

Internet infrastructure Practical Cryptography

Prof. dr. ir. André Mariën

Cryptography

- Seems like magic
 - Is very mathematical
 - Is very complex
 - Is highly sensitive to details
-
- Stay away from core cryptography
 - Use experts if needed
 - Protocols are even more difficult

Substitution ciphers

- Replace each symbol with another symbol
- Attack: frequency analysis
 - The more data you have, the faster it goes
 - One plaintext – ciphertext combination is extremely valuable
- Lessons
 - Analysis is getting better with the possession of
 - More cipher text
 - Cipher text/plain text combinations
 - Substitution is present even in AES (just more intelligently)

Xor – often reinvented

- Idea:
 - Xor: $(0110) = (0011) \text{ xor } (0101)$
 - Cipher = text xor key
 - Text = cipher xor key
- The results looks very random and very well protected
- The system uses a key
- So it looks like it must be good

- Attack: bypass key problem
 - $C1 \text{ xor } c2 = pt1 \text{ xor } pt2$
 - $c1 \text{ xor } c2$
 - $= (pt1 \text{ xor } key) \text{ xor } (pt2 \text{ xor } key)$
 - $= pt1 \text{ xor } pt2 \text{ xor } (key \text{ xor } key)$
 - $= pt1 \text{ xor } pt2 \text{ xor } (0\dots0)$
 - $= pt1 \text{ xor } pt2$
- Now you have again frequency analysis
- One known plaintext/ciphertext reveals the key
 - $Key = ct \text{ xor } pt$ ($ct \text{ xor } pt = (pt \text{ xor } key) \text{ xor } pt = key$)

Frequency distribution graph - English

Table A-1. Frequency distribution digraphs.

		TOTAL																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
A	3	6	14	27	1	4	6	2	17	1	2	32	14	64	2	12		44	41	47	13	7	3		12		374
B	4				18				2	1		6	1		4			2	1	1	2				7		49
C	20		3	1	32	1		14	7		4	5	1	1	41			4	1	14	4		1		1		155
D	32	4	4	8	33	8	2	2	27	1		3	5	4	16	5	2	12	13	15	5	3	4		1		209
E	35	4	32	60	42	18	4	7	27	1		29	14	111	12	20	12	87	54	37	3	20	7	7	4	1	648
F	5		2	1	10	11	1		39			2	1		40	1		9	3	11	3		1		1	141	
G	7		2	1	14	2	1	20	5	1		2	1	3	6	2		5	3	4	2		1				82
H	20	1	3	2	20	5			33			1	2	3	20	1	1	17	4	28	8		1		1		171
I	8	2	22	6	13	10	19				2	23	9	75	41	7		27	35	27		25		15		2	368
J	1				2										2						2						7
K	1		1		6				2			1	1						1								13
L	8	3	3	9	37	3	1	1	20			27		1	13	3		2	6	8	2	2	2		10		183
M	36	6	3	1	26	1		1	9				13		10	8		2	4	2	2				2		126
N	26	3	19	52	57	9	27	4	30	1	2	5	5	8	18	3	1	4	24	82	7	3	3		5		397
O	7	4	8	12	3	25	2	3	5	1	2	19	25	77	6	25		64	14	19	37	7	8	1	2		376
P	14	1	1	1	23	2		3	6			13	4	1	17	11		18	6	8	3	1	1		1		135
Q													1					1			15						17
R	39	2	9	17	98	6	7	3	30	1	1	5	9	7	28	13		11	31	42	5	5	4		9		382
S	24	3	13	5	49	12	2	26	34		1	2	3	4	15	10		5	19	63	11	1	4		1		307
T	28	3	6	6	71	7	1	78	45			5	6	7	50	2	1	17	19	19	5		36		41	1	454
U	5	3	3	3	11	1	8		5			6	5	21	1			31	12	12		1					130
V	6				57				12						1					1							77
W	12				22			4	13			1	2	19				1	1						1		76
X	2		2	1	1	1		1	2					1	1	2		1	1	7							23
Y	6	2	4	4	9	11	1	1	3			2	2	6	10	3		4	11	15	1		1				96
Z	1				2				1																		4
TOTAL	370	46	154	217	657	137	82	170	374	8	14	189	123	397	373	130	17	368	304	462	130	75	77	23	99	4	5000

Basic building blocks

- Digests: one-way hash functions
- Encoding: normalized, transportable content
- Symmetric key cryptography (private key): confidentiality
- Public key cryptography (asymmetric cryptography): key exchange, signing
- Random generator
 - One of the most fundamental and difficult ones

Digest

- One-way (hash) function
 - Examples: MD5, SHA-1, SHA-2, RIPEMD
- Maps many bytes to one fixed length bit string
- Desired properties:
 - Very sensitive to -even single bit- changes
 - Hard (complexity theory) to reverse: practically impossible to find a message with a given digest
 - Birth day attack is highly unlikely: practically impossible to find two messages with the same digest
 - Collisions are highly unlikely: massive amounts of digested data still produces unique hashes
- If the digest does not possess these properties, the solution build on top is possibly unsafe

Note on digests

- Recent research (Jan 2005):
 - MD5 collisions have been published
 - Rumor is that a SHA-1 collisions could be found
- Must understand what it means
 - Collisions exist (obvious)
 - Examples are found (useful for further research)
- Depends on the usage if it is a problem
 - Counter example: HMAC
- More recent results
 - Two certificates with same MD5 hash created
 - One: normal certificate, other CA certificate
 - High risk!

Hash-derived solutions

- Password protection
 - Avoid database of plain text passwords
 - Store hash(salt+password)
 - To validate, compute (salt+maybe password), compare with stored result
- Digital signing
 - Compute hash(text+password)
 - Can only be computed if the password is known: can be verified
 - Drawback: need a database with real passwords
- First step in digital signing
 - Reduce the data to be signed to a hash
- Integrity checking
 - Compute the hash, store safely
 - To check, recompute and compare with previous

HMAC

- Defined in RFC 2104
- Standardized and proven digital signature
- Based on shared secret
 - No non-repudiation (without very strong processes)
- Insensitive – proven- for birthday problems in the hash
- Principle:
 - $\text{Hash}(\text{data}\&\text{mask1}, \text{hash}(\text{data}\&\text{mask2}, \text{secret}))$

Symmetric key

- Same key is used for encryption and decryption
 - Sender and receiver share same secret key
- Sender and receiver use same, well-known algorithm
 - AES originates from K.U.Leuven (COSIC)
 - Others: DES, 3DES
- All secrecy is in the key (space)
 - Security depends only on the secret, not the secrecy of the algorithm
- Major problem: how to establish the secret?
 - Key management
 - Key storage

Public Key Cryptography

- Two linked keys / key pair
 - One kept private (secret), the other one is published (in principle)
- Properties:
 - $m = \text{decrypt}(\text{public}, \text{encrypt}(\text{private}, m))$
 - $m = \text{decrypt}(\text{private}, \text{encrypt}(\text{public}, m))$
- Usage:
 - Confidentiality establishment across unsafe channel
 - Signatures

Public Key Cryptography

- NEVER consider this as similar to symmetric key crypto
- Example:
 - Answer “yes”/”no”
 - Encrypt yes with public key
 - Do I need your private key to know if you send yes/no?
 - NO! Just need to encrypt “yes” with the public key myself, and compare
- Lesson:
 - Only encrypt random data with this system

Random

- Random <> unpredictable
 - Pseudo random number generators
 - Random
 - But predictable
 - Time seeding:
 - Not good for security
 - Produces same result starting with same time
 - Prediction of generation => very limited set of possibilities
- Secure random number generation
 - Needs true unpredictable data
 - Hard to get on computer
 - May have hardware support
 - “entropy” as measure

Base protocol concerns

- Man in the middle
- Replay attack
- Handshake integrity
- Challenge-Response
- Synchronous-asynchronous
- Permanent keys – temporary keys

Man in the middle

- Most annoying man in the world
- Two types
 - Passive: can only observe communication
 - Active: can interfere at any point in time
- Story: mig-in-the-middle
 - Groundstation challenges enemy aircraft
 - Enemy aircraft relays to enemy groundstation
 - Enemy groundstation challenges our aircraft
 - Our aircraft responds to enemy groundstation
 - Enemy groundstation relays to enemy aircraft
 - Enemy aircraft responds to our ground control
 - We let pass – we stupid – we death

The frustrating man in the middle

- Problem:
 - exchange a message via a 3rd party
 - 3rd party should not be able to read it
- Try 1: use symmetric key to protect the information
 - Send encrypted message: OK
 - Exchange key? Problem still exists...
- Try 2: use symmetric key to protect the information, use public key to protect the key
 - Send encrypted information: OK
 - Send encrypted key, need public key
 - Exchange public key? Problem still exists!!!
 - Give up?

Replay attack

- Suppose:
 - Shared secret for communication confidentiality
 - Uid/pw for authentication over secret channel
 - Message: buy 1000 Fortis shares
- Safe?
 - Depends
- Attack:
 - Capture all packets
 - Replay, and replay and replay
 - How many shares would you have bought???
- Lesson:
 - You do not need to know what is sent, as long as you can replay it and it works

Integrity

- Q: "Do not deny you killed her"
 - A: "I do not deny just like that, I have a witness"
- Truncate:
 - T: "I do not deny"
- Note: truncate may work even over encrypted channel (just drop the last blocks)
- SSL handshake
 - Cipher choice was not checked, could be modified
 - The weakest cipher was "none" ...

Challenge – response

- Ensures “fresh” negotiation
 - Against replay
 - Against man-in-the-middle
- Typically the server challenges
 - Challenger must generate new, fresh, always other challenges
 - Must avoid replay attacks
- The client must trust the challenger to challenge
 - Server is first authenticated
 - Only then are his challenges trusted

Replay attack on authentication

- Authentication system:
 - Send challenge
 - Inspect response
 - If ok, authenticated
- Attack:
 - Capture challenge-response pairs
 - Attempt login: if challenge seen before, replay response, otherwise abandon and back to capturing mode
 - Note: probably not counted as failed login!
- Lesson:
 - If an old response becomes alive again (same challenge, same response expected), you may be at risk

Synchronous-Asynchronous

- Different protocols: S/MIME, SSL
- Different properties
 - Set up secure session, use it
 - Send secured messages
- Asynchronous is more difficult
 - No negotiation
 - Can always be replayed (is self contained)

Permanent keys – temporary keys

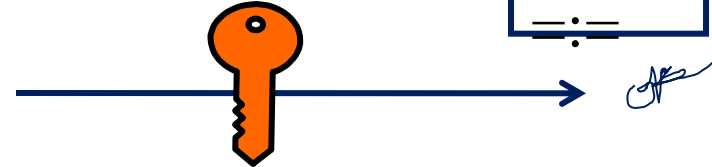
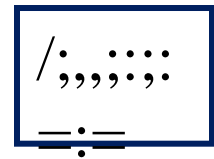
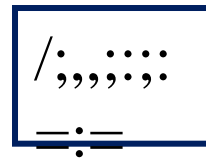
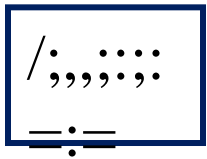
- Different life span
- Protocols establish temporary keys (most often)
 - Keys “loose” some information every time they are used
 - Use critical keys carefully, and infrequently
 - Use permanent keys to establish session keys

Public Key Cryptography

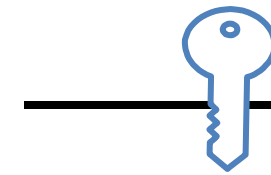
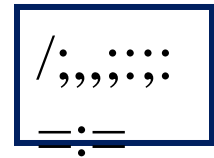
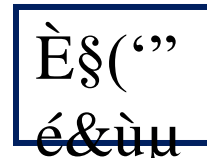
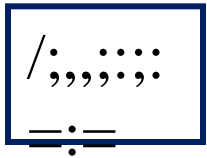
Signing

Sender

Receiver

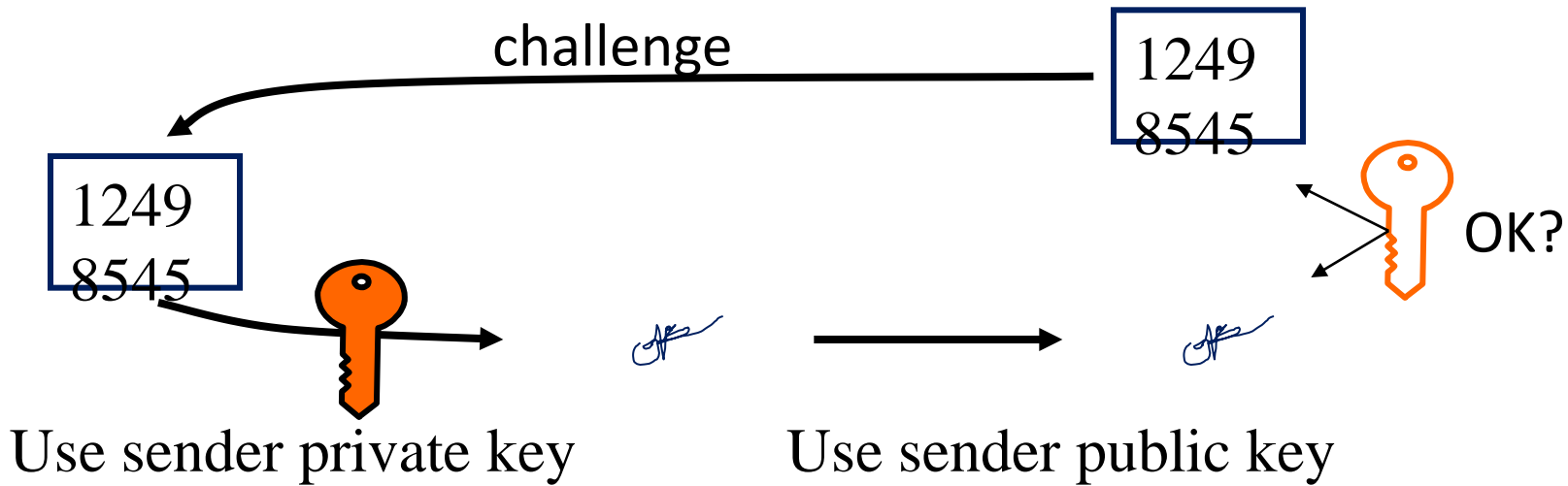


Encryption



	sender	receiver
public		
private		

Asymmetric – public key authentication



- Public part of keys are exchanged
- Server sends challenge
- Client signs challenge with private key: evidence
- Mutual authentication possible

Asymmetric – public key authentication

Properties:

- Even more complex exchange: even harder to integrate
- Validity of public keys must be verified: extra work
 - Out-of-band exchange, e.g. authorized key list
 - Transitive trust, e.g. PGP
 - PKI infrastructure: signed by Certification Authority
- Solves man-in-the-middle problem by external trust

PKI Operation & Infrastructure

- Certification Authority = TRUSTED third party:
- Secrecy of CA private key is paramount!
- Issuing and revocation process
- Certification Practice Statement (CPS):
procedures!
- Implementation possibilities:
 - Outsourced: e.g. Verisign, Globalsign
 - In-house:
 - High-end: Entrust, Baltimore
 - Low-end: Microsoft CA, OpenSSL

Base use cases

- Authentication
 - On-line
 - Asynchronous
- Key exchange
 - Confidentiality
 - On-line
 - Asynchronous
- Signature
 - Non-repudiation
 - Proof of origin
- Integrity
 - Signature

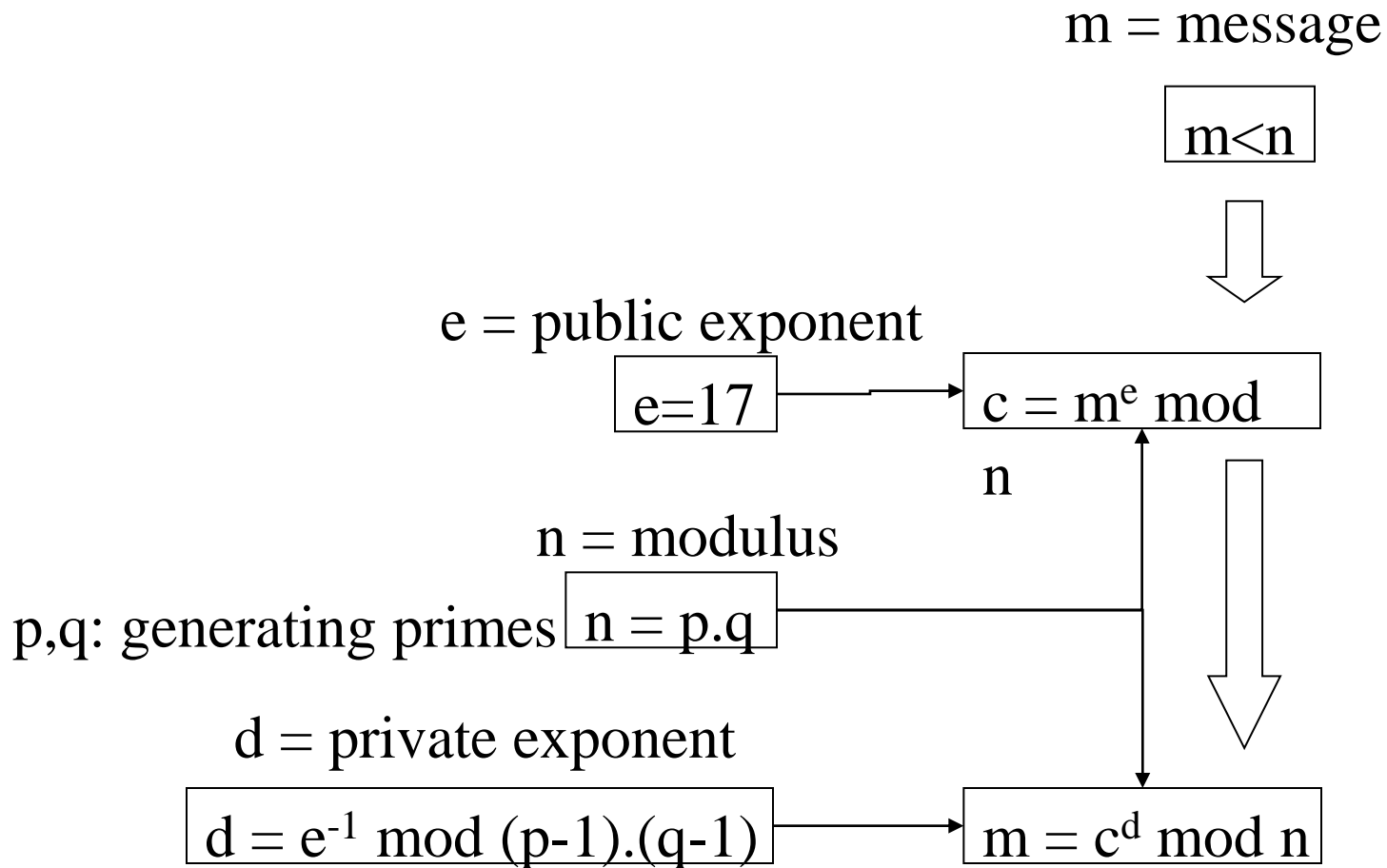
Authentication

- Symmetric keys:
 - P1: send Challenge
 - P2: compute $f(\text{challenge}, \text{id}, \text{secret})$
 - P2: Send $f()$, id
 - P1: $\text{secretid} = \text{db}(\text{id})$
 - P1: compute $f(\text{challenge}, \text{id}, \text{secretid})$
 - P1: verify $f() = f()$
- Asymmetric keys
 - P1: send challenge
 - P2: sign with private
 - P2: send signature & id
 - P1: check signature with $\text{public}(\text{id})$
- Note:
 - one possible protocol
 - Protocol design is difficult

Key exchange

- Diffie-helman
 - A sends $m_a = \text{power}(x,p) \bmod m$
 - B sends $m_b = \text{power}(x,q) \bmod m$
 - A computes $s_a = \text{power}(m_b,p) \bmod m$
 - B computes $s_b = \text{power}(m_a,q) \bmod m$
 - Result: $s_a == s_b$
- Private-public key based
 - A sends s_a encrypted with public b
 - B sends s_b encrypted with public a
 - A decrypts s_b with private a & computes $s = f(s_a, s_b)$
 - B decrypts s_a with private b & computes $s = f(s_a, s_b)$
- Note:
 - DH not resistant to man-in-the-middle
 - Private/public: just one possible protocol

RSA in a nutshell



Signature – Integrity

- Signature
 - Generate digest of the data that must be signed
 - Use your own private key to encrypt the digest of the data you want to sign
 - To check the signature
 - Compute digest of the data
 - use the public key of the signer to decrypt the signature, result is a digest
 - Check if the digests are the same
 - If the digest are the same, the signature is OK
- Integrity
 - Sign data
 - If the data is changed, the signature will not be valid

Need for an infrastructure

- To use public-private keys, the public keys must be exchanged
- They must be exchanged securely
- So, back to square one?

- “New” idea: use a trusted third party
 - This third party signs public key – identity associations
 - To check if a public key really is belonging to an identity, check the signed statement
 - Signed statement = certificate
 - To check the signature, use the public key of the third party
 - This is the only public key that is needed up front
 - Need to get that public key securely
 - Oops???

PKI infrastructure

- Certificate issuing third party = CA
- Support for multiple independent CAs
- CA signs subordinate certificates
- Certificate signature validation: via CA certificates
- Who signs CA certificates?
 - Cannot be “nobody”
 - So, sign the “root” CA with its own keys (pretty much the same as “no signature”)
- Which CAs are allowed?
 - Configuration
 - Root of trust

Some often occurring operations

- Generate a key
 - Symmetric key
 - Asymmetric key
- Encrypt data/decrypt data
 - Symmetric key
 - Asymmetric keys
- Sign data
 - Symmetric keys
 - Asymmetric keys

Generate a key

- Symmetric key
 - Smallest key 128 bits, otherwise insecure
 - Common sizes: 128, 168, 256
 - Need to be generated using a secure random number generator
 - No other processing needed
- Asymmetric key
 - Two main systems
 - RSA
 - Elliptic curve
 - Only RSA discussed here
 - Key size at least 1024 bits
 - Safe size is 2048
 - Keys are derived from two large prime numbers
 - Secrecy of these primes is crucial (not just resulting key)
 - Need a prime number generator for large primes
 - Need a test to check if prime (probabilistic test)
 - Uses infinite precision computations (large numbers)
 - Specific generators necessary

Encrypt/decrypt data - Symmetric key

- Block ciphers: use the key on data blocks
 - Each block encrypted independently (not so good): ECB mode
 - Cipher Block Chaining with Initialization Vector
 - Start with random block (size=key size)
 - Xor previous result block with new block
 - Encrypt
 - Repeat
 - Most common mechanism
 - Last data: padding to fill block

Encrypt/decrypt data - Asymmetric keys

- Both the secret and the public encryption are same operation:
 - Exponentiation module n
 - Public key = (n and public exponent (chosen to be easy to compute))
 - Private key = secret exponent (+ same n)
- Important:
 - Use only on random data, never directly!!!
 - Is not really encryption ...
 - Random data: symmetric key, digest, challenge, ...
 - So, how to encrypt other data?
 - Generate fresh symmetric key
 - Encrypt data with the symmetric key (using CBC)
 - Encrypt the symmetric key, then destroy it

Sign data

- Symmetric keys
 - The simple way: $\text{sign} = \text{digest}(\text{data}, \text{key})$
 - The better way: $\text{sign} = \text{hmac}(\text{data}, \text{key})$
 - $\text{Sign} = \text{digest}(\text{digest}(\text{data}, f1(\text{key})), f2(\text{key}))$
 - Much more robust against digest weaknesses
- Asymmetric keys
 - $D = \text{digest}(\text{data})$
 - Encrypt d with private key

X.509

- X.500 heritage
- LDAP-like naming
- LDAP
 - Tree structure
 - Nodes with attributes
 - Node name = unique path from top to node, containing “key” attribute values
 - Node name = DN (distinguished name)
 - Example: cn=andrem, o=inno.com, c=be
 - cn: common name
 - o: organization
 - c: country
- In X.509: tree is not used, typically, just attribute values

ASN.1

- Most common data format before XML
 - ITU-T X.680 / ISO 8824-1
- Abstract syntax notation
- Used
 - In X.*** standards (X.400, X.500)
 - in LDAP (because of link with X.500)
 - for SNMP
- Data formatting
 - For messages
 - For storage

DER/BER

- BER: basic encoding rules
 - ITU-T X.690 / ISO 8825-1
- Type-length-value encoding
 - Type
 - Basic types
 - Sequences
 - Length
 - Expanding opcodes
 - Short
 - Long
 - Sentinel based
- DER: distinguished encoding rules
 - Subset of BER
 - No sentinel, unique representation

Revocation:

- Why?
- How?
 - Certificate Revocation List (CRL)
 - Massive for larger CAs
 - Problem for both parties
 - Online Certificate Status Protocol (OCSP)
 - OCSP responder
- Revocation of root CA certificates: very difficult

Official CA

- Getting your CA Key into Browsers
- Data found on Internet:
 - Total cost: \$0.5M per browser
 - Netscape: Hand over the cash and a floppy
 - MSIE: No special charge, but you must pass an SAS70 electronic data security audit
 - US CPA Statement on Auditing Standards 70
 - Lengthy (up to 6 months), expensive, and painful
 - Infrastructure, policy, staff, and auditing costs run to \$0.5M

Certificate attributes

- Life expectancy lower as more attributes are added
- Too few is bad as well (eID)
- basicConstraints
- keyUsage
- extKeyUsage
- CRLDistributionPoints

Software – exercises

- Use “openssl”
- Versions exist for both unix and windows
 - <http://www.openssl.org/>
- Generate random data
- Generate symmetric and asymmetric keys
- Generate CA certificates
- Generate other certificates

Certificate types

- The private/public key system is always the same (essentially, lengths and algorithms may differ of course)
- But:
 - Keys should be used only for certain purposes
 - Need to have a way to indicate that
 - Solution: part of the certificate
 - CA decides on authorized uses
 - CA certificate
 - Subordinate CA certificate
 - SSL certificate
 - Server (HTTP, but also LDAP, SMTP, IMAP, ...)
 - Client
 - S/MIME certificate
 - Code signing certificate (activeX, java, ...)
 - IPsec

CRL – OCSP

- Problem statement
 - Certificate is signed statement
 - Can be copied, distributed, ...
 - Once created, no control anymore
 - Limitations:
 - Static, all inside the certificate
 - Time: Valid from, valid till
 - Use: attributes
 - But:
 - What if private key might have been copied?
 - What if the user loses his smartcard?
 - Need a way to signal these conditions
- Two mechanisms:
 - Certificate Revocation List
 - Online Certificate Status Protocol

CRL

- CAs maintain a list of revoked certificates
- CAs sign these lists: CRLs
- Users (relying parties) are expected to check against these lists
- How to obtain this list?
 - Location(s) are included in the certificate
 - Frequency is inside the CRL
- Obligation is
 - with the use to check
 - With the CA to generate CRLs on time
- CRLs can be BIG
 - For big CAs
 - To help: delta CRLs (incremental information)

OCSP

- Interactive synchronous request for status
- Signed response
- Required for highly critical use of certificates
- Need an on-line service, 24x7
- Used for eID

No CRL, no OCSP?

- May live without any of these, yet use certificate
- Application registration/de-registration
 - User is responsible for indicating which certificate he wants to use with you
 - Only check done: certificate expiration
 - If there is a problem with the cert, the user signals this to you
 - User is responsible for warning
 - Disadvantage for user: not just revoke cert, but also notify all depending services
 - Advantage for user: immediate effect

Key protection

- Always question key management
 - How is it stored?
 - Where is it stored?
 - How can it be safely used?
 - Is key recovery necessary and foreseen?
- Key protection
 - Physical: physical access is restricted
 - Key on USB stick, CD, and locked up in safe
 - Smartcard: private key never leaves it
 - Hardware security model, Tamper Resistant Device (HSM/TRD)
 - Logical:
 - Key is protected with another key (encrypted)
 - Other key is protected (example: Encryption key is password derived)
- Additional complexity:
 - Is the key still safe after being used?

Hardware protection

- Cheapest: USB stick/CD/DVD/...: portable memory
 - But can be copied in a fraction of a second
- Next: smartcard
 - Reasonably safe
 - Hardware protection: PIN build-in
 - Problems
 - Smartcards are not as safe as expected
 - Various analysis techniques
 - Depends on funds of attacker
 - Key in smartcard cannot be copied => availability risk
 - Solution: use keys protected with multiple smartcards
 - Any of them can decrypt the real secret

Hardware security modules

- Best protection
- FIPS 140-* rating
 - FIPS 140-2 Level 3 or FIPS 140-2 Level 4
 - (highest levels)
 - Certifies a level of security for the modules
- Costly
- Provides all features:
 - Multiple modules
 - In-system key use
 - Protection against many attacks, even physically opening the box
 - Provides key shares to operate the keys

	<i>Security Level 1</i>	<i>Security Level 2</i>	<i>Security Level 3</i>	<i>Security Level 4</i>
Cryptographic Module Specification	Specification of cryptographic module, cryptographic boundary, Approved algorithms, and Approved modes of operation. Description of cryptographic module, including all hardware, software, and firmware components. Statement of module security policy.			
Cryptographic Module Ports and Interfaces	Required and optional interfaces. Specification of all interfaces and of all input and output data paths.		Data ports for unprotected critical security parameters logically separated from other data ports.	
Roles, Services, and Authentication	Logical separation of required and optional roles and services.	Role-based or identity-based operator authentication.	Identity-based operator authentication.	
Finite State Model	Specification of finite state model. Required states and optional states. State transition diagram and specification of state transitions.			
Physical Security	Production grade equipment.	Locks or tamper evidence.	Tamper detection and response for covers and doors.	Tamper detection and response envelope. EFP or EFT.
Operational Environment	Single operator. Executable code. Approved integrity technique.	Referenced PPs evaluated at EAL2 with specified discretionary access control mechanisms and auditing.	Referenced PPs plus trusted path evaluated at EAL3 plus security policy modeling.	Referenced PPs plus trusted path evaluated at EAL4.
Cryptographic Key Management	Key management mechanisms: random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization.			
	Secret and private keys established using manual methods may be entered or output in plaintext form.		Secret and private keys established using manual methods shall be entered or output encrypted or with split knowledge procedures.	
EMI/EMC	47 CFR FCC Part 15. Subpart B, Class A (Business use). Applicable FCC requirements (for radio).		47 CFR FCC Part 15. Subpart B, Class B (Home use).	
Self-Tests	Power-up tests: cryptographic algorithm tests, software/firmware integrity tests, critical functions tests. Conditional tests.			
Design Assurance	Configuration management (CM). Secure installation and generation. Design and policy correspondence. Guidance documents.	CM system. Secure distribution. Functional specification.	High-level language implementation.	Formal model. Detailed explanations (informal proofs). Preconditions and postconditions.
Mitigation of Other Attacks	Specification of mitigation of attacks for which no suitable requirements are currently available.			

Software

- Common systems:
 - Key in code
 - Same key in dev, test, production: not conforming most guidelines
 - Key in configuration
 - Key in file with broad access needs: bad
 - Security configuration: depends on OS protection of configuration file
 - Passphrase at boot
 - Key not on system: good
 - No automatic restart: bad
 - Passphrase can be passed etc. (all bad things related to passwords)
 - Key server
 - System authenticates to key server, gets keys

Password derived key

- Passphrase is easier to remember than a key
- Must be long to have the same resistance as the actual keys
 - Dictionary: 32000 words (16 bits)
 - Reverse order: +1
 - First capital or not: +2
 - Substitute letters by ciphers: +4
 - Combine two words: *2
 - Total: 42 bits !!!
- So, how do you get to 128 bits!?
- Passphrase derivation:
 - Compute hash of password (MD5 = 128 bits, SHA1 = 160 bits)
 - Use salting to hide better (dictionary attack)
 - Hash(salt+password)

Key shares

- Create n shares
- M out of n are sufficient to operate (=reconstruct master key)
- Solution: polynomial function through n points of order $m-1$
 - Important point of the function: $(0, \text{secret})$
 - Generate random coefficients for other
 - Result: n points on polynomial
 - M points suffice to compute secret (evaluation of polynomial in point 0)

Example 2 out of 4

- Secret: 123 \Rightarrow (0,123)
- Random a: 2
- Points:
 - (1,125),(3,129),(4,131),(6,135)
- Use
 - Given (x_1, y_1) and (x_2, y_2)
 - Closed formula:
 - Secret = $(y_1 * x_2 - y_2 * x_1) / (x_2 - x_1)$

Public-Key Cryptography Standards

- From RSA
- PKCS standards:
 - <http://www.rsa.com/rsalabs/node.asp?id=2124>
- PKCS #7: Cryptographic Message Syntax Standard
- PKCS #10: Certification Request Syntax Standard
- PKCS #11: Cryptographic Token Interface Standard
- PKCS #12: Personal Information Exchange Syntax Standard