User Authentication Principles and Methods

David Groep, NIKHEF

Principles and Methods

- Authorization factors
- Cryptographic methods
- Authentication for login
- How secure is security?

Authentication

Establishing the identity of your partner

credential persistence	Authentication Factors *what you know, what you have, what you are*		
	0	1	2
none	web browser, coffee machine, …	`login(1)`, ssh without key agent, Girotel's GIN	DigiPass (Rabobank), SecurID, PKI, CryptoCard (sec), Schiphol's Privium
long(er) time	(DNS cache)	ssh key-agent, Kerberos TGT GSI (Grid Security Infrastructure)*	Kerberos+CryptoCard, GSI (Grid Security Infrastructure)*

Ingredients for $\geq$1 factor Auth

- Cryptography
  - symmetric
  - asymmatric

- Trust
  - user-to-system
  - system-to-system
  - system-to-user

Keeping it private: cryptography

- symmetric crypto:
  - common key is used to encrypt *and* decrypt
  - key must be exchanged over a pre-existing private channel
  - arbitrarily complex methods (XOR, 3DES, IDEA, …)
- asymmetric “public key” crypto:
  - a *key-pair* has encryption and decryption key
  - keys cannot be derived from each other
  - one key can be broadcasted publicly
  - popular methods: RSA, DSA

Symmetric crypto

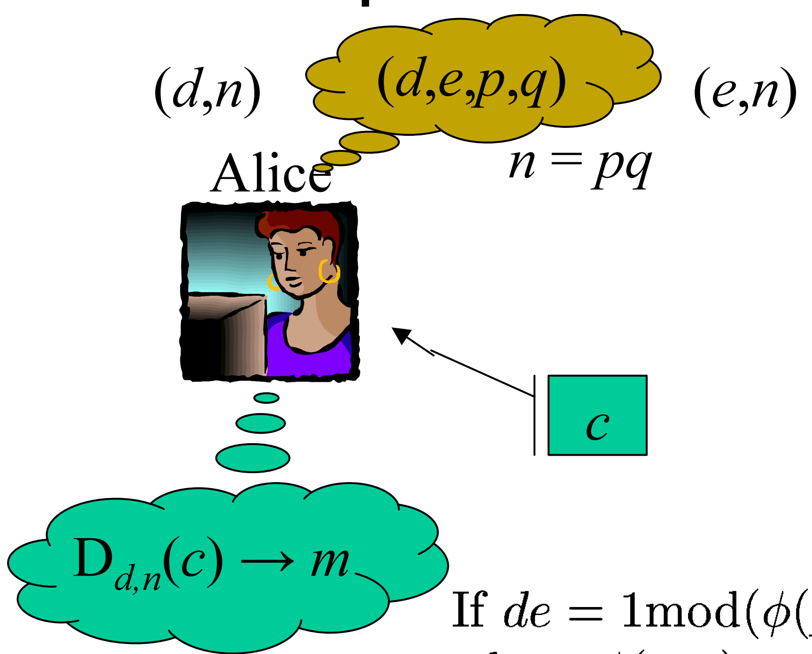
- Exchanging the key is main problem
- Many algorithms, from worthless to pretty good
  (Caesar's, XOR, Enigma, DES, 3DES, IDEA, CAST5)

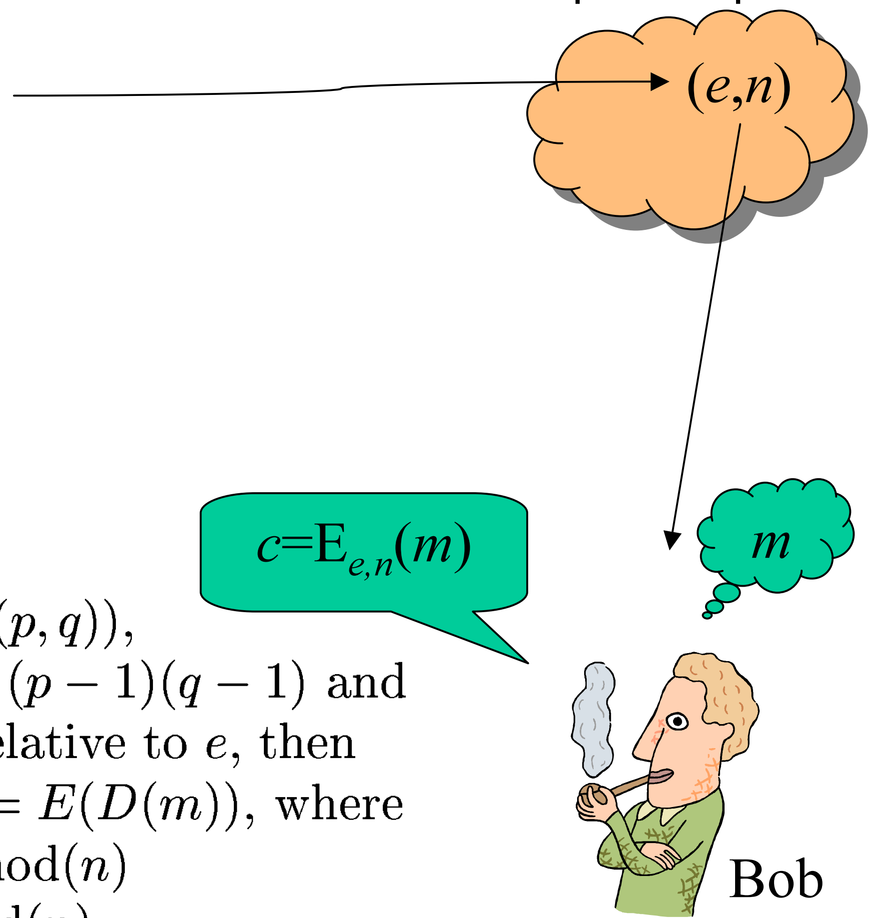
- Examples:
  - XOR: key=0x56, plaintext=45:

```
01010110 = 0x56  (key)
00101101 = 45    (plain text)
--------
01111011 = 123   (encrypted message)
01010110 = 0x56  (same key)
--------
00101101 => 45
```

Public Key crypto: how?

- Example:





If $de = 1 \bmod (\phi(p,q))$,
where $\phi(p,q) = (p-1)(q-1)$ and
$(p-1)$ prime relative to e, then
$m = D(E(m)) = E(D(m))$, where
$E_{e,n}(m) = m^e \bmod (n)$
$D_{d,n}(c) = c^d \bmod (n)$

RSA key generation

- Take a (small) value *e* = **3**
- Generate a set of primes (*p*,*q*),
  each with a length of *k*/2 bits, with (*p*-1) prime relative to *e*.
  (*p*,*q*) = **(11,5)**
- $\phi(p,q)$ = (11-1)(5-1) = **40**; *n*=*pq*=**55**
- find *d*, in this case **27** [3*27 = 81 = 1 mod(40)]

- Public Key: **(3,55)**
- Private Key: **(27,55)**

If $de = 1 \text{mod}(\phi(p,q))$,
where $\phi(p,q) = (p-1)(q-1)$ and
$(p-1)$ prime relative to e, then
$m = D(E(m)) = E(D(m))$, where
$E_{e,n}(m) = m^e \text{mod}(n)$
$D_{d,n}(c) = c^d \text{mod}(n)$

An RSA message exchange

(3,55)

Encryption:

- Bob thinks of a plaintext $m(<n)$ = **18**
- Encrypt with Alice's public key **(3,55)**
- $c=E_{3;55}(18)=18^3 \bmod(55) = 5832 \bmod(55) =$ **2**
- send message **"2"**

If $de = 1 \bmod(\phi(p,q))$,
where $\phi(p,q) = (p-1)(q-1)$ and
$(p-1)$ prime relative to e, then
$m = D(E(m)) = E(D(m))$, where
$E_{e,n}(m) = m^e \bmod(n)$
$D_{d,n}(c) = c^d \bmod(n)$

Decryption:

- Alice gets **"2"**
- she knows private key **(27,55)**
- $E_{27;55}(2) = 2^{27} \bmod(55) =$ **18** !

- If you just have (3,55), it's hard to get the 27…

Uses of public-key crypto

- Confidentiality
  no-one but the recipient can read what you say
- Message integrity
  encrypt a digest of your message with a private key
- Non-repudiation
  similar to integrity
- This encryption works both ways with 2 key pairs

From public-key crypto to trust

- You establish communication between key pairs but not between entities!
- Binding needed between key pair and an identity (*this is implicit in symmetric solutions, but not here!*)
- in a trusted way …
- Anarchic models (SSH)
- Distributed trust models (PGP)
- Hierarchical (authoritarian) model (PKI)

Methods (1): login/telnet -style

- Only one factor, a password in the user's memory
- Password must be kept secret
  - should not be sent in clear over networks
  - user must not write it down in clear
  - should not be guessable

  problems with all of the above…

Methods (2): ssh with passwords

- still only one factor: the password
- but each SSH daemon as a RSA* key pair:
  - public key is sent to the client
  - this is used to encrypt a (symmetric) **session key**
  - password and future data are sent within the encrypted session

Methods (2): ssh with passwords

- Problems with SSH password authentication:
  - key distribution problem
    - how can the client verify that the host public key is correct?
    - only trivial alerts against *change* of host key
  - no single sign-on
    (login to a new host requires typing the password)
  - leads to guessable passwords or writing them down!

Methods (3): ssh with client keys

- Have the client generate an RSA key pair locally:
  ssh-keygen → ~/.ssh/id_rsa & ~/.ssh/id_rsa.pub
- The public part of this key is stored on remote server in user homedir:
  ~remoteuser/.ssh/authorized_keys2
- ssh remoteuser@remotehost
  challenge encrypted with public key sent to user; can he decrypt it?
- same keypair can be used for all hosts

Methods (3): ssh with client keys

- The (local) user keypair is a very valuable target!
- Need to (symmetrically) encrypt the private key (~/.ssh/id_rsa)

- to get single sign-on:
  - in-memory proxy agent can serve the private key to new clients (ssh-agent, ssh-add ~/.ssh/id_rsa)
  - protected with unix file privileges on socket
  - contact information in environment variables

- Key distribution problem is still there…

Methods (4): Kerberos

- Based on symmetric cryptography
- One Key Distribution Centre (KDC) per `Realm'
  - Authentication Service (AS) and
  - Ticket Granting Service (TGS)
- KDC supplies limited-lifetime "tickets" to principals
  - Ticket Granting Ticket, encrypted with hash of password
  - Service Tickets (ST), verified using the TGT
- Every service also shares a secret with the KDC
  (kadmin: add_principal host/satan.hell.org@HELL.ORG)

Methods (4): Kerberos

- User contacts KDC and gets TGT, encrypted using 3DES with hash of password as key

- TGT used to encrypt session where ST is requested from KDC
- user gets ticket only when authorized by the KDC AS
- ST encrypted with password of service's principal
- If service can decrypt ticket, it can be used to exchange new session key

- KDC has copy of every principal's password!
- Has active role, thus central point of failure

Methods (5): PKI

- Public Key Infrastructure, PKI, aims to solve the key distribution problem for public key crypto

- Trusted third party (Certification Authority) binds *authentication data* to a *public key*:
  the Certificate

Methods (5): PKI

- The PKI Certificate `X.509'
  - structured message with:
  - public key
  - identifier(s)
  - digitally signed by a trusted third party

- Certification Authority (CA)
  - binds identifiers to a public key
  - in accordance with a defined Certification Policy
  - following the guidelines of a C. Practice Statement

Methods (5): PKI

- Certificate used without interaction with CA
- Life-time: usually 1 year

  (should depend on RSA key length)

- Used in TLS protocols (formerly SSL)
- Public key encrypts a (symmetric) session key
- Can be used both ways (client authentication)

- Also for message security

- Applications: https, S/MIME.
- Popular CA's: Verisign, Thawte

Methods (5): PKI

- Problems with PKI
  - public keys for trusted CA's need to be distributed
  - difficult to invalidate credentials (`revocation')
  - need to protect private key with passphrase,
    no implicit single sign-on; key may still be on disk…
  - CA is accountable for the binding he makes
  - heavy registration procedure (RA's, etc.)
  - site admins risk doing double work
    when working with user certs for sensitive work/login

Methods (6): GSI

- Grid Security Infrastructure (GSI), based on PKI
- user generates limited `proxies' of long-living credential

- proxy secured by regular unix file permissions
- life-time usually 12 hours
- possible to limit capabilities (`only read e-mail')
- proxy signed by long-lived key, that is signed by CA
- proxy implements single sign-on
- other PKI problems remain: key on disk, double work, heavy CA

- applications: grid job run, file access, gsi-ssh

One-time pads

- Adds extra factor to authentication (cryptocard)
- Cryptocard serves as password generator
  but needs activation data (PIN code)
- Clock syncronization cryptocard & host/server
- can be used over any channel (login, telnet, ssh, …)

Hardware tokens

- Store precious credentials on detached active storage

- Examples:
  - SecurID*: small processor on-board decrypts challenge with the built-in private key, key never leaves card (RSA or symmetric key)
  - Chipknip (3DES symmetric key)

Summary of Methods

1. Login, Telnet
2. ssh with password authentication
3. ssh with RSA authentication
4. Kerberos
5. PKI
6. GSI

- Additional bonus options for (almost) all:
  - one-time pads
  - hardware tokens

Conclusions

- Plenty of options,
  - from weak to strong,
  - for harmless stuff and for military-grade secrets

- No silver bullet
  - Security is about reducing risk, not eliminating risk
  - Users are oblivious to security:
    if it's too difficult, they will:
    - write their password on the wyteboard
    - type their password in plain text in scripts to renew credentials
    - install their own back-doors
    - ……