

Advanced Web Security

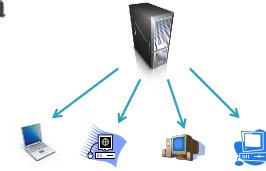
Data representation and PKI

EITN41 - Advanced Web Security

1

Representation of Data

- ▶ Many systems use the same data
- ▶ Systems have
 - Different architecture
 - Different OS
 - Different programs for reading/interpreting the data
- ▶ Data must be interpreted the same everywhere
- ▶ If we always know exactly what to expect it would not be much problem, but
 - Customized messages
 - Optional extensions
- ▶ Some language determining the rules is needed
 - ABNF
 - XML Schema
 - ASN.1



EITN41 - Advanced Web Security

2

Representing and Describing Data, ABNF

ABNF – Augmented Backus-Naur Form

- ▶ Describes some Internet protocols (HTTP, SMTP,...)
- ▶ Set of rules

^{rule name}
 Name = elements

One or more rulenames or terminal values

Examples in HTTP

- ▶ Version number

HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT

Terminal values

- ▶ HTTP request line

^{rule names} ^{Terminal values}
 Request-Line = Method SP Request-URI SP HTTP-Version CRLF
 Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | "CONNECT" | extension-method
 extension-method = token
 Request-URI = "*" | absoluteURI | abs_path | authority
 ...

See RFC 5234 for more details

- ▶ Ultimately, it all ends with terminal values

EITN41 - Advanced Web Security

3

Encoding ABNF

- ▶ Typically encoded using 7-bit ASCII

HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT
 encoded as

HTTP/1.1

and

Request-Line = Method SP Request-URI SP HTTP-Version CRLF
 Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | "CONNECT" | extension-method
 extension-method = token
 Request-URI = "*" | absoluteURI | abs_path | authority
 ...

can be encoded as

GET /index.htm HTTP/1.1

EITN41 - Advanced Web Security

4

XML Schema

- ▶ Another way of describing data is XML Schema
- ▶ Encoded as XML
 - XML Schema describes what a valid XML document looks like in a specific application or protocol
- ▶ Commonly used for user-defined data
 - Lots of applications in e.g., web services
- ▶ Encoding is very verbose

```
<xs:element name="phoneNumber">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="phoneID" type="xs:string"/>
      <xs:element name="number" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema

```
<phoneNumber>
  <phoneID>Mobile</phoneID>
  <number>0701234567</number>
</phoneNumber>
```

Encoding

- ▶ For both ABNF and XML schema, the encoded data is easily readable
 - But what about binary data?
 - Base64 can be used

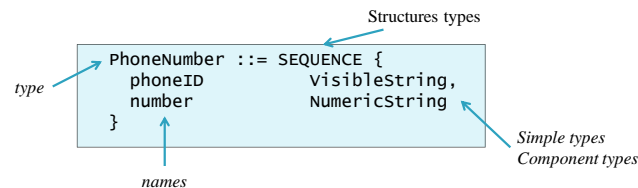
ASN.1

- ▶ ASN.1 – Abstract Syntax Notation One
 - Another way of describing rules
 - Widely used in telecommunications (GSM, 3G, LTE)
 - Used for certificates, encrypted/signed messages, keys etc (Main reason to focus on this here)
- ▶ Joint standard, ISO and ITU
- ▶ Built around types
- ▶ Similarities with programming languages
- ▶ Several versions during development
 - 1984, 1988, 1995, 2008
- ▶ Description and encoding are separated
 - ASN.1 for description/structure
 - BER, CER, DER, PER, XER,... for encoding

Example

- ▶ We want to send data regarding students at LTH between computers

First type definition



- ▶ Two main kinds of types
 - Simple types – does not have components
 - Structured types – has component types
- ▶ There are also tagged types and e.g., CHOICE which is another type.

Examples of Simple Types

Type	Description	Tag
BOOLEAN	A Boolean which can only take one of the values 0 or 1.	1
INTEGER	An integer of any size.	2
DATE	A date in the format YYYY-MM-DD.	
BIT STRING	Binary data that does not have to be a multiple of 8 bits.	3
OCTET STRING	Binary data that is a multiple of 8 bytes.	4
UTF8String	A string encoded with UTF-8, allowing the use of Unicode.	12
NumericString	A string of numbers 0-9 and the "space" character.	18
VisibleString	ASCII characters other than control characters.	26

- ▶ Each type has a pre-defined tag
 - Used for encoding

Structures Types (+ CHOICE)

Type	Description	Tag
CHOICE	A list of types and the value is one of the listed component types.	*
SEQUENCE	An ordered list of component types. The values must be given in the specified order.	16
SEQUENCE OF	Same as SEQUENCE but all components types have to be of the same type.	16
SET	An un-ordered list of distinct component types, i.e., the values can be of any order.	17
SET OF	An un-ordered list of a single component type, i.e., values can be in any order but they are always of the same type.	17

- ▶ Has component types
- ▶ Tag of CHOICE is the tag of the chosen component type

Expanding Example

```
Student ::= SEQUENCE {
    studentName    UTF8String,
    phone          SEQUENCE OF PhoneNumber
}

PhoneNumber ::= SEQUENCE {
    phoneID        VisibleString,
    number         NumericString
}
```

- ▶ One student can have several phone numbers
 - Structured type SEQUENCE OF is used
- ▶ Type "Student" is top-level type

Expanding Example

```
Student ::= SEQUENCE {
    studentName    UTF8String (SIZE(1..40)),
    phone          SEQUENCE OF SEQUENCE {
                        phoneID    VisibleString,
                        number      NumericString
                    }
}
```

- ▶ Explicit definition of phone number removed
- ▶ Restriction on size of string, called subtype

Object Identifiers

```
id-sha256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
    country(16) us(840) organization(1) gov(101) csor(3)
    nistalgorithm(4) sha256(1) }
```

- ▶ Globally unique identifier
 - Algorithms
 - Semantics meanings
 - Protocols
 - ...
- ▶ Global tree, similar to DNS in many ways
 - Responsibility for one subtree can be delegated
 - Distributed storage
- ▶ Branches are numbered from 0, and sometimes given a name
- ▶ OIDs used are assumed known to communicating systems

Expanding Example

- ▶ Keywords DEFAULT and OPTIONAL
 - DEFAULT gives default value if none is chosen
 - OPTIONAL says that it is optional to include data for this type
- ▶ Enumerated type
 - Give explicit semantic meaning to numbers (or the other way around if you wish)
- ▶ OID for a student – Should be in the LTH subtree

EITN41 - Advanced Web Security

13

```

Student ::= SEQUENCE {
  studentName    UTF8String (SIZE(1..40)),
  studentID      OBJECT IDENTIFIER,
  program        ENUMERATED {
    C(0), D(1), E(2), F(3), Pi(4)
  },
  phone          SEQUENCE OF SEQUENCE {
    phoneID      VisibleString,
    number       NumericString
  },
  status         StudentStatus,
  finishedCourses SEQUENCE OF SEQUENCE {
    courseInfo   course,
    grade        VisibleString
  }
}
StudentStatus ::= SEQUENCE {
  hp          INTEGER,
  avgGrade    REAL (3.00..5.00),
  active       BOOLEAN DEFAULT TRUE,
  comment      UTF8String OPTIONAL
}

```

EITN41 - Advanced Web Security

14

Modules, Importing, Exporting

```

Student-module
{ ... }
DEFINITIONS
  AUTOMATIC-TAGS ::=
BEGIN
  EXPORTS Student;

  IMPORTS Course FROM LTH-module { ... };

  Student ::= SEQUENCE {
    ...
  }
END

```

- ▶ Import definitions from other modules
- ▶ Allow exporting definitions in this module
 - If EXPORTS is not used, everything can be exported

EITN41 - Advanced Web Security

15

Encoding

- ▶ ASN.1 can be used to structure the data and inform about what type of data it is
- ▶ Actual encoding of data is a separate process
 - Encoding method can be chosen suitably
- ▶ Type of encoding must be agreed upon or predetermined by protocol



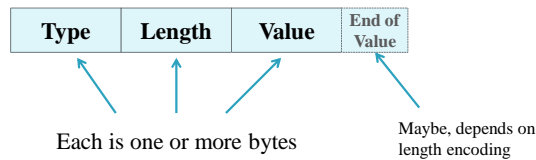
- ▶ ASN.1 has more encoding flexibility than ABNF and XML Schema
 - It can even use XML

EITN41 - Advanced Web Security

16

BER Encoding

- Basic Encoding Rules
 - Original encoding
- More efficient than text based encodings
- Important characteristic: Many encodings are possible for the same data (but decoding is unique)
- Type-Length-Value (TLV)
 - Also called Tag-Length-Value

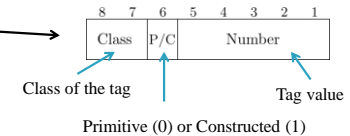


EITN41 - Advanced Web Security

17

BER, Encoding the type

- Type defines what is encoded (Boolean, Integer, Sequence, UTF8String etc)
- Identifier byte
 - Class
- P/C
 - Primitive: value is the value of the type
 - Constructed: value is a set of TLVs (e.g., a Sequence)
- Number is tag value
- Examples:
 - SEQUENCE: 0x30
 - BOOLEAN: 0x01



Larger tag numbers are possible (see notes)



EITN41 - Advanced Web Security

18

BER, Encoding the Length

- Short Definite Form 0xxxxxxx Max length of value = 127 bytes
- Long Definite Form
 - 1xxxxxxx xxxxxxxx ----- xxxxxxxx
 - Number of bytes that gives the length
- Indefinite Form
 - 10000000 value 00000000 00000000
 - Can only be used for encoding of constructed types



EITN41 - Advanced Web Security

19

BER, Encoding the Value

- Encoding of value will depend on the value
 - Booleans: TRUE is non-zero, FALSE is zero
 - TRUE can be TLV encoded as 01 01 FF
 - Integers: Encoded as two-complement with least number of bytes necessary
 - 2 can be TLV encoded as 02 01 FE
 - VisibleString is encoded as the ASCII representation
 - Can be split into segments with an outer constructed TLV
 - "string" can be TLV encoded as 1A 06 73 74 72 69 6e 67
 - ...or as 3A 80 1A 03 73 74 72 1A 03 69 6e 67 00 00
 - T L T L V T L V



EITN41 - Advanced Web Security

20

Tagging

- ▶ We are returning to ASN.1 now
- ▶ Built-in types have pre-defined tags
 - Not always enough

```
Dessert ::= CHOICE {
  a      [0] INTEGER
  b      [1] INTEGER
  c      [2] INTEGER
}
```

- ▶ Explicit tagging (default) – Adds outer tagging environment
 - Seen as structured type T L T L V
 - Read as "in addition to"
- ▶ Implicit tagging – changes the default tag
 - Read as "instead of"
 - Default can be changed to implicit
- ▶ Tagging can be automated (AUTOMATIC-TAGS)
 - Tagging determined by rules (start with 0 and increment)

EITN41 - Advanced Web Security

25

CMS

- ▶ Cryptographic Message Syntax
- ▶ Standard for representing signed, hashed and/or encrypted messages
- ▶ Originates in PKCS 7 – now IETF standard (RFC 5652)
- ▶ Uses BER as encoding rules (mostly)
- ▶ Used in e.g., S/MIME (encrypted/signed emails)
- ▶ *ContentInfo* is the top level type

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICIT ANY DEFINED BY contentType
}

ContentType ::= OBJECT IDENTIFIER
```

EITN41 - Advanced Web Security

26

CMS, ContentTypes

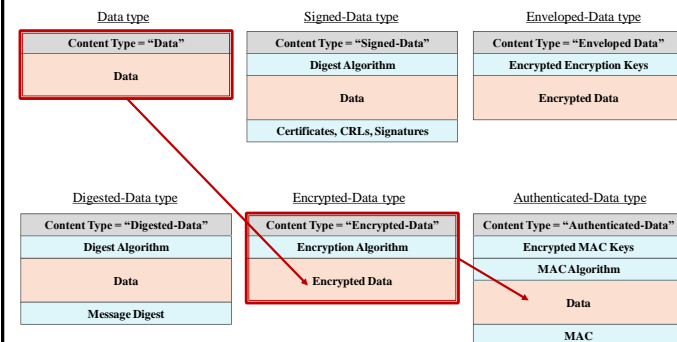
6 different types (can be nested)

- ▶ **Data Type** – String of bytes
- ▶ **Signed-Data Type** – Data with digital signature(s)
- ▶ **Enveloped-Data Type** – Encrypted data and an encrypted encryption key
 - Several recipients → encryption key is encrypted for each recipient
- ▶ **Digested-Data Type** – Data with a message digest
- ▶ **Encrypted-Data Type** – Encrypted data without encrypted encryption key
- ▶ **Authenticated-Data Type** – Data with a MAC of the data and an encrypted authentication key
 - Several recipients → authentication key is encrypted for each recipient

EITN41 - Advanced Web Security

27

CMS, ContentTypes



EITN41 - Advanced Web Security

28

SignedData Type

- ▶ Data which is digitally signed
 - Several signers for same data is possible

```
SignedData ::= SEQUENCE {
  version          CMSVersion,
  digestAlgorithms SET OF DigestAlgorithmIdentifier,
  encapContentInfo SEQUENCE {
    eContentType ContentType,
    eContent [0] EXPLICIT OCTET STRING OPTIONAL },
  certificates [0] IMPLICIT CertificateSet OPTIONAL,
  crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
  signerInfos     SET OF SignerInfo
}
```

SignerInfo Type

- ▶ Gives information about each signer

```
SignerInfo ::= SEQUENCE {
  version          CMSVersion,
  sid              SignerIdentifier,
  digestAlgorithm  DigestAlgorithmIdentifier,
  signedAttrs [0]  IMPLICIT SignedAttributes OPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature         SignatureValue,
  unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier
}
```

PKCS #12 – Personal information

- ▶ Represent personal information
- ▶ Uses parts of CMS and adds additional features
- ▶ Four possibilities for protection:

Public key

1. *Public key privacy mode* – encrypt content with symmetric key, encrypt key with public key
2. *Public key integrity mode* – Digital signature

Symmetric key

1. *Password privacy mode* – Encrypt with symmetric key derived from password
2. *Password integrity mode* – Use MAC with key derived from password

PKCS #12

- ▶ Top-level type: Personal Information Exchange (PFX)

```
PFX ::= SEQUENCE {
  version          INTEGER,
  authSafe         ContentInfo,
  macData          MacData OPTIONAL }

```

- ▶ MacData only used for Password Integrity Mode

```
MacData ::= SEQUENCE {
  mac              DigestInfo,
  macSalt          OCTET STRING,
  iterations       INTEGER DEFAULT 1 }

```

- ▶ ContentInfo is either
 - SignedData – Public Key Integrity Mode
 - Data – Password Integrity Mode
- Content here is called **AuthenticatedSafe**

PKCS #12

▶ AuthenticatedSafe

```
AuthenticatedSafe ::= SEQUENCE OF ContentInfo
```

▶ ContentInfo

- EnvelopedData – Public-Key Privacy Mode
- Encrypted Data – Password Privacy Mode
- Data – no encryption

▶ Content is a sequence of SafeBags

- **KeyBag** – A PKCS #8 private key.
- **PKCS8ShroudedKeyBag** – A PKCS #8 encrypted private key.
- **CertBag** – A certificate.
- **CRLBag** – A Certificate Revocation List.
- **SecretBag** – A personal secret.
- **SafeContents** – Allows for nesting the other types instead of just putting them in a sequence.

PKI

▶ A public key infrastructure includes all aspects of certificates

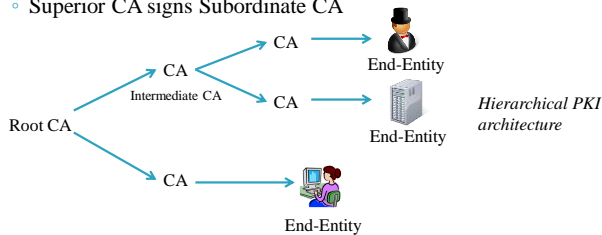
- Who signs
- How they are signed
- How they are revoked
- How they are stored
- How keys are generated

▶ Different profiles can have different rules

- PKIX – uses X.509 certificates (Internet PKI profile)
- Many countries have their own profile

PKI Participants

- ▶ End-Entity – User/application where the private key cannot be used to sign certificates
- ▶ CA (Certification Authority) – Issues certificates
 - Root CA is on top
 - Superior CA signs Subordinate CA



PKI Participants, optional

▶ RA (Registration Authority)

- Does not sign certificates
- CA can delegate management functions to an RA
 - Verify identity
 - Assigning names, generating keys, storing keys, reporting revocation information

▶ CRL Issuer

- Certificate Revocation Lists are used to revoke certificates
- Typically, the CA issues a CRL
- CA can delegate CRL issuing to another party

Certificate Requests

- Two main variants: PKCS#10 and CRMF

```

CertReqMsg ::= SEQUENCE {
  certReq      CertRequest,
  popo         ProofOfPossession OPTIONAL,
  regInfo      SEQUENCE SIZE(1..MAX) of AttributeTypeAndValue OPTIONAL
}

CertRequest ::= SEQUENCE {
  certReqId    INTEGER,
  certTemplate CertTemplate,
  controls     Controls OPTIONAL
}

```

CRMF

- ID maps request ↔ response
- Template* are the fields in the certificate that are filled in by requester
 - Public key, subject name, some extensions
- Controls can be used to e.g., authenticate the requester
- Proof of possession allows requester to prove ownership of private key
 - Many ways possible, e.g., signature on request (should depend on key usage)
- RegInfo can have additional information (contact info, billing info etc)

EITN41 - Advanced Web Security

37

X.509 Certificate

```

Certificate ::= SEQUENCE {
  tbsCertificate TBSCertificate,
  signatureAlgorithm AlgorithmIdentifier,
  signatureValue  BIT STRING
}

TBSCertificate ::= SEQUENCE {
  version          [0] EXPLICIT Version DEFAULT v1,
  serialNumber     CertificateSerialNumber,
  signature        AlgorithmIdentifier,
  issuer           Name,
  validity         validity,
  subject          Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo,
  issuerUniqueID   [1] IMPLICIT UniqueIdentifier OPTIONAL, -- v2 or v3
  subjectUniqueID  [2] IMPLICIT UniqueIdentifier OPTIONAL, -- v2 or v3
  extensions       [3] EXPLICIT Extensions OPTIONAL -- v3
}

```

same

- signatureAlgorithm determines the structure of the signatureValue BIT STRING

EITN41 - Advanced Web Security

38

Extensions

```

Extension ::= SEQUENCE {
  extnID      OBJECT IDENTIFIER,
  critical    BOOLEAN DEFAULT FALSE,
  extnValue   OCTET STRING
}

```

- OID to define the extension
- Critical determines how important the extensions is
 - Applications must not process unrecognized critical extensions

Some Extensions

- Authority key identifier (non-critical)* – identify issuer's public key if it has several
- Key usage (critical)* – Specify purpose of public key (separating keys gives some damage control)
 - Data encryption
 - Signatures
 - Data
 - Certificates
 - CRLs
- Subject (Issuer) Alternative Name (critical or non-critical)* – Extra name for subject (issuer)
- Basic Constraints (critical or non-critical)* – If certificate is CA

EITN41 - Advanced Web Security

39

Representing a Public Key

- SubjectPublicKeyInfo given by

```

SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm      AlgorithmIdentifier,
  subjectPublicKey BIT STRING
}

```

- The BIT STRING is the DER encoding of the public key specified by the algorithm OID
- Example, RSA

```

RSAPublicKey ::= SEQUENCE {
  modulus          INTEGER, -- n
  publicExponent   INTEGER -- e
}

```

- ASN.1 encoding defines how integers are represented

EITN41 - Advanced Web Security

40

Representing a Private Key

- ▶ Not in certificates

```

RSAPrivateKey ::= SEQUENCE {
    version          Version,
    modulus          INTEGER, -- n
    publicExponent   INTEGER, -- e
    privateExponent  INTEGER, -- d
    prime1           INTEGER, -- p
    prime2           INTEGER, -- q
    exponent1        INTEGER, -- d mod (p-1)
    exponent2        INTEGER, -- d mod (q-1)
    coefficient       INTEGER, -- (inverse of q) mod p
    otherPrimeInfos  OtherPrimeInfos OPTIONAL
}

```

- ▶ In theory, only privateExponent and modulus needed to decrypt and sign
 - Other values can be used to speed up decryption/signing
- ▶ This layout will be used in the home assignment

Certificate Revocation List (CRL)

- ▶ List certificates that are no longer valid
 - Expired certificates are not listed
- ▶ Maintained by the CA, but can delegate to CRL issuer
- ▶ Should be updated often
- ▶ **Scope of CRL:** The category of certificates that the CRL should cover
 - Can depend on CA, subjects, reason for revocation etc
- ▶ **Complete CRL:** All revoked certificates within scope are included
- ▶ **Indirect CRL:** Contains revoked certificates issued by some other CA
- ▶ **Delta CRL:** Contains certificates revoked after issuance of some complete CRL (**base CRL**)

CRL

- ▶ ASN.1 description of a CRL

```

CertificateList ::= SEQUENCE {
    tbsCertList      TBSCertList,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue    BIT STRING
}

TBSCertList ::= SEQUENCE {
    version          Version OPTIONAL,
    signature         AlgorithmIdentifier,
    issuer            Name,
    thisUpdate        Time,
    nextUpdate        Time OPTIONAL,
    revokedCertificates SEQUENCE OF SEQUENCE {
        userCertificate      CertificateSerialNumber,
        revocationDate       Time,
        crlEntryExtensions   Extensions OPTIONAL
    } OPTIONAL,
    crlExtensions         [0] EXPLICIT Extensions OPTIONAL
}

```

Extensions

CRL entry extensions

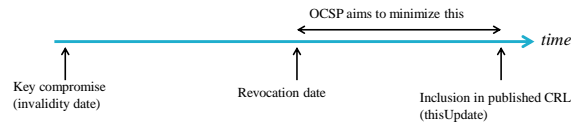
- ▶ *Reason code* – Why the certificate was revoked
- ▶ *Invalidity date* – When did the certificate become invalid
- ▶ *Certificate Issuer* – Give the CA of the certificate (useful for indirect CRLs)

CRL extensions

- ▶ *CRL number (non-critical)* – sequence number for CRL
- ▶ *Delta CRL Indicator (critical)* – Points to CRL number of the base CRL (used for delta CRLs)
- ▶ *Freshest CRL (non-critical)* – Points to a delta CRL (used for complete CRLs)

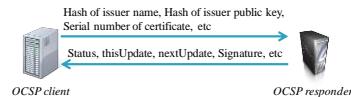
OCSP

- ▶ Online Certificate Status Protocol
- ▶ Alternative to CRL



- ▶ Compromised private keys can have huge consequences in e.g., financial transactions

- ▶ Nonce can optionally be included
- ▶ Request can be signed
- ▶ Status = good/revoked/unknown

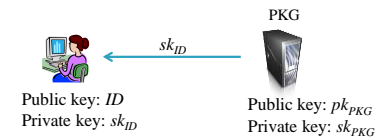


EITN41 - Advanced Web Security

45

Avoiding PKI

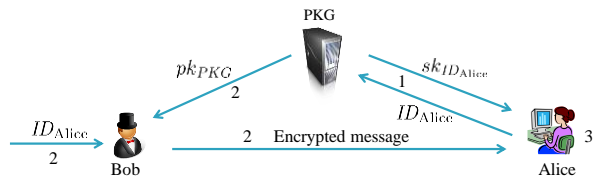
- ▶ Is it possible to have asymmetric cryptography without an underlying PKI?
- ▶ Identity Based Cryptography (IBC)
 - Identity Based Signatures (IBS)
 - Identity Based Encryption (IBE)
- ▶ Use identity as public key
 - Can be email address or some other identifying info
- ▶ Use Private Key Generator (PKG)



EITN41 - Advanced Web Security

46

Identity Based Encryption

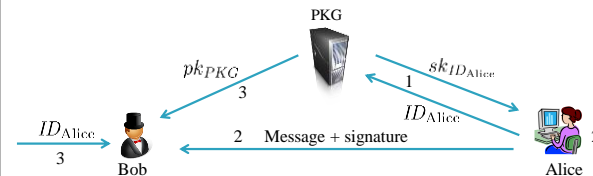


1. Alice proves she owns ID and receives her private key
 2. Bob uses Alice's ID and PKG's public key to encrypt message to Alice
 3. Alice decrypts using her private key
- ▶ Steps 1 and 2 can be interchanged

EITN41 - Advanced Web Security

47

Identity Based Signatures



1. Alice proves she owns ID and receives her private key
2. Alice uses her private key to sign message
3. Bob verifies signature using PKG's public key and Alice's ID

EITN41 - Advanced Web Security

48

Properties of IBC

Advantage

- ▶ No need for complete PKI – just have to trust the PKG

Drawbacks

- ▶ Not possible to revoke IDs (or at least difficult)
 - How do we revoke an email address, personnummer, fingerprint?
 - Possible to add time stamps to ID (structure and rules must be known to users)
- ▶ PKG has access to private key (called key escrow)
 - Problems with non-repudiation
 - Also a feature – the PKG can help to decrypt

Other Solutions

- ▶ *Certificate based encryption*
 - Certificate needed in order to decrypt
 - Revoked certificate → not possible to decrypt
 - Not necessary to check public key before encryption
- ▶ *Certificateless encryption*
 - PKG creates half private key, user creates half
 - No key escrow
- ▶ See references in lecture notes for more details