Information Security - part2

67404 characters in 10165 words on 1695 lines

Florian Moser

August 8, 2018

1 introduction

1.1 motivation

basic infrastructure based on networked systems
functionality vs security tradeoff

1.2 basic definitions

computer security
prevent, detect improper actions of users
improper, proper actions defined relative to security

information security
more general than computer security
proper use of information, independent of computer system
information /= data (can publish report without sources)

side-channels attack
secure cryptography breaks due to faulty real world application
faults in the implementation, information leaks (sound, power usage)

1.3 security as policy compliance

target is to build a system that meets specification within environment
the approach taken by software engineering, goal oriented view of security
for security, define (im)proper actions
assumed environment are users wanting to use (not break) the system
assumed adversary may networks, but no physical/side-channel access

1.3.1 policy compliance overview

specification (what to do, security policy)
implementation (how is it done, security mechanism)
correctness (does it work, compliance)

formal
secure if S||E |= P for system S, environment E, policy P
E difficult to define (need to know potential attackers)
S difficult to define (needs to include libraries, hardware, OS)

1.3.2 security policies

also called security goals

CIA (traditional)
confidentiality (no improper disclosure of information)
integrity (no modification of information)
availability (no impairment of service/functionality)

more
secrecy (no disclosure of information)
authenticity (who formulated message, more specific than integrity)
non-repudiation (accountability for actions)
plausible deniability (weak form of secrecy, contrary to non-repudiation)
auditability (integrity of whole system)

privacy
anonymity (secrecy of principal identities, communication relationships)
data protection (personal data used only in certain ways)

access control
protection of system resources against unauthorized access
enforcing the security policy by regulating use of system

data protection
access control enforces conditions in present & past (provisions)
usage control enforces conditions in the future (obligations)

1.3.3 define security

functional correctness, reliability, security may overlap

properties
high-level description of system behaviours

policies
conjunctions of different properties
or low-level & operational (”when choosing pw it must be long”)

mechanisms
concrete system components used to archive policy compliance

1.3.4 example policies

bank
authenticity of clients
integrity of accounts, user data
secrecy of customer data
non-repudiation of transactions
availability of logging

e-voting
only registered voters can vote
each voter can vote only once
integrity of votes
privacy of voting information
individual (+universal) verifiability
availability of system during voting period

1.3.5 IBM secure processor

cryptographic coprocessor with secure memory
firmware & hardware from IBM, software from customer

layers
each layer has owner
each layer/owner has keypair, stored in persistent storage
higher level can't access lower level, physically secured
owner can use private key to authenticate commands for his layer i
owner can use private key to assign ownership of i+1
IBM acts as root of trust

properties
outbound authentication (distinguish message from layer i from others)
inbound authentication (if A owner of layer then only A can command)
access to secrets (layer i secrets only accessed by trusted parties)

1.3.6 security types

unilateral security
security properties given to single system
system is secure if attacker can't deviate it from normal behaviour

multilateral security
protect participants from each other, may need to employ 3rd party
SW vendor & customer both protect their IP by executing at 3rd party

1.4 security as risk minimization

target is to minimize risk of abuse while keeping system running
the approach taken by administrators
avoid, minimize, accept or transfer risk

1.4.1 risk analysis

focus on risk from vulnerabilities and their exploitation

owners
they have valuable assets, which are under threat
impose countermeasures to reduce risk & vulnerabilities

threat agents
threats from employees (unintentional or on purpose)
hackers, criminals, terrorists, foreign espionage agents
information-warfare operations to disrupt weapon, command structures

threats
confidential data intercepted
integrity harmed by modification, fabrication
availability stopped by interruption of service

vulnerabilities

interaction in unknown, hostile environment
vulnerable infrastructure, too short time to market, monocultures

how it fits together
threat agents abuse threats which exploit vulnerabilities
threat agents abuse or damage assets
weakness can be exploited by threats to cause damage

reduction steps
identify assets to protect
identify risks to those assets, risk = (chance ∗ impact) of abuse
deduce how well countermeasures reduce risk and their tradeoffs
employ most effective countermeasures (intrusion detection vs prevention)

1.4.2 apply risk analysis

iterate risk analysis & employing of countermeasures
use standards such as ISO 27000

email reduction
mail content, sender/receiver link, availability
others reading, tampering mail, observe communication patterns
propose PGP
PGP reduces risk dramatically
but key exchange timeconsuming, limited support, users need education

face scanning reduction
air travellers & those on the ground
terrorists will board planes
propose face scanning
detects some terrorists
but less privacy, false positives, false sense of security

1.4.3 examples

web server
bugs in server, php, modules
compromise often because systems not maintained or misconfigured

authentication vulnerabilities
undetected exploitation with leaked password
passwords often poor or non-existent
poorly secured administration-level accounts
weak hashing algorithms stores with weak security

open-SSL
cryptographic support library, used by TLS, POP, IMAP, ...
multiple vulnerabilities exposed, remote code execution (even as root)

1.5 core mechanisms

DH
choose generator g, use prime p
compute X with secret s ⇒ g^s mod p
compute Y with secret s2 ⇒ g^s2 mod p
prevent MitM by authenticating public keys (but preserve PFS)
prevent DoS (repeating Y computation) with proof of work

key formats
pre-shared symmetric keys
asymmetric key-pair for en/decryption or sign/verify

2 network security

network context from security perspective (layers, protocols, internet)
options, tradeoffs of securing systems in practice

2.1 computer networks

physical (links, bit transfer) like wire, wireless
abstract (communication medium between principal, secure channel)

2.2 layered communication

communication divided in layers (separation of concerns)
i-th layer communicates with its peer on same level
uses only services of lower layer

TCP/IP protocol reference model
application (telnet, FTP, HTTP)
transport (TCP, UDP) for end-to-end transport
network (IP) for the internet
link (IEEE 802.x) for single-link transport

ISO reference model
additionally presentation, physical, session layers

encapsulation
header/trailers added to packets on stack traversal
encryption, decryption, transformation of lower levels

internet protocol (IP)
delivers data across network
headers specify source, target address
compute next hop from destination address & forward (best-effort)

transmission control protocol (TCP)
reliable transport over network (all or nothing)
all packets delivered without loss, duplication, reordering

2.3 internet

confederation of networks used for TCP/IP
no global domain of trust (subnetworks, long paths, spoofing, faking)
TCP/IP does not provide authenticity or confidentiality (spoofing)

how to secure communication
where (end-to-end, hop-to-hop, ...)
which layer (link, network, transport, application)

examples for secure communication
PGP (end-to-end, at application layer)
SSL/TLS (end-to-end, implementation below application layer)
WPA (encrypts first hop, hop layer) for confidentiality, access control
IPsec VPN (client to gateway) to protect internet traversal

2.4 application managed security

enterprise security policy integrated in applications
each app responsible for secure communication, authentication, logging, ...

end-to-end principle
complete, correct implementation only possible in application layer
target application has specific requirements, can support implementation
trust-to-trust (implement functionality between trusted entities)
brain-to-brain (desired end state)

reliable file transfer example
possible errors from disk, memory, routers, ftp software, checksums limited
stronger guarantee if computed end-to-end
no trust/change/configure/manage of intermediate nodes/layers/protocols
security decision based on userID, data, etc

more examples
mail encryption / authentication PGP, S/MIME
specialized client/server systems like eVoting

assumes correct mechanisms
design (crypto done right, no attacks from lower levels)
implementation (no buffer overflows, injections)

assumes mature users
understand & respect enterprise security policy
properly configure mechanisms, client credentials
immune to social engineering

problems
3rd party, legacy applications can't keep up with policy
midbox still needed (incident response)
interference with midbox (encrypted mail hinders spam/virus scanner)
poor fit with centrally managed & enforced security

2.5 network managed security

use SSL, VPN

implementation of IP stack
transport layer & below (like IPsec) implemented in OS
above implemented in applications in user process

advantages
implement security solution once
don't bother users with security (lack of knowledge, interest)
secure legacy, standard applications without own security
central management of security at midboxes

examples
SSL (OS unchanged, but no client side authentication)
IPsec (transport security, but only IP address authentication)
WPA (secures most vulnerable link, but no others)

2.6 protecting networks

secure using different approaches, general protection often too coarse

2.6.1 concepts

defence in depth
redundancy, secure each layers independently
employ technical, physical, administrative measures
but difficult to administrate, different policies

keep it simple
easier to implement, administrate, configure
similar to system hardening

system hardening
only needed services running
good engineering (code review), maintenance (install patches)

trust perimeter
boundary between trusted, untrusted machines
multiple layers of trust possible with their own boundaries
control everything that passes perimeter (with firewall)
but maybe ill-defined (inside attackers, machine not trustworthy)
usability vs security tradeoff

2.6.2 firewalls

don't allow adversaries on machine

packet filters
router with simple access control
based on packet content, sourceId, port, ...
enforce preconfigured policies like "block FTP"

application-layer proxies
wrapper around application, control io/access
may enforce additional authentication
can inspect content of traffic, filter undesirable data, ...

use multiple
but complex to setup, maintain, false sense of security
does not stop inside attacks, malware, etc

secure web server
prime target for attack, sacrificial machine (wipe if hacked)
put firewalls between server & outside & inside (at both ends)

pro
scales better than host security, secures legacy systems
central place to setup secure policy

contra
bad at surfing, emails, https, perimeter not well defined (cloud)

2.6.3 minimize risk of exploits

good security engineering practices (patch vulnerabilities, maintain)
access control & compartmentalization

keep backups
multiple physically separate locations
useful for integrity, availability but not confidentiality

policies & support for incidents
detections & response
plan for worst case (attacker in network for years)

2.6.4 detect intruders

monitor, analyze networks for possible incidents

intrusion detection systems (IDS)
host-based (identify break-ins), network-based (DoS)
combine multiples (but high false positive rate)
signature based (known threads, repeated login attempts)
anomaly based (ML to identify abnormal behaviour)

IDS drawbacks
limited effectiveness (black box, unknown if attacks well handled)
lots of false positives (costly to handle, responsibles stop careing)
high live-cycle costs (training, maintenance)
arms race (powerful attacker can subvert system)

3 security protocols

3.1 motivation

omnipresent & critical
authentication (bank card, single sign on)
secure communication (SSL, SSH, IPsec)
special purpose (evoting, car-key systems)

in general
use cryptographic primitives for security objectives
"security protocols are three line programs people manage to get wrong"

bad samples
A → B {{message}_KsA}_KB → but B can now resend message of A
A → B {{message}_KB}_KsA → but C can now resend message of A

goals
understand arising of problems
be precise (goals, execution correctness of protocol)
examine protocols & understand strengths & weakness

3.2 definitions

3.2.1 protocols

set of rules describing how messages are exchanged (distributed algorithm)
in practice, informal/formal description with diagrams, data types

security protocol
also called cryptographic protocol
uses cryptographic mechanisms to archive security objectives

3.2.2 attacker models

usually knows protocol but cannot break crypto
can be passive but overhears all communication
or active and can intercept & generate messages
or insider impersonating a role (running the protocol)

dolev-yao attacker
the standard symbolic attacker model, strong model
read, intercept & create all messages
decompose message in parts
may compromise some agents and learn keys
but needs inverse keys for decryption (assume crypto secure)
communication between honest agents should still archives objectives
formalization like N1 ∈ (IK U M) ⇒ N1 ∈ has(IK U M)

3.3 protocol properties

prefix with "if honest A completed run with honest B, then"

aliveness
B has run the protocol in the past
prove with signature

weak agreement
B has run the protocol believing to communicate with A
prove with signature including A

non-injective agreement
B has run the protocol believing to communicate with A
A,B agree on all values exchanged
prove with signature including A & all other values

(injective) agreement
B has run the protocol believing to communicate with A
A,B agree on all values exchanged
each run of A corresponds to a unique run of B
prove with signature including A, nonce & other values

recent dimension
if property only established if both run has been recent
then can prefix property with "recent" (like "recent aliveness")
prove with nonce

3.4 protocol claims

prefix with "if honest A completed run with honest B, and claimed *** on some value, then"

secrecy
specific value is not learned by attacker
ensures data only available to those authorized
key authentication if data is a key
key confirmation if assurance that other possesses key

authentication
specific participant is not impersonated
ensures to one party identity of other

synchronization protocol
ensures recent injective agreement & authentication

3.5 protocol design

3.5.1 notation

A, B as roles
Alice, Bob as names (instantiations of roles)
KA for public key A, KsA for secret key A; {message}_KsA signing
KAB for symmetric key between A and B; {message}_KAB encryption
N_A fresh data (nonce), T timestamp; used for challenge, response
M_1 || M_2 message concatenation
A → B: {A, N_A}_KB when A sends B a message
modelled asynchronous, defines set of event sequences (traces)

3.5.2 principles of security protocols (Abadi Needham)

informal guidelines to spot errors (not necessarily minimal/optimal)
but no guarantees even if followed appropriately

explicit communication
interpret message based on content only
to counter replay or usage in different context

appropriate action
define conditions upon message is appropriate
to allow for transparent review

naming
name identity in message if needed to process it
to prevent usage in different context

encryption
define why what is encrypted
confidentiality assumes only receiver knows key
authenticity assumes only sender knows key
bind together parts of message by encrypting as one
produce random numbers by encrypting counters
to avoid redundancy

encoding
be explicit about encoding
prevent cross-protocol, different context reuse

trust
explicitly define trust relations protocol depends on
reason why they are appropriate and required
to allow appropriate implementation

signatures
do not prove knowledge of singed encrypted content
only prove knowledge of content readable by sender

nonces
define guarantees (new, random, unpredictable, timely, authorship?)
nonces can be used to prove freshness and authorship
define if nonces must be new / random
define if nonces must prove authorship
protect predictable nonces from replay
protect timely nonces from faulty clock

keys
do not assume used key has been establish recently
prevent continuous usage of compromised keys

3.5.3 basic mechanisms

cookie exchange
initial roundtrip to establish cookie, always resend on subsequent request
before any expensive computation validate cookie to prevent DoS
A → B : C1, B → A : C2 C1; A → B : C1 C2 X; etc....

freshness mechanism
nonce (but must be random)
one-way counter (but must sync & store, encrypt to avoid guessing)
timestamps (but must sync clock)
challenge-response w/ nonces (but needs roundtrip)

3.5.4 attacks

forms
man in the middle (C between A, B communication)
replay attack (record, later resend message, parts)
reflection attack (send information back to sender)
oracle attack (use protocol as decryption/encryption service)
guessing attack (protocol using pw allows to verify if correct pw found)
type flaw attack (B interprets M differently)

why its difficult to spot attackers
assumptions unclear (intruder or insider?)
complex model despite it looking simple
humans bad at envisioning all possible interleaving computations
real protocols more complex & faulty design/standardization process

3.5.5 examples

NSPK
(1) A → B : {A, N_A}_KB
(2) B → A : {N_A, N_B}_KA
(3) A → B : {N_B}_KB
MitM attack with I(A)

Otway-Res protocol
authenticated key distribution with key authentication, freshness
I as protocol run identifier, KSA, KSB already known
(1) A → B I, A, B, {N_A, I, A, B}_KSA
(2) B → S I, A, B, {N_A, I, A, B}_KSA, {N_B, I, A, B}_KSB
(3) S → B I, A, B, {N_A, K_AB}_KSA, {N_B, K_AB}_KSB
(4) B → A I, A, B, {N_A, K_AB}_KSA
type flaw attack with I(B), replays #1 as #4 back to A
type flaw attack with I(S), replays #2 in #3

Andrew Secure RPC
exchange fresh, shared key between two principals having a shared key
K_AB' key & N_B' nonce for next session
(1) A → B : A, {N_A}_KAB
(2) B → A : {N_A + 1, N_B}_KAB
(3) A → B : {N_B + 1}_KAB
(4) B → A : {K_AB', N_B'}_KAB
type flaw attack when MitM I(A), I(B) use #2 as #4

key exchange CA (denning & sacco)
key secrecy & agreement
T_A limits usage, B knows its the target because its K_B was used
T_A timestamp from A
(1) A → S : A, B
(2) S → A : C_A, C_B
(3) A → B : C_A, C_B, {{T_A, K_AB}_KsA}_KB
MitM attack when I(C) ↔ A uses #3 for I(A) ↔ B

electronic car key example
requirements are fast, cheap, usable
personal key K, car C, key between car & key K
(1) K → C "open" (but can do replay attack)
new requirement "if door opens, key was pressed"
(2) K → C {"open"}_K
new requirement "different secret, car remembers secrets"
(3) record, block, replay later
replace "key was pressed" → "recently pressed his key"
(4) time skew, open multiple times
new requirement "X times pressed, X times door open"
(5) K → C "open", C → K "challenge", K → C {"challenge"}_K

3.6 formal analysis of protocols

approach protocol correctness as system correctness
build formal symbolic model M as transition system with algebra
specify property to be archived (e.g. secrecy)
show correctness (model checking, theorem proving) in environment E

3.7 interleaving trace model

has linear ordering, synchronous model
trace is a sequence of events
protocol denotes a trace set (all possible interleavings)

formalism
P formalizes protocol steps
t formalizes existing state (require t ∈ P, previous_messages ∈ t)
N_A, N_B are nonces (require fresh(N_A))

example NSPK
0-3 formalize protocol, 4 formalizes attacker
0. <> ∈ P
1. t, A → B : {A, N_A}_KB ∈ P
2. t, B → A : {N_A, N_B}_KA ∈ P
3. (similar to 2)
4. t, Spy → B : X ∈ P if X ∈ has(sees(t))

if A sent N_A to B and A received N_A then B sent N_A

has(T)
gets smallest set of messages inferable from T
formal rules define attack model
trivial, pairing / unpairing, encryption / hash, decryption if key known
includes base cases (spy knows public keys, can generate nonces)

property

set of traces, formulate predicates with those sets
predicates describe property, evaluate to true or false

interactive verification
create proof scripts, then check with compiler
but unprovable goals

analysis with model checking
inductive definition allows to construct infinite tree
property corresponds to set of nodes
state enumeration to find node in property set
model checking tools to avoid complete enumeration
automatic, algorithmic methods (symbolic trees, nodes)
better than interactive constructed proofs

3.8 scyther

supports protocol verification/falsification for different attacker models
keys, hashes, key-tables, multiple protocols, composed keys
backwards search (starts with security violation) to initial (valid) state
generates isabelle, HOL proofs

specify protocols
declare existing key architecture
declare roles with vars, send_i, recv_i, claim_i

semantics
protocol P generates traces(P) in presence of intruder
intruder learns all messages (intruder knowledge IK)
intruder knows all previous messages at receive
intruder can replace any messages with all IK from before

transition system semantics
create thread (for role R and agent A create thread tid)
thread as ordered list of events, look at next event for conditions
send (select tid with send(m), add to trace & remove from tid)
recv (select tid with recv(m), add to trace & remove from tid)
claim (select tid with claim, log for subsequent checking)

example secrecy property
if honest A executes secrecy claim x then attacker does not know
for all t $\in$ traces(P)
if (thread X communicates with a in t and t(i) = claim(A, Secret, x)#X)
then x $\notin$ has(IK)

reasoning with semantics
can reason by hand, but computer much better
scyther automates search, gives attack example if found
fastest method, guaranteed to terminate, user friendly with UI

complexity results
undecidable in general
NP-complete for strong constraints (therefore only limited #thread)
analysis assumes only protocol being executed (chosen protocol attack)
could use separate keys for separate protocols (but expensive)

3.9 why is status quo so miserable

competitions for cryptographic primitives but not for protocols

inadequate processes for protocols
industry push main driver
often new standards, speed more important than verification
lack of understanding & politics
unhealthy competition, committees

examples
NSPK proved with BAN logic, but logic broken
WEP not proven, industry push overrides security analysis
WiMAX (telecom, for wide-distance) no formal specification
mesh network standard proven by unsound PCL
ISO 9798 (entity authentication) with errors, ISO amended

4 Protocols

4.1 access control

in centralized systems
security admin has authorization monitor
user authenticates against reference monitor
reference monitor implements access control to objects
auditing oversees the whole process

in a distributed setting
authentication by assertion was standard (just user id)
then introduced password (but in cleartext, thus not much better)

4.2 kerberos

authentication, authorization, audit (audit not implemented)
single-sign on protocol
one password per session, subsequent authentication behind the scenes
developed in the 80's, still used today, windows built-in

4.2.1 properties

secure (authenticated users access their authorized resources)
single sign-on (single password to obtain all network services)
scalable (scale with number of users & servers)
available (supported by replicating kerberos server)

4.2.2 general pattern (1)

setup of secure channel
A, B, trusted T which shares keys with A, B
A $\rightarrow$ T : request to communicate with B
T $\rightarrow$ A : {K_AB}_AT, {K_AB}_BT
A $\rightarrow$ B : {K_AB}_BT, {m}_K_AB
C can be between A and T to impersonate B

4.2.3 (1) + freshness & names (2)

loosely based on needham-schroeder shared-key protocol
establish shared channel & freshness
A $\rightarrow$ T : A, B, N_1
T $\rightarrow$ A : {N_1, B, K_AB, {K_AB}_BT}_AT
A $\rightarrow$ B : {K_AB}_BT
B $\rightarrow$ A : {N_2}_K_AB
A $\rightarrow$ B : {N_2 - 1}_K_AB (transformed to disable replay)
kerberos takes this without double encryption, nonces are timestamp
does not provide secrecy

4.2.4 kerberos IV

authentication using Kerberos Authentication Server KAS
authorization using Ticket Granting Server TGS
access control of servers which check the TGS ticket

operation
(1) login & request network service, send request ticket (once per session)
(2) session keys (like K_AB & ticket (like {K_AB}_BT) granted by KAS
(3) workstation sends ticket & authenticator to TGS
(4) TGS creates ticket for requested server
(5) workstation sends ticket & authenticator to server
(6) server verifies & grants access (may starts two-sided authentication)

authentication phase (authentication server)
once per user login session to get AuthTicket for TGS
K1 can be derived from user password
K2 shared between KAS, TGS
K3 session key between A, TGS; livetime of several hours
A $\rightarrow$ KAS : A, TGS
KAS $\rightarrow$ A : {K3, TGS, N1, {A, TGS, K3, N1}_K2}_K1

authorization phase (ticket server)
once per type of service to get ServTicket for B
K4 shared between B, TGS
K5 session key between A, B
A $\rightarrow$ TGS : {A, TGS, K3, N1}_K2, {A, N2}_K3
TGS $\rightarrow$ A : {K5, B, N3, {A, B, K5, N3}_K4}_K3

service phase (network service)
once per service session to authenticate at B
A $\rightarrow$ B : {A, B, K5, N3}_K4, {A, N4}_K5
B $\rightarrow$ A : {N4 + 1}_K5

multiple realms, kerberi
can use multiple realms (single realm defined by KAS & TGS)
inter-realm protocol supported, using symmetric keys (n^2)
made easier in v5 with public key cryptography

limitations
encryption not needed, attacker can flood KAS (initial message), DoS
double encryption redundant when returning tickets (removed in v5)
relies on synchronized clocks (replay using old tickets)
cryptographic weaknesses

in practice
application layer protocol
can wrap applications ("kerberize") to make secure (ftp, smtp)

4.3 TLS

provides secrecy, integrity, (optional mutual) authentication
conceptually a transport layer protocol, but no OS implementation

4.3.1 subprotocols

handshake (initiates connection with desired properties)
record (to send data, describes how compressed, authenticated, encrypted)
others (renegotiation of cyphers, error recovery)

4.3.2 core concepts

session (defines keys/encryption/MAC between client/server)
connection is a secure stream within session

4.3.3 handshake

hello
S_id for session identifier, P_a for cypher suite
everything unprotected, authenticated later
A $\rightarrow$ B : A, N_a, S_id, P_a
B $\rightarrow$ A : N_b, S_id, P_b (chosen from P_a)

server certificate
may allows additional key exchange for DH exchange
server may also request client certificate
B $\rightarrow$ A : certificate(B, K_B) (X.509v3 certificate)

client exchange
PMS pre-master secret, PRF pseudo random function, H hash function
PMS used to compute master secret M=PRF(PMS, ...)
A $\rightarrow$ B : certificate(A, K_A) (optional)
A $\rightarrow$ B : {PMS}_K_B
A $\rightarrow$ B : {H(...)}_K_a (optional)

finish
compute keys from (N_a, N_b, M) for MAC/IVs/clientK/serverK
finished is hash of all previous messages to avoid downgrade
authenticates server, guarantees integrity
A $\rightarrow$ B : {Finished_A}_clientK
B $\rightarrow$ A : {Finished_B}_serverK

4.3.4 using in practice

server has public key certificate (private key stored on server)
organization may additionally provide user certificates

problems
attacks on handshake, record layer protocols
downgrade attacks on cypher suites
timing attacks, error messages leaking information
implementation errors with buffer underflows (heartbleed)
but social engineering probably easier than exploiting SSL

4.3.5 attacks

MitM attack
copy target webpage & forward
social engineering, DNS poisoning, IP highjacking
users! (don't understand SSL, certificates, error messages)

phishing
5% of users give sensitive info to spoofed webpages
2 mio users resulting in loss of 1.2 billion on US
23% don't notice security indicators, majority ignores warnings

4.3.6 enhancements

two channel authentication
multiple communication channels (like TAN)
but requires existing relation with server (like registered phone number)

session aware user authentication
combine channel information with credentials
so forwarding does not work anymore

user client certificates
attacker cannot sign all previous messages correctly
PKI rollout complicated, not userfriendly

4.4 IPsec

end-to-end security between clients/servers or midboxes (firewalls, routers)
traffic filtering with a policy database
very complex standard, no comprehensive analysis exists
works on transport layer (done by OS), can be used by any application

4.4.1 elements

security association (SA)
defines security services (encryption, integrity, authenticity)
defines algorithms (DES, AES, MD5, SHA1) & session keys, IVs
setup by hand or negotiate with IKE

security policy database (SPD)
to decide for each packet transport mode, SA, ESP or AH, ...
configured by administrator, also basic firewall functionality

4.4.2 protocol modes

transport
between endpoints, gateway
payload encrypted & authenticated, IPsec inserted after IP header

tunnel
protects the path G1, G2 between parties A, B
payload protocol encapsulated in delivery protocol
original packet has A, B as source, target
G1 modifies packet such that G1 new IP source, G2 new IP target
G2 strips out original header (A,B)

4.4.3 authentication header (AH)

for integrity, authenticity of message/IP
next header (type of next payload), length (of AH), reserved
32bit security parameter index SPI to identify SA
32bit sequence number field (prevent duplication, replay)
32bit words with authentication data (like sha1 MAC)

transport mode
AH after original IP header (before IP payload)
using MAC of entire packet (except mutable fields)
end-to-end protection, but integration with IP needed

tunnel mode
AH after IP tunnel header (before original IP header)
inner header carries original target, source
outer header protected, may contains different IPs

4.4.4 encapsulating security payload (ESP)

for integrity (optional), secrecy of message/IP
32bit security parameter index SPI
opague transport data (parameters for crypto algorithms)

transport mode
adds ESP header after original IP header, ESP trailer & auth after data
encrypts data portion (payload) of each packet, IP header unchanged

tunnel mode
ESP header after IP tunnel header, ESP trailer & auth after data
old header, payload encrypted (optionally authenticated)

4.4.5 internet key exchange protocol (IKE)

on application layer, to negotiate security association
invoked if IPsec session started but SA unknown
establishes a SA (protocol, keys, cryptographic & hash algorithms)
based on DH and extensions, very flexible & complex

establishment phase 1 (main mode)
parties negotiate SA to use in phase 2
different variants (pre-shared symmetric or signatures or 2 PK variants)
offers identity protection, flexibility in terms of parameters
(1) A $\rightarrow$ B : C_1 (cookie) ISA_1 (cryptographic proposals for ENC, AUTH)
(2) B $\rightarrow$ A : C_1, C_2, ISA_2 (chosen proposals, B keeps state)
(3) A $\rightarrow$ B : C_1, C_2, X[,g,p], N_1
(4) B $\rightarrow$ A : C_1, C_2, Y, N_2
(5) A $\rightarrow$ B : C_1, C_2, {ID_1, AUTH_1}_K
(6) B $\rightarrow$ A : C_1, C_2, {ID_2, AUTH_2}_K
use skeyid=H({N_1, N_2}, DH secret) to derive keys
signed HASH=H(skeyid_a, {X, Y, C_1, C_2, ISA_1, ID_1}) for authenticity
need cookies to prevent DoS
can merge (6) and (4), but was not merged to do DH in parallel
need nonces because DH reused, cookies potentially not unique
there exists an aggressive mode which provides no identity protection

establishment phase 2
SA of phase 1 used to create more SA for further communication

5 public key infrastructure

5.1 key management

mechanism to bind identity & purpose to key
distribution, generation, maintenance, revocation of these keys

symmetric key with trusted parties, principals having keys with all others
asymmetric keys with public keys bound to principals

5.2 CA-base PKI

generate key-pair, send public key to certificate authority CA
CA verifies who generated key-pair, and signs public key
third parties can obtain signed public key

CA process
certificate signing request, money, legal documentation to CA
CA checks info then signs PK (by mail, call)

clients
store CA keys (root of trust), may have self-signed certificates
use PKI for encryption, authentication, non-repudiation

5.3 certificates

certificate binds identity to key
contains issuer, expiration, usage
signature of hash of content done by CA

X.509
defines structure of certificates, widely used
version, serial number SN, issuer name (pair (SN,IN) unique)
signature algorithm identifier (algorithm, parameters)
period of validity (start, end)
subject name
subject public-key info (algorithm, parameters, key)
signature (hash of all other fields, signed by CA)
issuer unique identifier, subject unique identifier (optional)

validate
validator already knows CA public key and trusts it
validator computes hash and checks signature & validity period

certificate forms
domain validation DV sends email to confirm membership
extended validation EV contacts organization, many other checks
marketing feature to make customer trust more

5.4 trust models

trust in an entity if it behaves as it should
trustworthiness of an entity if it actually behaves as it should

5.4.1 certification chains example

verify certificate with public key X
base case if signing key of X already trusted
recursive case if X signed by other trusted entity Y
trust that X correctly issues certificates (for signature)
trust that X recommended by Y is trustworthy (for recursive)

construction
A believes to have authentic X public key
A trusts X to certify keys
A has certificate for Y signed by X

set up problems
how does A have root of trust keys
how does A obtain the certificates
which CA does A trust and for what

5.4.2 direct trust

single CA for entire world

advantage
certificate chain of length 1
need only one public key to distribute
only a single CA we need to trust

disadvantages
no organisations trusted by all countries, companies, universities
inconvenient, insecure, expensive to obtain from distant organisation
periodic rekeying can't be done by single CA
CA has monopoly and can charge excessive prices

5.4.3 single CA + registration authority RA

RA verify name/key binding, send signing request to CA
CA has key of all RA
users interact with local RA

advantages
certificate chain of length 2
need only one public key to distribute
can revocate RA key easy because simply contact CA

disadvantages
disadvantages of direct trust
additionally must trust all RA

5.4.4 oligarchy CA

multiple CA, else same like direct trust

advantages
more convenient as CA can be local
competition prevents abusive pricing

disadvantages
compromise of any CA key suffices
requires principals to store keys (browsers, but MitM)

proposals
local scoping (regional CA for regional entities)
variant scoping (can only issue for specified namespace)

5.4.5 oligarchy CA variants

cross certificates
cross certificates (CA1 signs CA2 and vice versa)
they effectively recommend each others

configured + delegated CA
configured keys authorize delegated CA, complete transitive trust
good because users can obtain certificates easily
bad because any compromise is complete due to assumed transitive trust
also called web model, browsers have own store or use OS store

web model
certificate discovery (collect from different kinds of stores)
path validation (walk the walk, retrieve from URL or store)
revocation checking (check if any on path are revoked)

web of trust (PGP)
people recommend each other
no trust in CA, verify others manually
keys for communication, keys to sign others identity
store known keys & level of trust (full, marginal, no trust)
key valid if singed personally OR by fully trusted OR three marginal
key valid if path of singed keys is 5 steps or shorter

5.5 web model

MitM possible with single corrupted CA
root of trust in OS, browser (over 1400 in microsoft, mozilla)
buy-in with 50k per browser

breaches
incompetence, greed, collusion, corruption of CA lead to multiple breaches
use cases of bad certificates include espionage, observations
lenovo embedded root CA to inject ads, private key leaked
bogus certificates can be bought to MitM any connection

CA assumptions
has no root or delegated key compromised
does really check identity before issuing

crypto assumptions
secure, no vulnerabilities

browser assumptions
root certificates are correct, unaltered
routines to update certificates work correctly
chaining works as intended
no remote code execution
malicious site cannot overwrite locks

user assumptions
user checks for https, locks, correct urls
user does not accept bogus certificates, takes warnings seriously

5.6 alternative models

5.6.1 stakeholders

users (identity theft; needs to remove bad certificates)
domains (may be liable for damages; can't really defend)
browser/OS (trust issues, very powerful; rarely use power)
certificate authorities (weakest link security; want to avoid blacklist)

5.6.2 client-centric

requires no change from server operators & reduce trust in CAs

policy engine
restrict type of certificates a CA can generate (scope with name-spaces)

perspective repository
notary server visits SSL webpages & stores certificates
user configures notary server, verifies all certificates
user detects MitM, but no privacy (notary server learns connections)

convergence repository
contact convergence server over intermediary, onion routing
like perspective with privacy, but more latency

SSL observative
collects global SSL info, but not frequently updated

monkeysphere project
distribute keys independently of CA's, for SSH keys, RSA verification
kicks in if other certificate failed, uses PGP web of trust

5.6.3 CA-centric

easier revoking of certificates

certificate revocation list (CRL)
list of all revoked certificates maintained by CA
url contained in root certificates & publicly accessible
clients check themselves or use validation service
revocation because changes to names, private key compromised
X.509 with creation date, issuer, date of next CRL, revoked certificates

online certification status protocol (OCSP)
OSCP server states timestamped if certificate is good/revoked/unknown
but another roundtrip, servers slow, errors not treated fatal
attach OSCP to all certificates ("stapling") to counter some issues

5.6.4 domain centric

accountability or removal of CA validations

key pinning
store hash of certificates for domain, accept exclusively these
secure but does not scale, cumbersome to change
trust on first use (TOFU) with http public key pinning header HPKP
but if first use intersected, can disable usage & MitM
trust with TACK sent in TLS server hello, a versioned public key
defined by site administrator; client accepts after seeing it multiple times
trust over pin preinstalled in browser

DANE
DNS based authentication of named entities (DANE)
keys tied to DNS entries, DNSSEC protected
DNSSEC hierarchical, single root of trust in USA

certificate transparency (CT)
log servers for accountability, as misbehaviour deterrence
certificates have to be in public log, domain owners audit log
log is merkle hash tree MHT, append only
log server verifies, issues promise to include in MHT, later adds to MHT
promise in form of singed certificate timestamp SCT
certificate must be in MHT or have a SCT to be valid
does not prevent attacks, no revocation, assumes others audit logs

sovereign keys (SK)
owner of domain registers certificate to name at timeline server
can update that certificate with private key
browser contacts the timeline server/mirros to get newest certificate
but no revocation, needs additional roundtrip / data

accountable key infrastructure (AKI)
domain owner defines security policy (#signatures, trusted CA's)
pushes certificate to integrity log servers (ILS)
domains send ILS verification information together with certificate
client verifies ILS information with root keys
auditors validate ILS integrity (check MHT)

attack-resilient public key infrastructure (ARPKI)
domain requests at CA1, CA1 adds to log servers, CA2 signs
client gets domain policy (defines CA, revocation, updates)
attacks visible, proven correctness for up to 2 colluding entities

5.6.5 other issues

naming & identity
difficult to bind identity to name (not unique, can change)

backup
backup critical keys to ensure continuous access
store key at one or multiple authorities (using sharing schemes)
loss of decryption key problematic, need to revoke
loss of signing key unproblematic, signatures are still checkable

escrow
storing keys for legal entities to get access to data

CDN's
improve availability & performance because server closer to end user
but user needs valid ssl certificate, so CDNs use X509.3 extension
subject alternative names SAN allows to specify other domain names
but certificates change frequently and revocation difficult
security vs usability tradeoff

6 access control

6.1 motivation

cryptography solves CI (confidentiality, integrity) for (network) files
but ill suited for controlling processes (availability)
security policy defines access restrictions, processes, data
need to formalize these policies enough to allow enforcing

6.2 principles

declarative access control
authorization specified by relation
user is granted access if he has all required permissions

programmatic access control
role-based authentication combined with more (eg. state)

6.3 core processes

6.3.1 identification

associate an identity to a subject

6.3.2 authentication

verify the validity of something claimed by an entity

mechanisms (combine to be stronger)
what the user knows (pin, password)
what the user has (smart card)
what the user is (fingerprint, voice, eyes)

problems
passwords bad (can't keep long secrets, user is stupid)
biometrics difficult (can't be changed, leaks dangerous)

6.3.3 authorization

grant rights/permission to an entity
specified using mathematical structure, rules, programs

relation based
state of the form user x object x right
state transitions cause by administration

6.3.4 access control

protect system according to security policy
subjects, objects, rights differ for each application
use abstraction to derive general rules

applications
memory management hardware
OS, file systems
middlewares, software, firewalls
physical protection

centralized reference monitor
mandatory access control
security admin creates authorization db
monitor intercepts requests, decides if authorized using db
additional components for auditing, administration

mechanism design
principle of complete mediation (check all accesses)
principle of minimal trusted computing base (KISS)
but hard because faulty implementation, lower-layer attacks
even harder because system tampering, side-channel attacks
even if everything works, user himself may exposes sensitive info

6.4 policies & models

6.4.1 security policy

defines what is allowed
in terms of high-level rules & requirements

6.4.2 security model

formal representation of a general class of system
highlight security features at chosen level of abstraction
policies/models abstracted as states/transitions

protection state P
part of system state concerned with security
file system (read/write access)
network (packet header, packet location)
program (run-time image like call stack)
security policy defines Q (allowed states, subset of P)

transition system
abstraction capturing systems dynamic, visualized as state machine
security mechanism prevents transitioning into P \Q

6.4.3 examples

6.4.3.1 banking

security policy
users can only view their own info

mechanism
access control on application/db servers

protection state
access control data ACL, account ownership data, ...
unauthorized state would be where X sees data of Y

6.4.3.2 kerberos

security policy
state which users can access which servers
formalized by administrator which registers servers/users in db

mechanism
kerberos servers & kerberized applications

protection state
determined by kerberos server, protocol, client, server states

6.4.3.3 proprietary data

security policy
info on Y is confidential, can only be read by group X

mechanisms
printouts must be securely stored / shredded
digital copies must be protected (AC, cryptography)
backups must also implement security policy

protection state
includes technical, procedural, organisational controls

6.5 access control matrix model (ACM)

simple framework describing privileges of subjects on objects
focus is on authorization, not mechanisms themselves
abstract model which can be instantiated by different concrete entities

6.5.1 model

subjects (users, processes, agents, groups)
objects (data, memory banks, other processes)
privileges (read, write, modify)

reference monitor
subject (principal) does operations (request)
checked by reference monitor (guard) then forwarded to object (resource)

protection state st
triple of subjects S, objects O, matrix M
M defines privileges P for each (s,o) tuple
initial state st_0 = (S_0, O_0, M_0), system trace st_i = (S, O, M)

state transitions
formalized by set of commands composed of operations, transition |-
operations can be expressed as operational semantics
operations like enter/delete p into M(s,o), create/destroy s or o
commands like if permission ∈ M(s, o) then delete o

6.5.2 application

specify concrete systems as instances
verify that instances are secure relative to a specific property
reason about general properties of all systems in the class
use as basis for mechanism design

unix protection system
subjects are users, objects are files
each file has owner, group
file operations are read r, write w, execute x
directory operations are list l, usable in constructing paths x
each permission specified for user, group, others
M with f in x-axis, f & u in y-axis; M(f,f) for rights, M(u,f) for owner
if own ∈ M(u, f) & owner_can_read ∈ M(f,f) or others_can_read ∈ M(f, f)
then allowed to read the file

6.5.3 policies & security

model instance describes what subjects can do (mechanism)
not necessarily describes what they are authorized to do (policy)
if both model & policy formal then correctness well defined
policy semantics implemented as set of authorized states
states S, by policy authorized states P, by mechanism reachable states R

correctness
for all properties it applies that system |= property(formal policy)
secure if R subset_of P, precise if R = P

example policy
only owner of file can read it (policy)O objects,
authorized iff for_all s ∈ S, o ∈ O, if R ∈ M(s,o) then Own ∈ (s,o)
system is secure if R ≤ P, precise if R=P

limitations of sets
properties depending on the past need set of traces (e.g. max 5 access)
system secure if all possible traces authorized (use security automata)

6.5.4 decidability

leakage safety
state st is safe for some permission p if later p never written into new cell
so no state st' of st |-∗ st' such that st' contains new occurrence of p

undecidability theorem
given arbitrary |-, st (state), p (permission) undecidable if st is safe

proof by reduction
modify matrix M such that objects, subjects same set, own/end symbols
describe command as transition, eg. f(q, X) = (p, Y, L)
create turing machine TM with 2D configuration, transitions
TM halts if p written to new cell, reduced to halting problem

consequences
cannot determine algorithmically if permissions are leaked
proof only for specific class of systems
decidability may be possible for instances, other properties
expressivity & complexity in analyzing go hand-in-hand

weaker models have decidability
fixed number of subjects, objects
restrictions to mono-operational rules (single action)

6.5.5 implementations

other data structures as possible implementations besides matrix

access control list (ACL)
linked list or table directly associated to object
i entry gives rights for i user = M(s_i, o)
for each request check permission, relies on correct authentication
easy implementation, delegation, removal, fit for DAC
but viewing permissions & deleting subject difficult

capability list (CL)
linked list to express all rights of single user
entry of list is signed capability tuple of (object, permissions)
reference monitor checks tuple, checks identity if delegation disallowed
easy delegation, sign tuple for usage in distributed systems
but revocation, overview of capabilities is difficult

6.6 role based access control (RBAC)

create a role for each business role in enterprise
associate each role with minimal permissions to get it done
assign roles to users based on their business role
implement with ACM, but does not scale & difficult to maintain

structure
decouple users & permissions using roles
recast AC matrix from triplet (u, o, p) to tuple (u, o p)
now assign each permission a role, and give users roles
relation user x role UA, relation role x permission PA
access control AC = PA o UA, join all the stuff

with role hierarchies
hierarchies are partial orders on roles ≥

larger roles inherit all roles from smaller roles
AC = PA o $\geq$ o UA

advantages
roles are a stable concept, users come and go
assignments understandable by business (easy administration, audit)
supports least privilege (assigning minimal permissions required for role)
support separation of duty (ensure invocation of different roles for task)

disadvantages
no fine-grained attribute based AC (power only over subordinates)
no system-state based ac (transfer money if enough funds)
no support for delegations & obligations
can't enforce based on sequences of events

battle disadvantages
combine with programmatic access control
use richer policy languages like XACML

in practice
many AC mechanisms are based on this
usually combined with programmatic access control
difficult if number of roles large (maintenance, audit)
do role-mining, change impact analysis, other techniques

XACML (OASIS standard)
modern extensible access control markup language, XML
define policies to be invoked, can also depend on system state
target defined by subject (user), resource (location), action (command)
for target, define policies & rules, evaluated by CA
rules / policies defined like <target><conditions><effect>
full example <any user, read, file1><in-range-time 8-20><deny>
combination algorithms deny-override, permit-override (strongen, weaken)

6.7 discretionally access control (DAC)

owners of objects have sole authority to grant / revoke rights to others
flexible but open to mistakes, negligence, abuse
no central control of information dissemination
all users need to understand mechanisms & security policy
users may prefer delegation restrictions (less risk of trojan, virus)

unix DAC
each file has owner, group
rights for group, user, others
only files owner & root can change its ACL
right delegation using setuid (executor takes identity of owner)

6.8 mandatory access control (MAC)

specifies system wide access restrictions to objects
mandatory means the system owns resource and controls access
shifts power from user to system, more rigid than DAC
MAC policies identified with multilevel security policies
implemented in SElinux, based on DAC but can specify additional policies

6.8.1 labels / classifications

AC decisions based on this
sensitivity / clearance represents degree of sensitivity
compartments / categories represent need to know

6.8.2 military context

sensitivity clearly labelled on folder's cover
each user cleared to see data up to his clearance level
physical security used to control data access
user presents clearance to guard

labels
specify clearance level
top secret (level), background check (subject), grave (objects)
secret (level), interview (subject), serious (objects)

compartment
specifies domain for need to know policy
{middle east, satellite data, isreal}
create lattice from different compartments
see info if label dominates object

6.8.3 set theory

partially ordered set
set S with relation $\leq$, written as (S, $\leq$)
(1) reflexive for all a, $a \leq a$
(2) transitive if $a \leq b$ && $b \leq c$ then $a \leq c$
(3) anti-symmetric if $a \leq b$ && $b \leq a$ then a == b
natural numbers (total order)
set under subset relation (not total order)
natural number ordered by divisibility (no total order)

lattice
(1) all from partially ordered set
(2) each item has least upper, greatest lower bound
natural numbers in interval

computing dominance
authorization compares labels component wise
label = <clearance, compartment set>
linearly ordered clearances, subset ordered compartments

why use lattices
all pairs have guaranteed least upper / greatest lower
given two objects, assign minimal label to user
given two users, assign maximal label to object
well suited for need-to-know policies, least privilege labels
else incomporable allow info flow (based on deny/allow)

6.8.4 Bell-LaPadula (BLP) (confidentiality)

read-down (label x_s can only read x_1 if x_s $\geq$ x_1)
write-up (label x_s can only write x_1 if x_1 $\geq$ x_s)
prevent reclassification by write-down of high-read secrets

BLP high to low flow solution
subjects have two labels, curlevel(s) and maxlevel(s)
maxlevel(s) is the dominating label of objects allowed to read
curlevel(s) is the dominating label of all objects actually read
if maxlevel(receiver) > curlevel(sender) then its OK

other solutions
temporarily downgrade subjects classification
trusted subjects which can violate property (4-eyes, committee)

6.8.5 BIBA (integrity)

read-up (object label dominates readers)
write-down (writes label must dominate object label)
preserves integrity of data, while still allowing modifications

low-water mark alternative
integrity is equal to lowest source integrity
but too simplistic for real-world

6.8.6 combine integrity & confidentiality

only read/write at same classification
assign different labels for integrity/confidentiality
use BLP for classification, biba for integrity

mandatory integrity control (MIC)
read-down is allowed, else similar to biba
uses hierarchy of integrity labels (low, medium, high, system)
users, processes have medium integrity
priority over DAC
used in vista, programs & IE run at low integrity

chinese wall model
commercially inspired conflict of interest solution
organize data in sets, and avoid knowledge transfer (competitors)
if accessed data in one set, can't access other

limitations
restricts read/write, but other actions may leak information
covert channels (error messages, timing behaviours, lock/unlock of files)

6.8.7 interface model

specifies restrictions on IO rather than actions (like ACM)
more abstract, have to define possible covert channels

specification tricky
over specifications (secret key identical to legitimate leak)
under specification (leak secret key -1)

non-interference
no information flow from A to B inside a system S
if subject s with security level l uses device
then output of device may only depend on data $\leq$ security level l
f_s(x, y) = x is not-interfering if x $\leq$ security level l

non-interference formalization
out(u, I) is value of function for user u and input list I
purge(u, I) removes all entries from I which are above clearance of u
out(u, I) must equal out(u, purge(u, I))

6.9 monitor based enforceability

security models define systems, polices (with allowed states)
fulfilment of a policy by system is well defined (reachable states)
if policy not fulfilled by system, add security mechanism

execution monitoring
security kernels, reference monitor, firewalls, ...

formal setup
universe U of all finite/infinite sequences (abstract executions)
security policy P as a predicate on sets of executions, subset of P(U)
system S defined set of actual executions, E_s subset of U
S satisfies P if $E_s \in P$

formal requirements
safety properties (nothing bad has happened)
temporal properties (nothing bad ever happens)
(1) computable over executions (can determine for each single element)
(2) prefix closure (prefix dominates over extension)
(3) finite refutability (determined by finite prefix)

security automata
<Q, Q_0, S, f> for f: $(Q \times I) \rightarrow P(Q)$, I : {action(s) | $s \in S$}
$Q_{i+1} = f(q, s_i)$ for s input symbol, if Q_{i+1} empty then reject
s are actions ("open file"), Q are states ("file closed", "file opened")
simply only put arrows (actions) with what is allowed
run automaton in parallel with system
each action on system leads to a transition, abort if none found

non-interference
can't be analyzed with monitor, because of (1)
purge(u, I) works over all input histories

7 privacy

7.1 anonymity

assume actions can be observed
anonymous within a group if anonymity set large enough
then actions are indistinguishable from actions of others

7.1.1 anonymity set stages

absolute anonymity (like shuffle network)
beyond suspicion
probable innocence (like proxy)
possible innocence
exposed
provably exposed (like IP)

7.1.2 use cases

socially sensitive communication (crime victims chats)
law enforcement (crime reporting)
corporations (hiding collaboration)
political dissidents (criticizing government)
government (negotiations, whistle blowing)
criminals (arrangements)

7.1.3 anonymous communication

sender, receiver anonymity, unlinkability
confidentiality of principals identities, relationships

pseudonyms
hide real name behind pseudonym

computer accounts
logins are pseudonym, create new ones at any time
but ISP knows real identity

7.1.4 recipient anonymity

recipient is not known

broadcast
just broadcast message (radio, multicast, ring network)
encrypt information so only intended receiver understands
but DDoS

7.1.5 sender anonymity

sender is not known

proxies
packets anonymized by proxies
but single point of failure, correlations

cascaded proxies (onion)
encrypt message with proxy public key
include next proxy address for chaining

7.2 mix networks

unlinkability of sender/receiver against global eavesdropper

attacker model
can learn origin, destinations, representations of messages
can inject, remove, modify messages

randomized pk cryptography
attacker could guess cleartext & encrypt with pk to verify
therefore encrypt with random R, sign with public constant C

full workflow
users sends {R_1, {R_0, M}_KA, A}_K1 for K1 pk of mix
mix discards duplicates
mix decrypts message to {R_0, M}_KA, A
mix discards dummies if any
mix generates dummies
dummies & real messages put into buffer, then sorted
mix sends batch (including {R_0, M}_KA to A)

foiling traffic analysis
hide message size (pad/split into fixed-sized blocks)
hide order of arrival (use fixed or random ordering)
suppress replays (filter duplicates, include timestamps)
sufficient traffic (clients send/receive dummy messages)

recipe
the mix returns a recipe (signature of whole message)
of the form {C, {R_1, {R_0, M}_Ka, A}_K1}_K
sender proves by publishing R_1, {R_0, M}_Ka, A
verifier encrypts this with K1, verifies with signature

mix networks
to avoid single link weakness
encrypt for all mixes around the way
each mix peels off most outer layer

untraceable return address
sender includes {R1, Ax}_K1 (return address), K_x (new pk)
responder sends response back to mix {R1, Ax}_K1, {R0, M}_Kx
mix sends {{R0, M}_Kx}_R1 to Ax

application in evoting
collect digital pseudonyms K (public key to verify signatures)
roster is the published list of all pseudonyms
K, {C, V}_K-1 consists of a vote
can eliminate duplicates, tally votes publicly

achievements mix networks
no correlation between input output
only single mix on path used needs to be honest
dummy traffic makes the whole network to an anonymity set
sender anonymous to receiver, receiver can even respond
recipes allow certified mail by all mixes on the path
but delay, dummy overhead, encryption overhead

volunteer operated
expensive, bad press, lawsuits, compromised by governments
but TOR quite successful (at TCP level, no dummy traffic)

7.3 crowds

randomization, encryption for anonymity
peer-to-peer; each client is also a server
messages take random paths
different degrees of sender/receiver anonymity

technical
user runs process called jondo (reference to Jon Doe)
on start, jondo registers with trusted server called blender
blender informs other jondos in crowd of new member
blender distributes symmetric keys between jondos

request
send HTTP request to some other jondo
jondo flips coin; to either forwards to other jondo or executed request

guarantees
local eavesdropper compromises sender anonymity
set of corrupted jondos fine as long as n big
corrupted end servers do not hurt sender anonymity
but weaker than mix networks (only versus local adversaries)

8 data protection

8.1 GDPR

motivation
increasing amounts, increasingly sensitive data
anonymity services only decrease, but can't stop collection

new legal mandate
more individual rights
high-risk activities have stronger requirements
high fines, max(20mio, 4% of global turnover)

company needs to answer
what data is collected & how it is processed
for what purpose the data is collected
where does the data flow (necessity, protection, longevity)
how are rights of customer protected at each step
internal answers (how to prove compliance, detect data breaches)

setup
formulate (interests, purposes, security of data)
usage (data subject uses service)
processing (collect, process according to formulation)
legitimate checking (formulation is reviewed)
audit (check that collection/processing valid)

scope
identifiable personal data (anything about personal/business life)
must be protected according to GDPR
sensitive data (like political/religious believes, medical data)
must be protected additionally (do a data privacy risk assessment)

rights
transparency concerning purpose of collection
access control, adequate security for data
personal copy of own data
rectification (update) of personal data with adequate delay
erasure (right to be forgotten)
processing restrictions demanded by data subject
data portability for export / transfer of data
data processing respects subjects rights

data processing rights
purpose limitation (only use for purposes collected for)
accuracy (up to date)
immediate erasure (if no longer needed)
anonymous (users made unidentifiable as soon as possible)
everywhere (also in cloud, shared responsibility with processor)

8.2 data protection policy

how data is used ("to create your account")
under which conditions ("unless you indicate..")
with which obligations ("after 6 months account is removed")

platform for privacy preferences (P3P)
made by of W3C, structured way to specify privacy policy
tools can summarize, compare with user preferences
XML specifies consequence, purpose, recipient, retention
other standards more limited (SAML, EPAL, SOAP)

mechanisms
based on company, legal process (NDA, regulations, contracts)

US online privacy alliance
guidelines for collection, use, disclosure of personal data
special mechanisms to protect children
self-enforcing, with third party monitoring & audit

8.3 technical perspective

system model
actors are data provides (servers) & data consumers (clients)
operations are communication & CRUD on data
each data has data owner

provisions
conditions in past & present, ensured with access control
specification, enforcement well understood

obligations
conditions in the future, ensured with usage control
time (how long stored)
purpose (for what used)
actions (notify at each usage)
cardinality (#of uses)
technical, governance restrictions (stored encrypted)
necessary updates (freshness required by laws)

enforcement of data protection
enforce both provisions and obligations, on server & client
feasible if checkable that all executions of system compliant
needs controllability (prove it or controlled environment)
observability as weaker option (watermarks, audits, high trust)

server-side mechanisms
state of the art like IBM Tivoli Privacy Manager
provision handling (access control)
limited obligations handling (like logging)

user-side mechanisms
not established
protect from careless, negligent, dishonest users

9 evoting

voting process with digitalized parts
collection & tallying of ballots lot faster by machines
lower cost (no print) & effort (convenient voting), less errors (misspellings)

9.1 issues

hackable machines
filmed while typing root password
CCC against evoting, wants prohibition of evoting
ballots must be secret & verifiable, looks like a contradiction

9.2 target

archive verifiability & privacy

9.3 voting systems

paper voting systems
mechanical voting (marble into drum)
direct recording (computer in booth)
optical scan (handwritten but tally electronically)
internet voting (webpage)

9.4 context

protocol participants, adversaries $\models$ property

9.5 roles / participants

election authority
sets up the election
tabular tellers collect/tally the ballots
registration tellers generate credentials used to vote

voters
cast the vote

devices
used to compute, cast votes
often voters/devices not distinguished

auditors
perform checks to detect malicious behaviour

bulletin board
append only storage, publicly accessible
stores ballots (votes) & election results
evidence that all works correctly

9.6 adversaries

compromised roles
compromised devices
compromised authorities

communication channels
dolev-yao adversary
assume channels which limit what adversary can do
assume anonymous channel to send vote
assume authentic, confidential channel to register

physical presence
voting booth as protection
but adversary could be with voter at all times
limitations of indirect coercion (misinformation, etc)

9.7 human rights

will of the people expressed by periodic & genuine elections
universal & equal suffrage with secure vote

9.8 voting protocol properties

9.8.1 confidentiality

X is known term (yes/no)
but if only single vote, attacker can deduce X

9.8.2 privacy

unlinkability of sender/receiver, can't link vote to voter
adversary may not distinguish yes/no vote from alice
but may learn something form final result

receipt-freeness
voter can't prove to adversary how he voted

coersion resistance
adversary can give active commands to voter
but still can't be sure voter did vote that way

security against forced abstention
adversary can't hinder voter to vote

vote-independence
can't relate single given vote to any other given votes

everlasting privacy
vote privacy ensured long term
even if cryptographic stuff breaks (use one-time pads)

9.8.3 integrity

result must be correct

9.8.4 verifiability

everyone can verify result is correct
makes system secure against malicious behaviour, bugs

individual verifiability
each voter can verify that its vote is recorded
can only verify himself

universal verifiability
everyone can verify that recorded votes are counted correctly

end-to-end verifiability
each voter can verify that its vote is tallied

eligibility verifiability
only registered voters can vote
nobody can submit more votes than allowed

9.8.5 availability

no voter can be prevented from casting the vote

9.8.6 robustness

procedure tolerates misbehaving voters

9.9 code voting

end-to-end verifiable internet voting with ballot privacy
voters platform compromised, server can keep secrets

process
voter receives vote codes for candidates
voter sends corresponding code

assumptions
codes random & unique; known to election authority
random order of candidates

bulletin board (individual verifiability)
public only after closing of election
shows sorted list of vote codes

cut&choose (universal verifiability)
election authority shuffles table with code, candidate, bool
conceals first two columns, auditor chooses one to uncover
auditor verifies bulletin board (code) or tally (candidates)
repeat multiple times till high probability
but assumes rows of election authority correct

ballot audit (soundness)
voter receives two ballots (code/candidate sheets)
chooses one to audit, one to vote for
audited ballots published on bulletin board

privacy
not receipt free, must trust election authority

9.10 homomorphic encryption (Benaloh)

public/private key pair, private key may be shared

setup
block size r, prime p, prime q
n = pq, s = (p-1)(q-1), t = s/r
choose 1 < x,y < n such that x = y^t mod n
pk(y, n), sk(s, t)

protocol
send pk to B
B picks message m < r, chooses random u < n
B sends back c = y^m u^r mod n
a = c^t mod n = y^t^m u^s = x^m
m = log(a) / log(x)

verifiability
enc(m_1) ∗ enc(m_2) = enc(m_1 + m_2)

9.11 new forms of democracy

liquid democracy
delegate vote to someone else

random sample voting
randomly selected subset of population votes

9.12 alethea

provable secure voting protocol
server trusted for privacy
dolev yao network adversary

selection phase
server commits to public event
server generates vote id for all voters
server posts list of voters to bulletin board
event happens, produces randomness
server posts sublist of voters based on randomness
voter checks if on bulletin board

voting phase
device computes encryption (pk of server) of vote, voteid
server tallies all votes, decrypts all voteid
server posts all to bulletin board (vote/voteid separated)

properties
privacy, receipt-freeness
verifiability (plain text zero knowledge proof)

10 varia

first order logic (FOL)
boolean statement of the form b1 ^ b2
can contain quantifiers for_all, there_exists