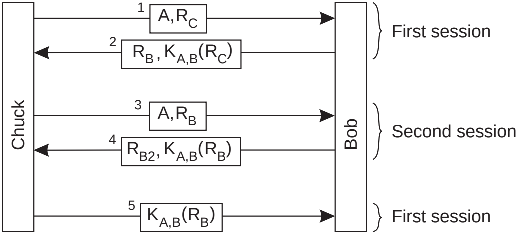
2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

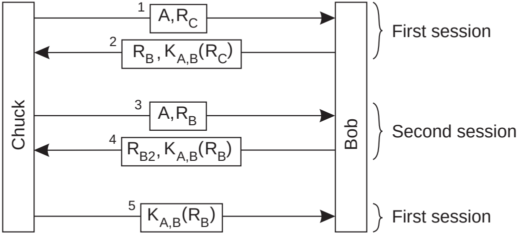
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C
2: Bob returns a challenge R_B and the encrypted R_C
3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B
4: Bob sends back a challenge, plus $K_{A,B}(R_B)$
5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C

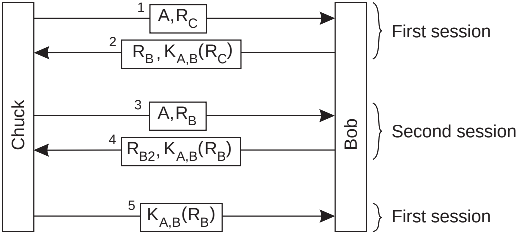
2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



1: Chuck claims he’s Alice, and sends challenge R_C

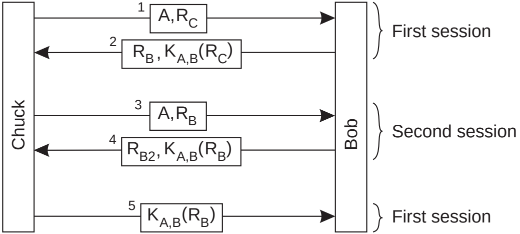
2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack



1: Chuck claims he's Alice, and sends challenge R_C

2: Bob returns a challenge R_B and the encrypted R_C

3: Chuck starts a second session, claiming he is Alice, but uses challenge R_B

4: Bob sends back a challenge, plus $K_{A,B}(R_B)$

5: Chuck sends back $K_{A,B}(R_B)$ for the first session to prove he is Alice.

Authentication: Secret keys reflection attack

Question to the audience

How can we fix the protocol?

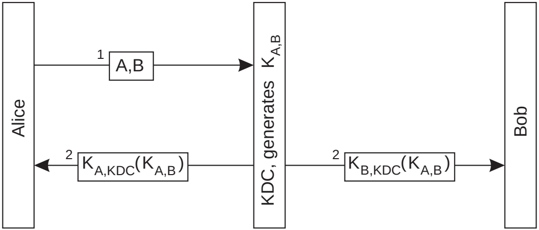
Your choices are:

- Bob remembers which challenges he used (head)
- Bob disallows a second session (nose)
- This protocol is broken, no way to fix it (ear)

Authentication: KDC

Problem

With N subjects, we need to manage $N(N-1)/2$ keys, each subject knowing $N-1$ keys $\Rightarrow$ use a trusted Key Distribution Center that generates keys when necessary.



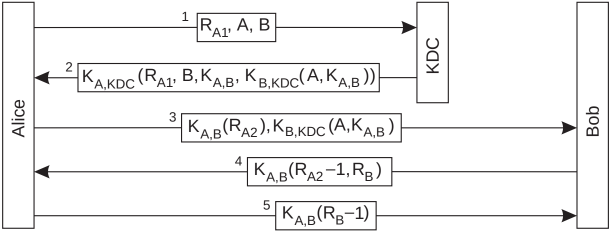
Question

How many keys do we need to manage?

Authentication: KDC (Needham-Schroeder)

Inconvenient

We need to ensure that Bob knows about $K_{A,B}$ before Alice gets in touch ⇒ let Alice do the work and pass her a ticket to set up a secure channel with Bob.



Alice

KDC

Bob

1 R_{A1}, A, B

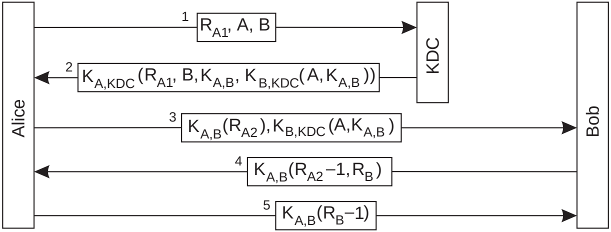
2 $K_{A,KDC}(R_{A1}, B, K_{A,B}, K_{B,KDC}(A, K_{A,B}))$

3 $K_{A,B}(R_{A2}), K_{B,KDC}(A, K_{A,B})$

4 $K_{A,B}(R_{A2}-1, R_B)$

5 $K_{A,B}(R_B-1)$

Needham-Schroeder: Subtleties



Some issues

Q1: Why does the KDC put *B*ob into its reply message, and *A*lice into the ticket?

Q2: The ticket sent back to Alice by the KDC is encrypted with Alice's key. Is this necessary?

Needham-Schroeder: Subtleties

Security flaw

Suppose Trudy finds out Alice's key ⇒ she can use that key anytime to impersonate Alice, even if Alice changes her private key at the KDC.

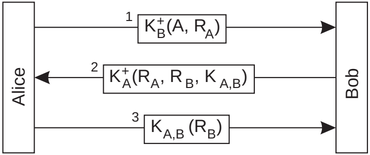
Reasoning

Once Trudy finds out Alice's key, she can use it to decrypt a (possibly old) ticket for a session with Bob, and convince Bob to talk to her using the old session key.

Solution

Have Alice get an encrypted number from Bob first, and put that number in the ticket provided by the KDC ⇒ we're now ensuring that every session is known at the KDC.

Authentication: Public key

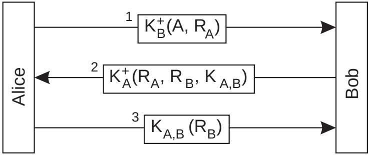


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.

2: Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K_A^+.

3: Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

Authentication: Public key

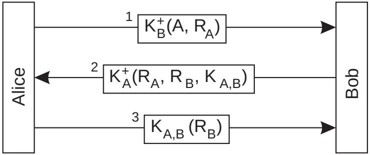


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.

2: Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K_A^+.

3: Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

Authentication: Public key

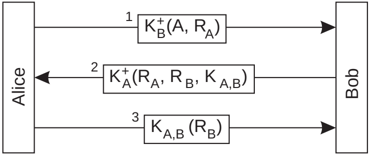


1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.

2: Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K_A^+.

3: Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

Authentication: Public key



1: Alice sends a challenge R_A to Bob, encrypted with Bob's public key K_B^+.
2: Bob decrypts the message, generates a secret key $K_{A,B}$ (session key), proves he's Bob (by sending R_A back), and sends a challenge R_B to Alice. Everything's encrypted with Alice's public key K_A^+.
3: Alice proves she's Alice by sending back the decrypted challenge, encrypted with generated secret key $K_{A,B}$

Confidentiality

Solutions

Secret key: Use a shared secret key to encrypt and decrypt all messages sent between Alice and Bob

Public key: If Alice sends a message *m* to Bob, she encrypts it with Bob's public key: $K_B^+(m)$

Problems with keys

- Keys wear out: The more data is encrypted by a single key, the easier it becomes to find that key $\Rightarrow$ don't use keys too often
- Danger of replay: Using the same key for different communication sessions, permits old messages to be inserted in the current session $\Rightarrow$ don't use keys for different sessions

Confidentiality

Problems with keys

- Compromised keys: If a key is compromised, you can never use it again. Really bad if *all* communication between Alice and Bob is based on the same key over and over again ⇒ don't use the same key for different things.
- Temporary keys: Untrusted components may play along perhaps just once, but you would never want them to have knowledge about your really good key for all times ⇒ make keys disposable

Confidentiality

Essence

Don’t use valuable and expensive keys for all communication, but only for authentication purposes.

Consequence

Introduce a “cheap” session key that is used only during one single conversation or connection (“cheap” also means efficient in encryption and decryption: in RSA 100x-1000x slower than DES).

Digital signatures

Scenario

Alice sells her iPhone5 to Bob for 500 EUR

- Bob wants to ensure, it's indeed Alice selling the item (and vice versa)
- Bob wants to ensure, that Alice cannot later claim a higher price
- Alice wants to ensure that Bob cannot later claim a lower price

Digital signatures

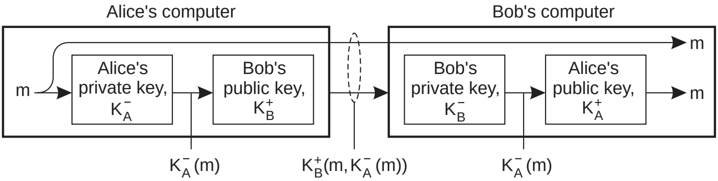
Harder requirements

- Authentication: Receiver can verify the claimed identity of the sender
- Nonrepudiation: The sender can later not deny that he/she sent the message
- Integrity: The message cannot be maliciously altered during, or after receipt

Solution

Let a sender sign all transmitted messages, in such a way that (1) the signature can be verified and (2) message and signature are uniquely associated

Public key signatures



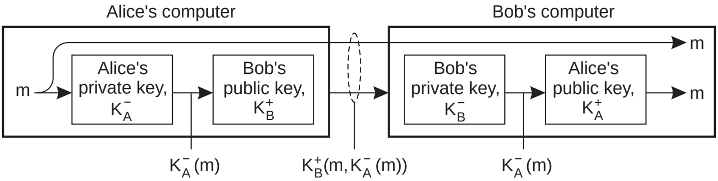
1: Alice encrypts her message m with her private key K_A^- $\Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



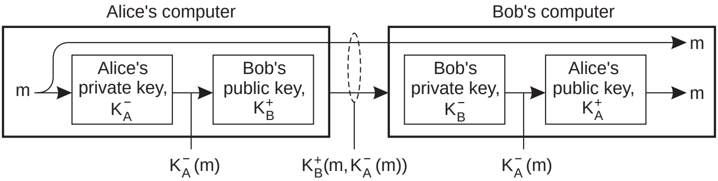
1: Alice encrypts her message *m* with her private key K_A^- $\Rightarrow m' = K_A^-(m)$

2: She then encrypts *m'* with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends *m''* to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read *m*, nor *m'* during their transmission.

4: Bob decrypts *m'* with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



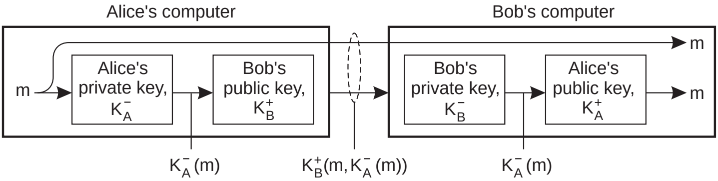
1: Alice encrypts her message m with her private key K_A^-
$\Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



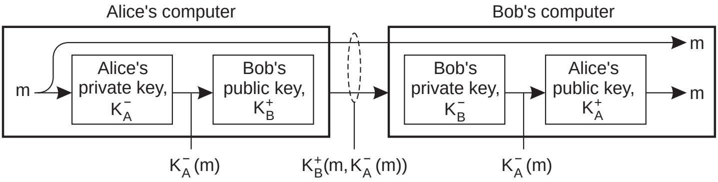
1: Alice encrypts her message m with her private key K_A^-
$\Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Public key signatures



1: Alice encrypts her message m with her private key K_A^- $\Rightarrow m' = K_A^-(m)$

2: She then encrypts m' with Bob's public key, along with the original message $m \Rightarrow m'' = K_B^+(m, K_A^-(m))$, and sends m'' to Bob.

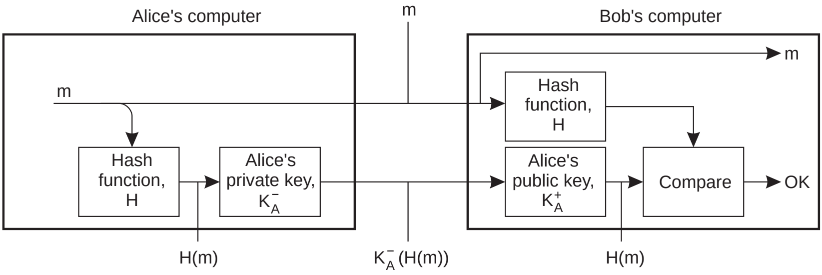
3: Bob decrypts the incoming message with his private key K_B^-. We know for sure that no one else has been able to read m, nor m' during their transmission.

4: Bob decrypts m' with Alice's public key K_A^+. Bob now knows the message came from Alice.

Message digests

Basic idea

Don’t mix authentication and secrecy. Instead, it should also be possible to send a message in the clear, but have it signed as well ⇒ take a message digest, and sign that.



Security management

- Key establishment and distribution
- Authorization management

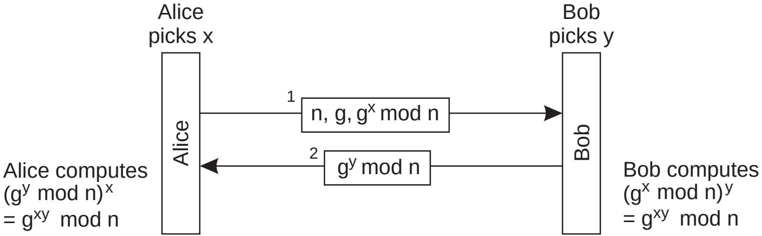
Key establishment: Diffie-Hellman

Observation

We can construct secret keys in a safe way without having to trust a third party (i.e. a KDC):

- Alice and Bob have to agree on two large numbers, n (prime) and g. Both numbers may be public.
- Alice chooses large number x, and keeps it to herself. Bob does the same, say y.

Key establishment: Diffie-Hellman

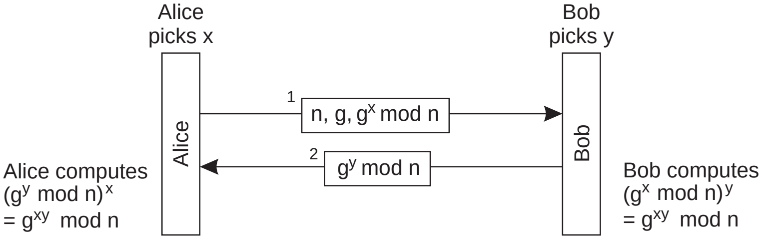


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime > 160 bits.
In practice, $n, g > 512$ bits.

Key establishment: Diffie-Hellman



1: Alice sends $(n, g, g^x \bmod n)$ to Bob

2: Bob sends $(g^y \bmod n)$ to Alice

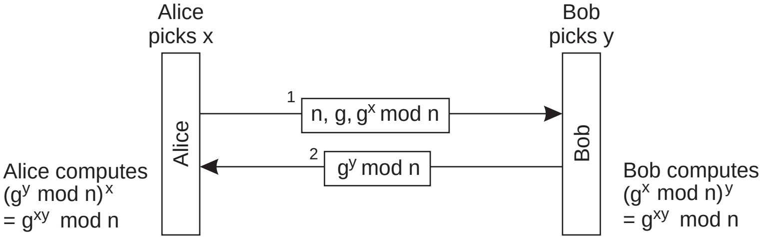
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$

4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime > 160 bits.
In practice, $n, g > 512$ bits.

Key establishment: Diffie-Hellman



1: Alice sends $(n, g, g^x \bmod n)$ to Bob

2: Bob sends $(g^y \bmod n)$ to Alice

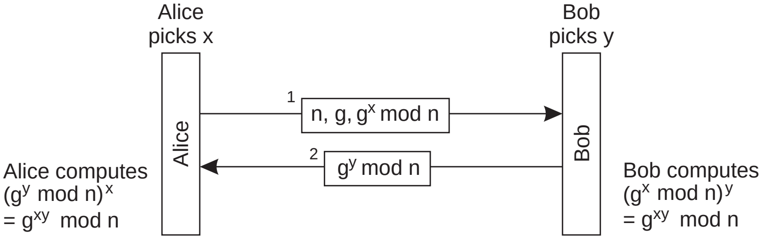
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$

4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime > 160 bits.
In practice, $n, g > 512$ bits.

Key establishment: Diffie-Hellman

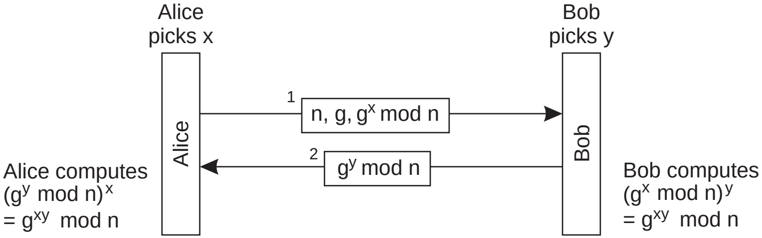


1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime > 160 bits.
In practice, $n, g > 512$ bits.

Key establishment: Diffie-Hellman



1: Alice sends $(n, g, g^x \bmod n)$ to Bob
2: Bob sends $(g^y \bmod n)$ to Alice
3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$
4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

Note

$n = kq + 1$, with q being prime > 160 bits.
In practice, $n, g > 512$ bits.

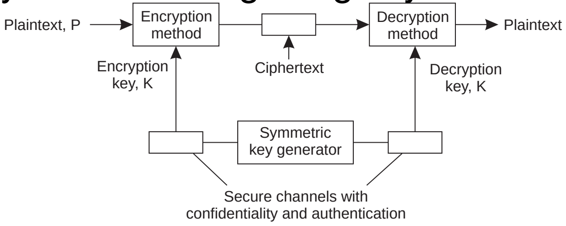
Key distribution

Essence

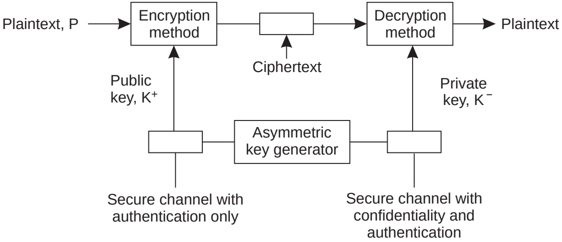
Authentication requires cryptographic protocols, ⇒ require session keys to establish secure channels, who's responsible for handing out keys?

- Secret keys
  - Create your own and exchange it out of band
  - Trust a key distribution center (KDC) and ask it for a key.
- Public keys: How to guarantee that A's public key is actually from A?
  - Personally exchanged out of band
  - Use a trusted certification authority (CA) to hand out public keys. A public key is put in a certificate, signed by a CA.
- Trust Hierarchy: work your way up the hierarchy to increase your confidence

Key distribution: getting keys to owners



(a)



(b)

Certificate Lifetime

Essence

Lifelong certificates would be nice but need revocation when compromised

- Certificate Revocation Lists: CRL regularly published by CA
- Expiration Time: limit lifetime, invalid after expiration time (extreme case reduce to zero) and couple with CRL
- In Practise: certificates with limited lifetime, users hardly check CRLs, but some software installers do

Some Common Attack Scenarios

Distributed systems security can be compromised on any layer

Thus remember: any security breach potentially renders the entire system insecure.

Just a small set of examples

Following attacks happen in practice all the time

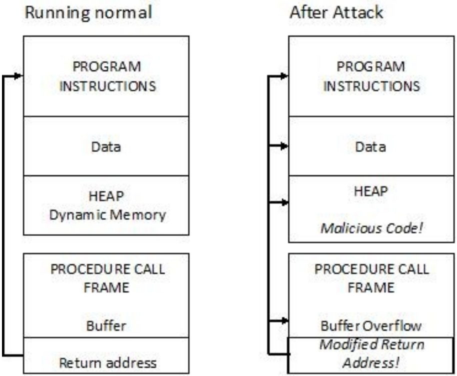
- Buffer Overflows
- SQL Injection Attach
- Cross-Side Scripting Attach (XSS)
- Distributed Denial-of-Service Attack (DDoS)
- Sidechannel Attacks
- Social Engineering

Buffer Overflows

Common security problem in unmanaged programming languages (e.g., C / C++)

- Input data larger than reserved heap space
- Hence data flows over into next frame, allowing an attacker to overwrite the return address pointer of a procedure call with a custom address
- Hence allowing the attacker to execute arbitrary code

Buffer Overflow



Attacker plants code that overflows buffer and corrupts the return address. Instead of returning to the appropriate calling procedure, the modified return address returns control to malicious code, located elsewhere in process memory.

Source

`http://cis1.towson.edu/~cssecinj/modules/cs2/buffer-overflow-cs2-c/`

SQL Injection Attack

Observation

Some web applications do not sufficiently check data received from users before issuing SQL queries

```
select * from users where user = $username
                          and pw = md5($pw)
```

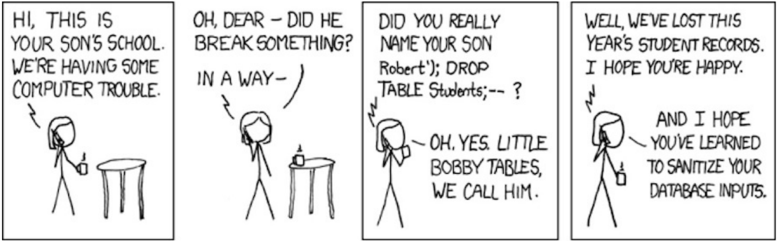
now assume following input:

```
$username = '1 or 1=1; drop table users; --'
```

and you get:

```
select * from users where user = 1 or 1=1;
                          drop table users; --
```

SQL Injection Attack



Cross-Side Scripting Attack (XSS)

Observation

Some web applications do not sufficiently check data received from users

- Similar principle to SQL injection
- Allows attacker to inject arbitrary scripts into a legit (trustable) web site
- Example: blog with commentary function that accepts arbitrary HTML code

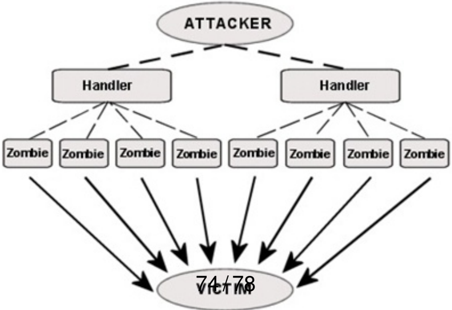
```
Very interesting article!

<script type = ”text/javascript”>
<!-- window.location=”http://62.178.71.105”;
-->
</script>
```

Distributed Denial-of-Service Attack (DDoS)

Observation

Attacker uses a network of hacked machines

- Bots/Zombies overload the resources of the target with requests
- Difficult to protect against (needs to be done at ISP level)
- Difficult to identify the attacker (all request come from unassuming zombies)



Sidechannel Attacks & Social Engineering

Ignore the technical security mechanisms

finding out the secret that the mechanism was based on

- Phishing for passwords or keys
- NSA demanding private keys from certification authorities
- Reverse-engineering keys in embedded devices by measuring energy comsumption

Social Engineering

Sidechannel attacks on humans behind the ”secure” technical system

- usual assumption: people are easily manipulated
- e.g.: incoming call ”Hey, I’m from IT. We have a problem with your account here”

Further Lectures on Security

- [183.367] Security for Systems Engineering
- [183.645] Advanced Security for Systems Engineering
- [183.633] IT Security in Large IT Infrastructures
- [188.312] Organizational Aspects of IT Security
- [183.606] Seminar aus Security

Organizational Issues

Next/Last Lecture

on Monday, Nov 10.
No lecture next week!

Lecture Feedback

on TISS: between 24.11.2014 and 12.02.2015

Looking for a Bachelor Thesis topic?

Check out some topics on my webpage

`http://christophdorn.wordpress.com/`
`stuff-for-students`

Not related to consistency, replication, or security algorithms, but very implementation-centric

- Software Architecture
- Self-Adaptive Systems
- Collaboration/Coordination Systems
- Context-aware Systems