Practical Aspects of Modern Cryptography

Winter 2011

Josh Benaloh
Brian LaMacchia

Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II

Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II

Message-Based Protocols

- "Session" vs. "Message"
  - Synchronous vs. Asynchronous
- In message-based protocols, we cannot assume we have the luxury of being able to negotiate ciphersuites, parameter values, etc.
- In the common scenario, each message is a "fire-and-forget" communication
  - Each message has to contain enough information to allow the recipient to decrypt it.

Message-Based Protocols

- There are lots of message-based protocols
  - Examples: RPC, routing table updates
- The most common scenario to date, though, is e-mail
  - Digitally signed for sender authentication and integrity protection
  - Encrypted for confidentiality

Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II

S/MIME

- Secure Multipurpose Internet Mail Extensions
- Initially designed by RSA-led vendor consortium in 1995
- S/MIME messaging and S/MIME certificate handling are Internet RFC’s
  - Widely supported format for secure e-mail messages
  - Uses X.509v3 certificates

Scenario Assumptions

- Each participant has two public-private key pairs: one for signing messages and one for receiving encrypted messages from others
  - “Separation of duty” – separate keys (with separate controls) for separate uses
  - Encryption key archival/escrow/recovery
- For now, we assume key distribution isn’t a problem for participants
  - If I want to send you a message, I can obtain a copy of your encryption public key that I trust.
  - If you want to verify a message I signed, you can obtain a copy of my public signing key that you trust.

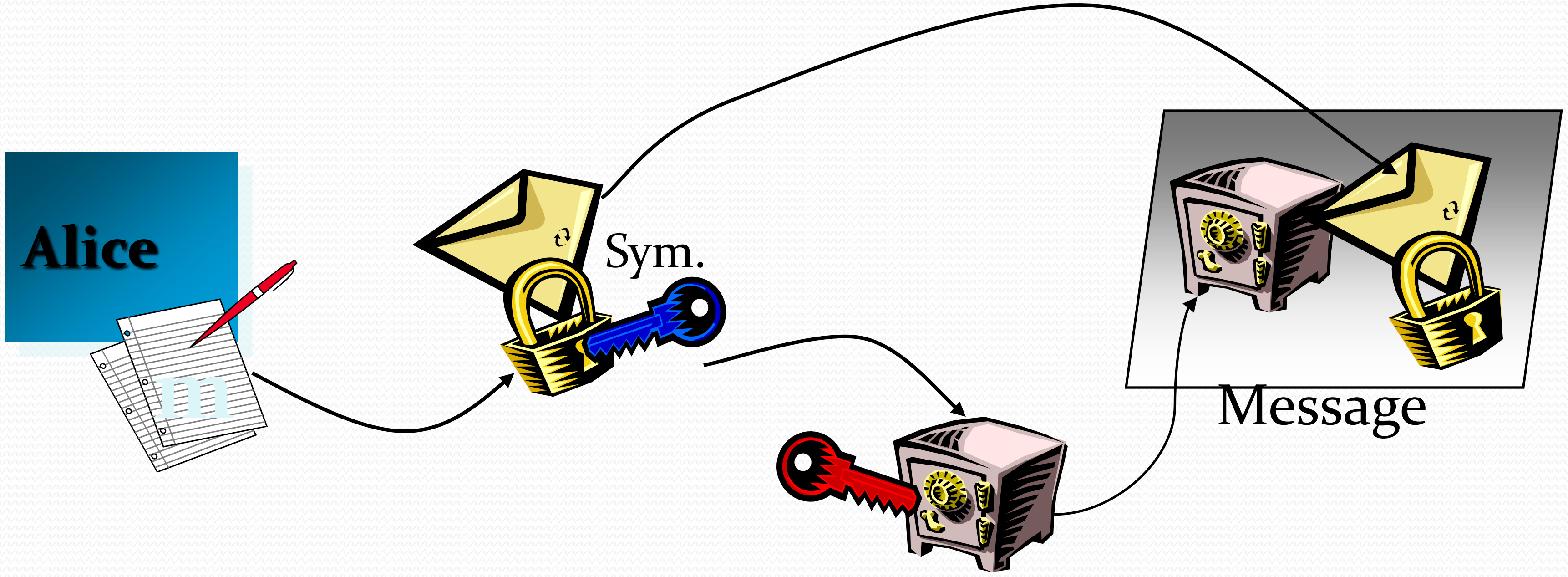
Encrypting Messages

- How do we want to encrypt messages?
- We have public keys for recipients, so we could repeatedly apply PK-encryption to portions of the message
  - Recall that we can only RSA-encrypt messages M with $|M| \leq |n|$
  - Plus, public key encryption is relatively slow, so we'd like to use it efficiently
- Idea: use PK to convey a random symmetric "session" key to recipients

Encrypting Messages

- We use symmetric encryption with randomly-generated session keys to encrypt message bodies
  - Since symmetric encryption is fast and messages may be arbitrarily large
- We use public-key encryption to encrypt the session keys to message recipients
- We send both encrypted message and session key as a unit to recipients…

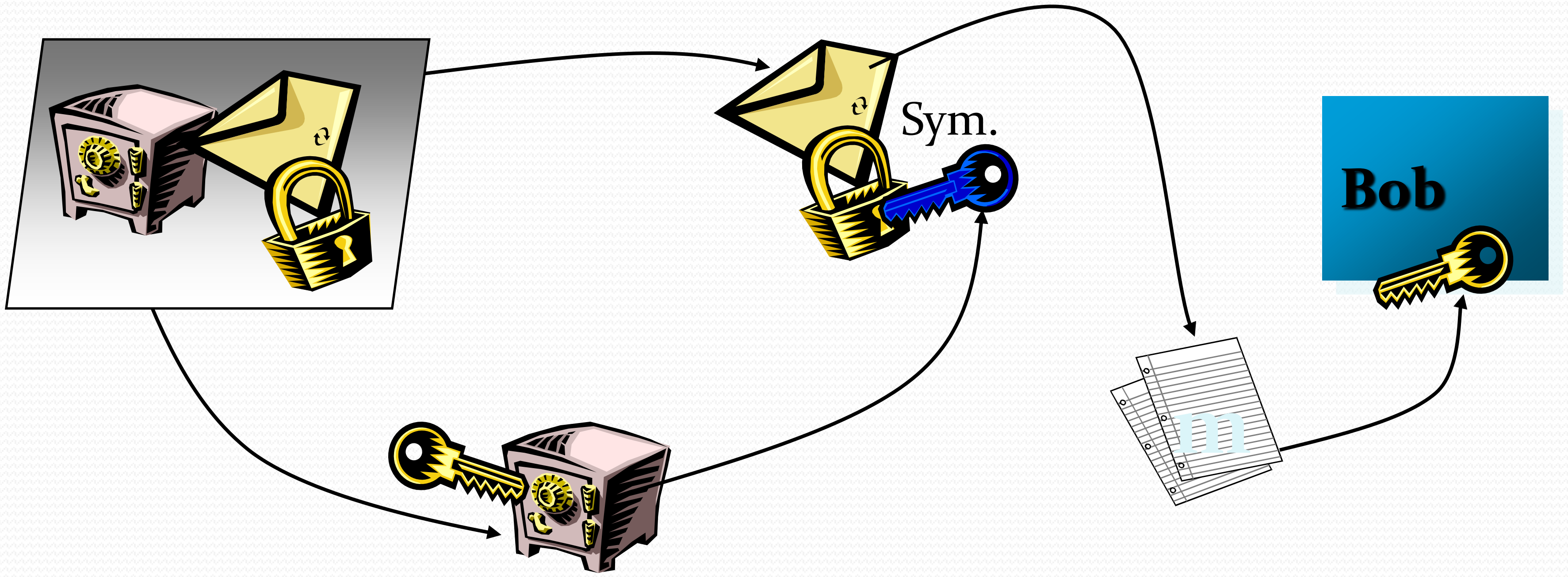
Message Encryption



Decrypting Messages

- Message decryption is just the reverse from encryption
- Recipients use their private encryption key to decrypt the session key for the message
- Recipients then use the session key to symmetrically decrypt the message body.

Message Decryption



Signing Messages

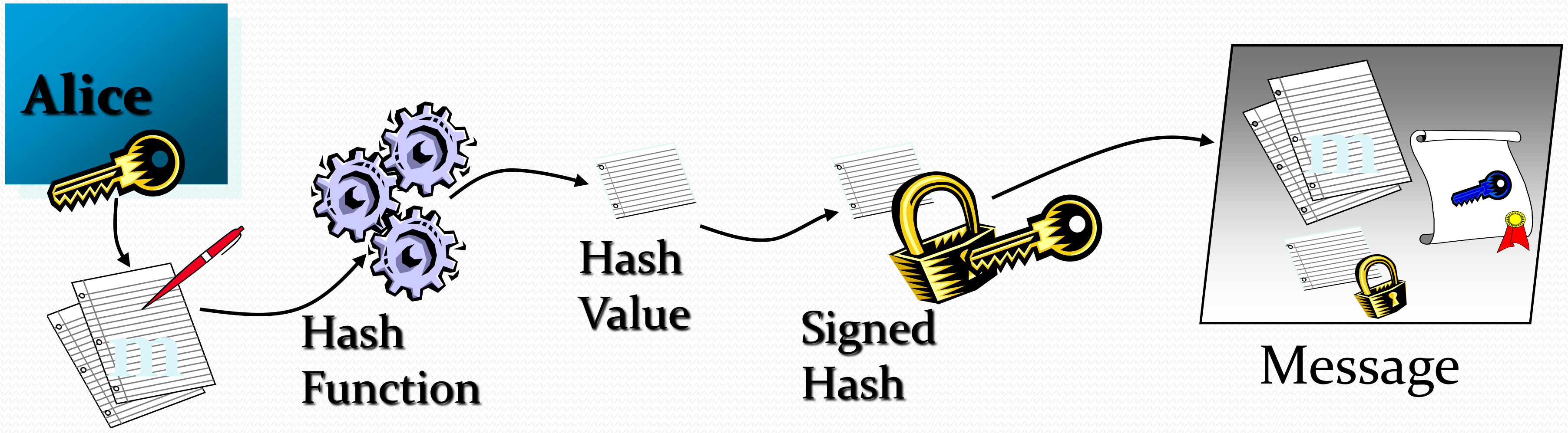
- How do we want to sign messages?
- Each user has a signing key pair, but again we can only sign values that are at most the same size as our signing public key modulus
  - So we can't sign the entire message directly, and repeated signing of parts of the message would open us up to attacks
- Idea: Sign a hash of the message

Signing Messages

- To sign a message, we first choose a cryptographic hash function H() to use with our signature algorithm
  - Normally defined as part of a signing ciphersuite
- We apply the hash function H to the exact sequence of bytes that forms our message (usually including header info)
- We sign the hash value
- We append the signed hash value to the message.

Digital Signatures

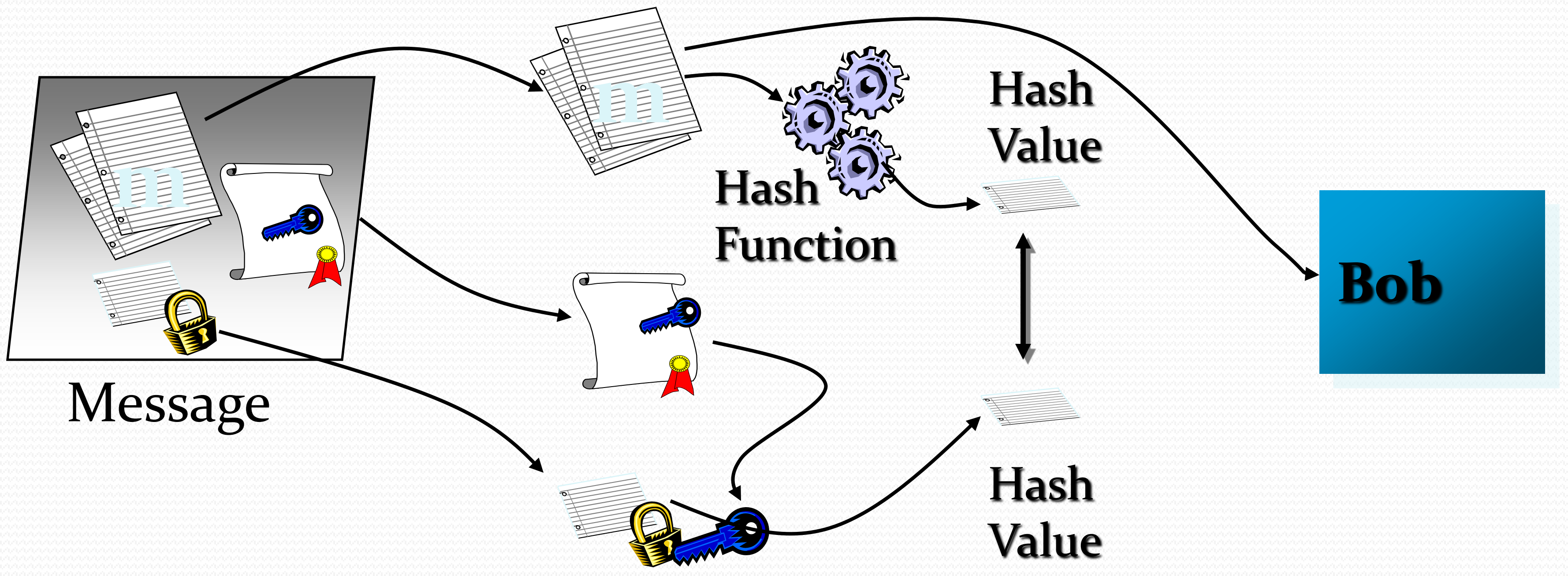
Provide Authentication and Integrity



Verifying Signatures

- To verify a signed message, the recipient has to do three things:
  - Independently compute the hash value of the signed portion of the message
  - Verify that the signature on the message came from the sender (by applying the sender's public signing key)
    - This yields the hash value signed by the sender
  - Compare the independently-computed hash value with the one the sender signed
- If the hash values are equal, then the message has not been modified since it was signed.

Verifying Signatures

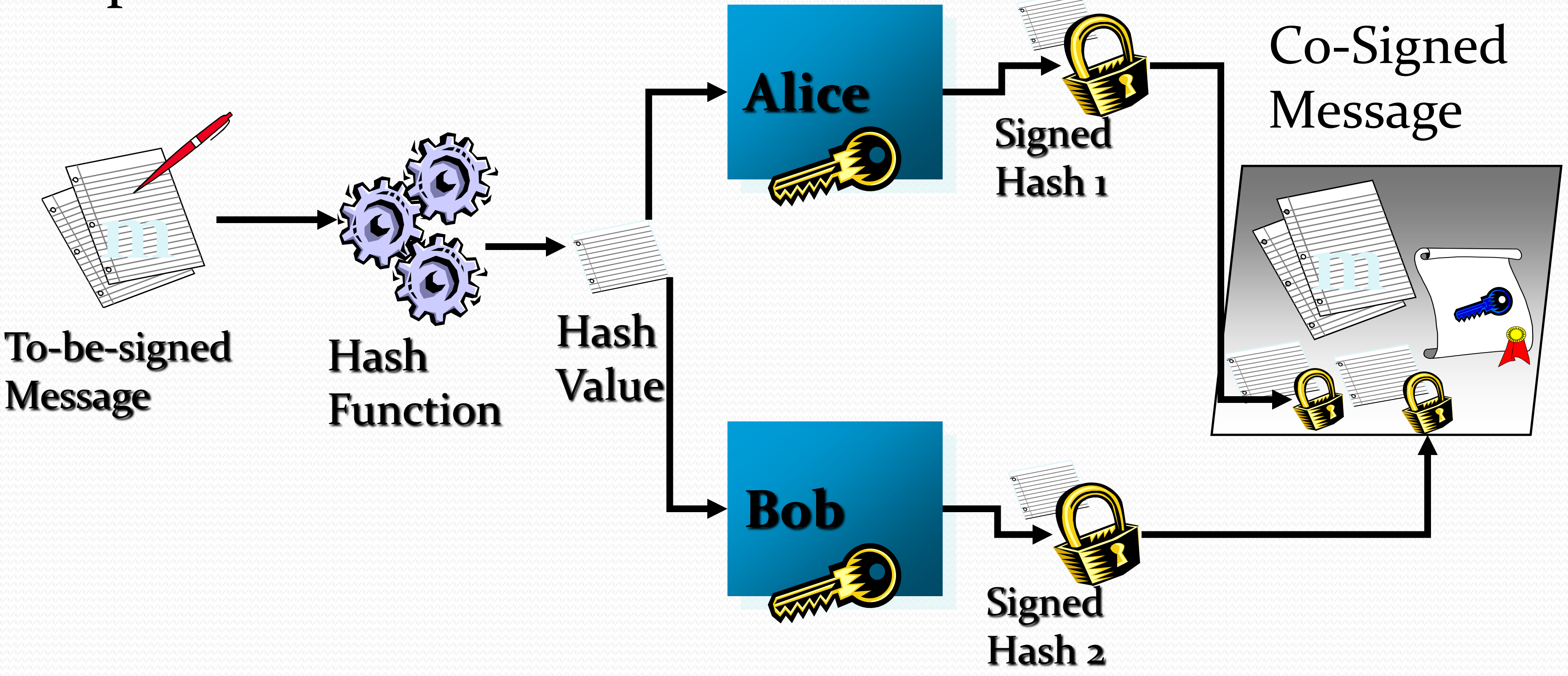


More Complex Signatures

- A single signer acknowledging understanding or commitment to different concepts or agreements within one document.
- Multiple signers signing unique content within the same document.
- Multiple signers “co-signing” the same content within the same document.
- Multiple signers, one signing content the other “counter-signing” the prior signature.

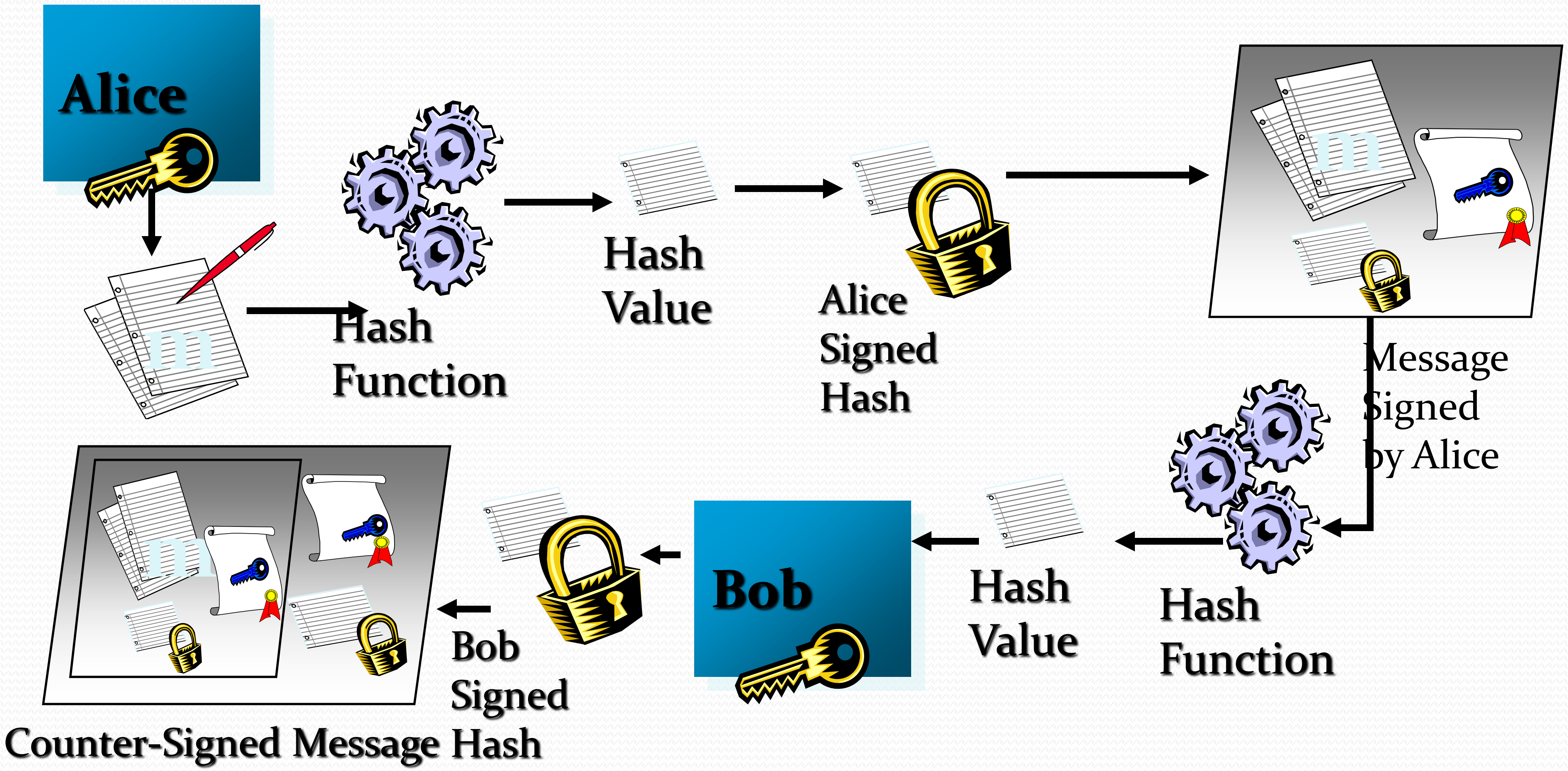
Co-Signing

- Alice and Bob want to sign the same message "in parallel"

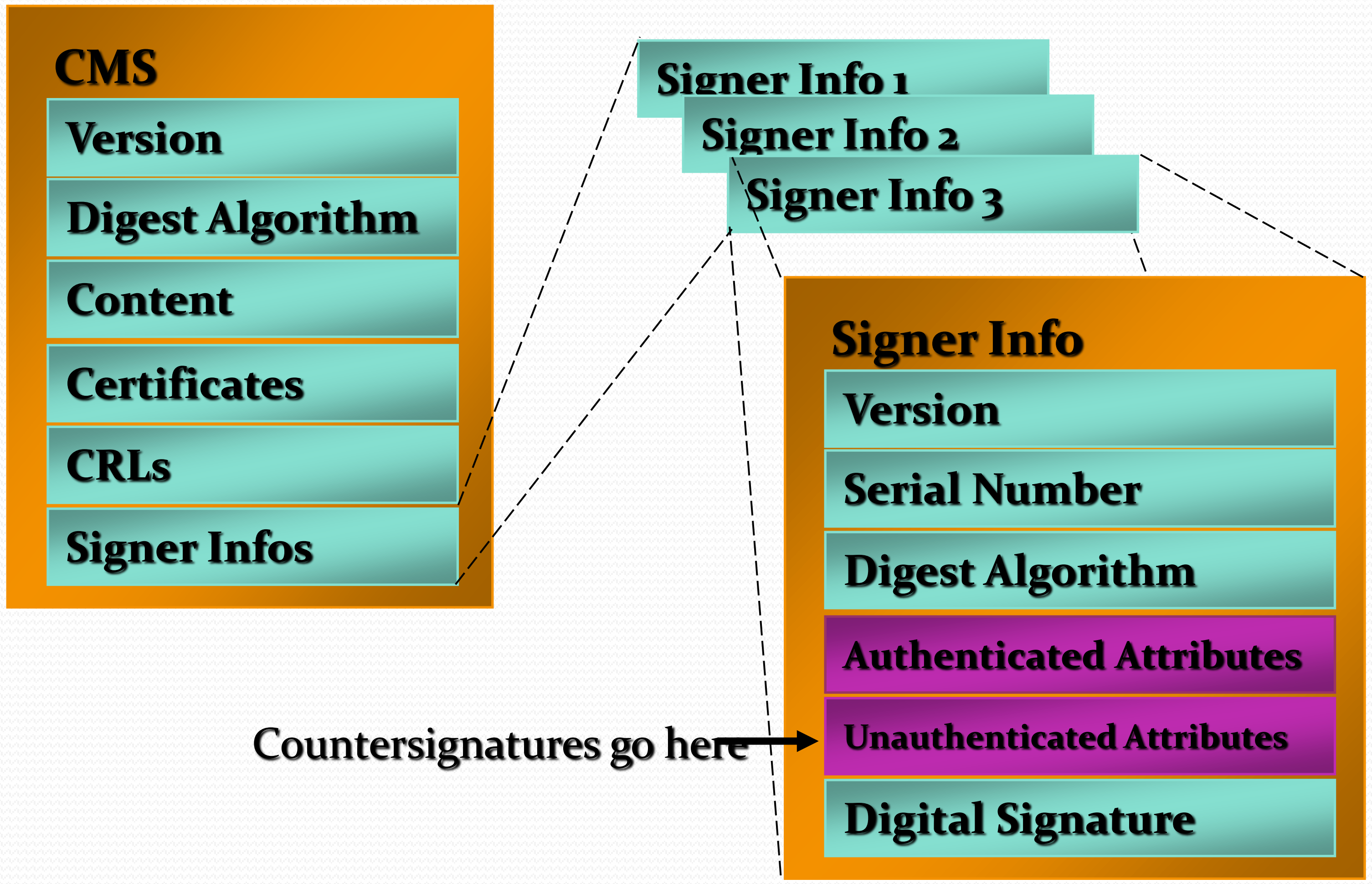


Counter-Signing

- Alice and Bob want to sign the same message "in series" (Alice first, then Bob)



PKCS #7/CMS Structure



Limitations of the CMS format

- The CMS standard only covers “wrapped” signatures
  - Signatures where the signed content is enclosed by the signature object
- Signing assumes you start with a bytestream that is completely immutable
  - This is the safest assumption, but sometimes it’s overly conservative
  - Example: CR-LF rewriting and tab/whitespace conversions for text.

Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II

What is XML?

```
<Address>
   <Street>1 Microsoft Way</Street>
   <City>Redmond</City>
   <State>WA</State>
   <ZipCode>98052</ZipCode>
</Address>
```

What is XML?

- XML is a W3C standard for describing “markup languages”
  - XML == “eXtensible Markup Language”
- Had its roots in SGML (of which HTML is an offshoot)
- Now, though, XML has really become a standard means of representing data structures in text.
  - “XML provides a text-based means to describe and apply a tree-based structure to information.” -- Wikipedia

Securing XML

- As XML's popularity grew, so did the need to secure XML objects (trees of XML elements)
- How should we sign & encrypt XML?
- One possibility: just treat an XML object as a byte sequence and use S/MIME
    - It's just a sequence of characters, so we can Unicode encode that sequence, hash it, encrypt it and wrap it in S/MIME

Securing XML

- Using S/MIME works, but it has some drawbacks:
  - The result of signing or encrypting an XML object is now some binary blob, not an XML object, so signing & encrypting this way doesn't "play nice" with the XML ecosystem
  - An XML object isn't a piece of text – that text is just a representation of the object
    - There are many equivalent representations of an XML object
  - There are semantically-neutral transforms allowed on XML representations that should not break signatures.

Signing & Encrypting XML

- Thus, there was a need to develop a standard for signing & encrypting XML objects
  - July 1999: work began on XMLDSIG, a standard for signing XML objects and representing signatures as XML
  - Summer 2000: work began on XMLENC, a standard for encrypting data and representing the ciphertext and associated key information as XML

The XMLDSIG Standard

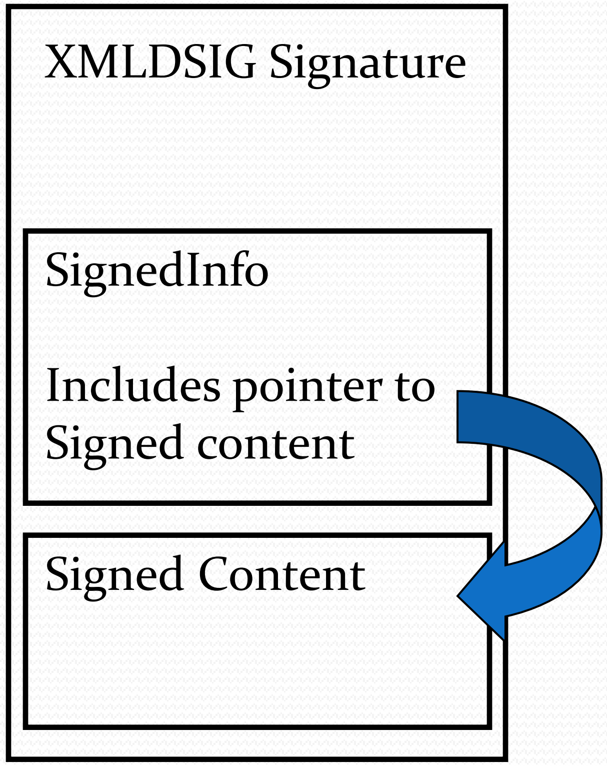
- XMLDSIG is an IETF/W3C joint standard for XML Digital Signatures
  - Signatures are represented as XML objects
  - Signed content may be XML documents, document fragments, or any binary stream
  - Baseline standard for further security work on XML Web Services (WS-Security)

Major Requirements and Key Features of XMLDSIG

- XMLDSIG supports three methods of signing an XML element
  - Wrapped, Detached and Embedded
- XMLDSIG signatures can be over an entire XML document or a fragment (sub-part) of a document
- XMLDSIG has to support the fact that an XML object might have multiple representations
  - Some modifications to the text must be allowed and not break the signature
- XMLDSIG has to support signatures over groups or collections of XML objects

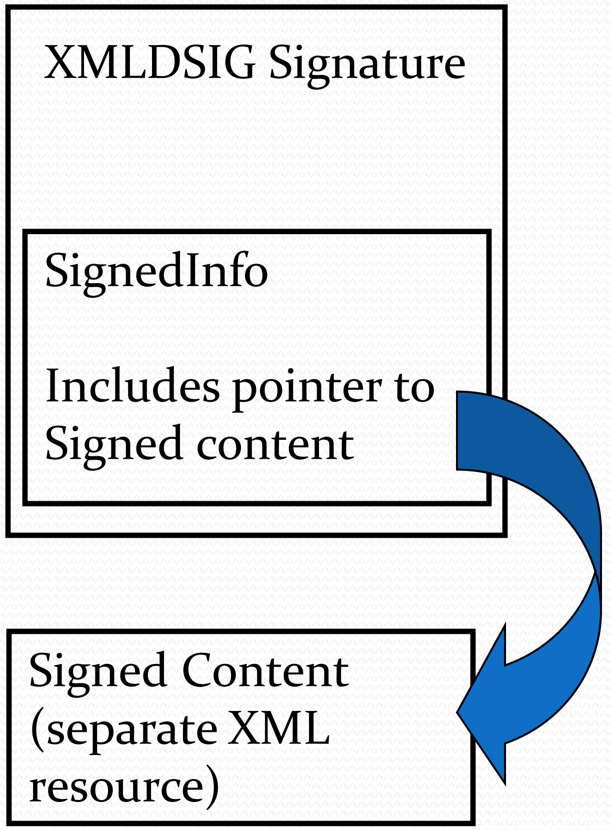
Wrapped Signatures

- Wrapped signatures include the signed content within the XMLDSIG structure
- Similar in format to a CMS (S/MIME) message
- Useful if the amount of to-be-signed data is small
  - Note: the signed content's schema is not preserved at top-level



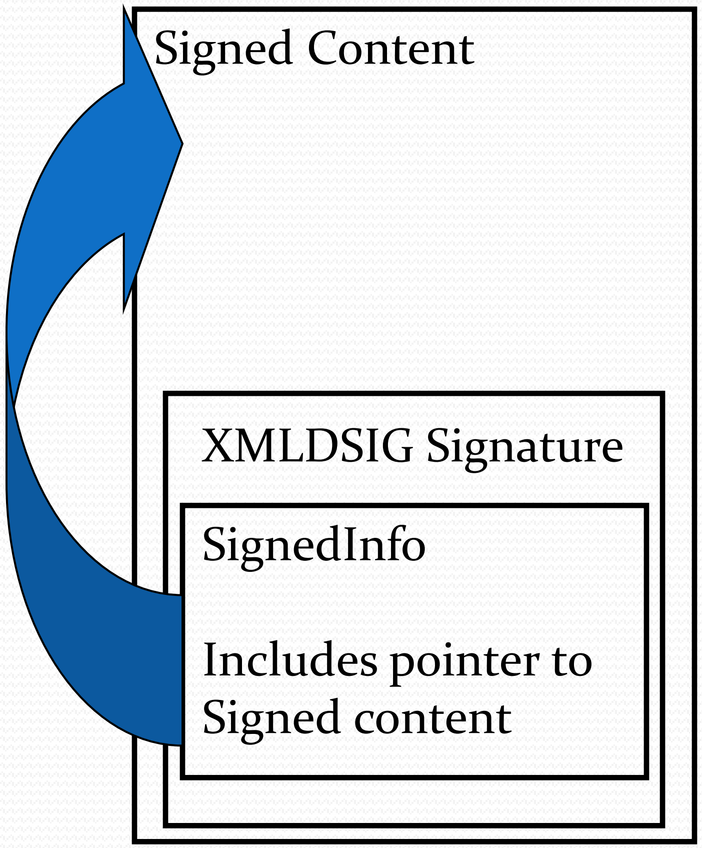
Detached Signatures

- Detached signatures separate the signature from the signed content
  - Signature travels in a separate XML document
- Useful when you want to sign non-XML data
  - E.g. audio/visual data stream


XMLDSIG Signature
SignedInfo
Includes pointer to Signed content
Signed Content (separate XML resource)

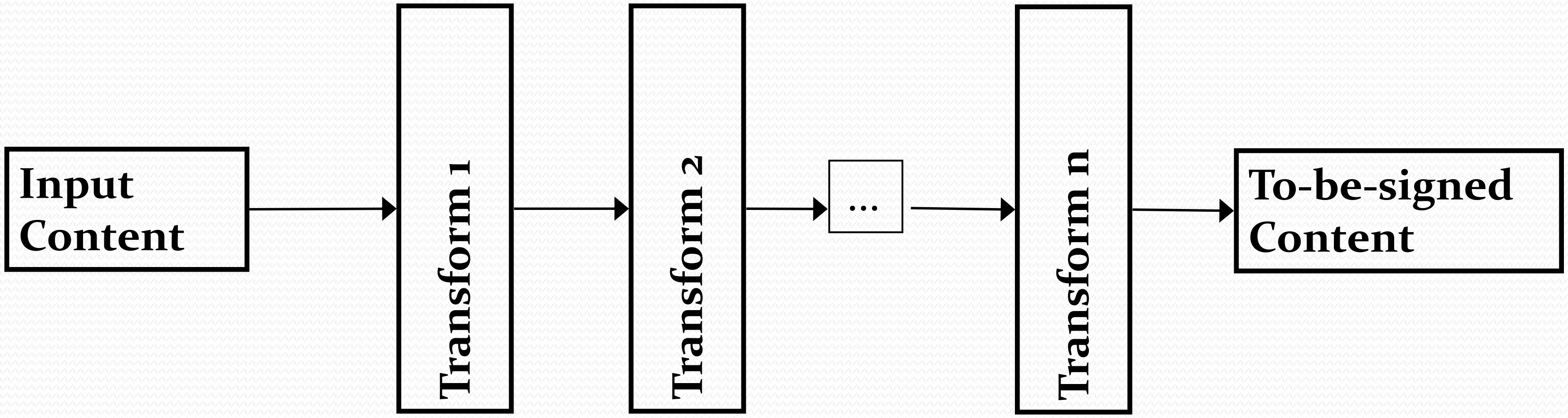
Embedded Signatures

- New mechanism unique to XMLDSIG
- Standard way to embed an XMLDSIG signature within another XML document
- Signed document carries the signature inside itself

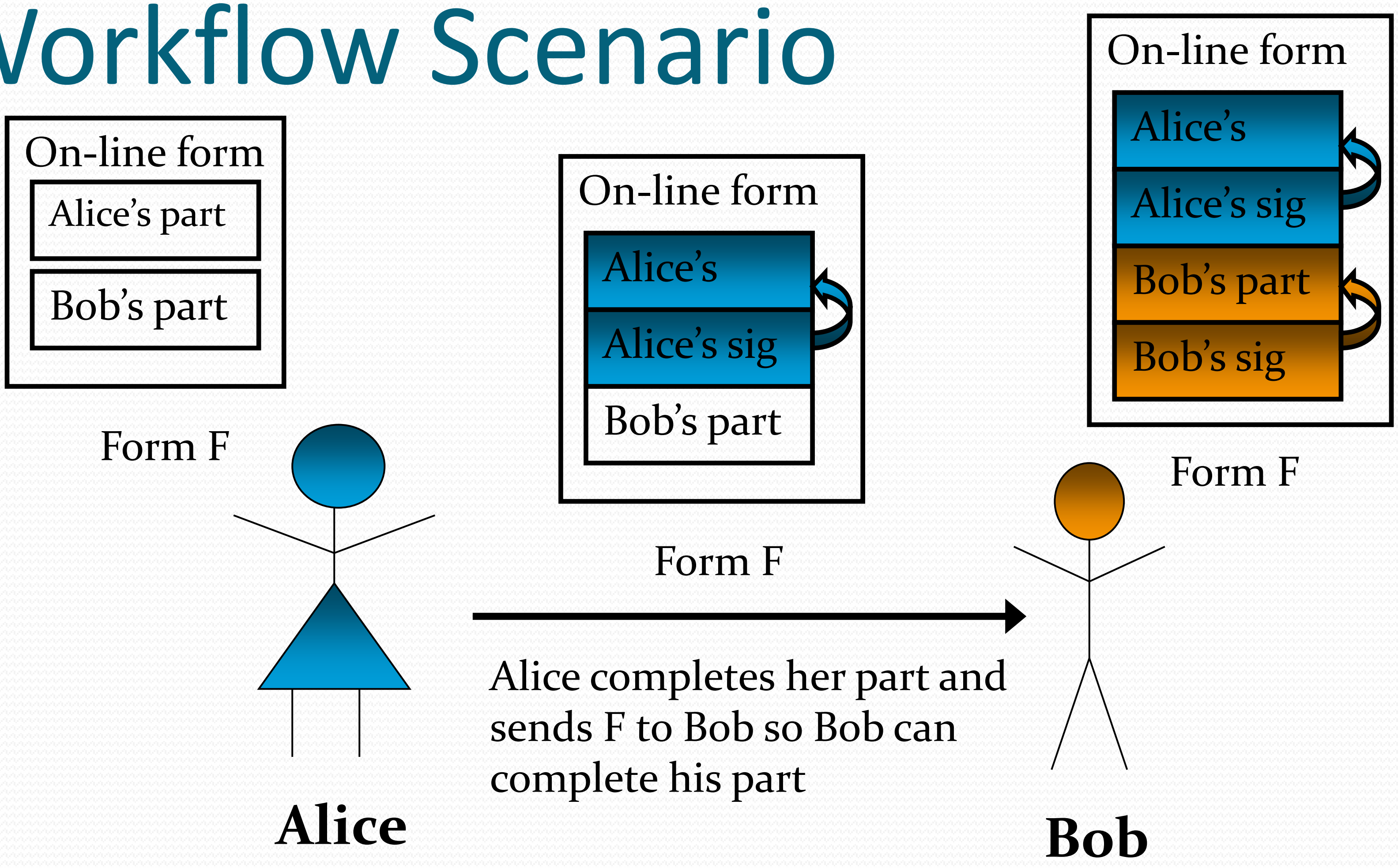


Signing Portions of Docs

- A key feature of XMLDSIG is its ability to sign selected portions of documents
  - Instead of hashing the entire document, identify & hash only those sections requiring protection
  - “Transform processing model”



Workflow Scenario



Alice completes her part and sends F to Bob so Bob can complete his part

Alice starts with a blank form

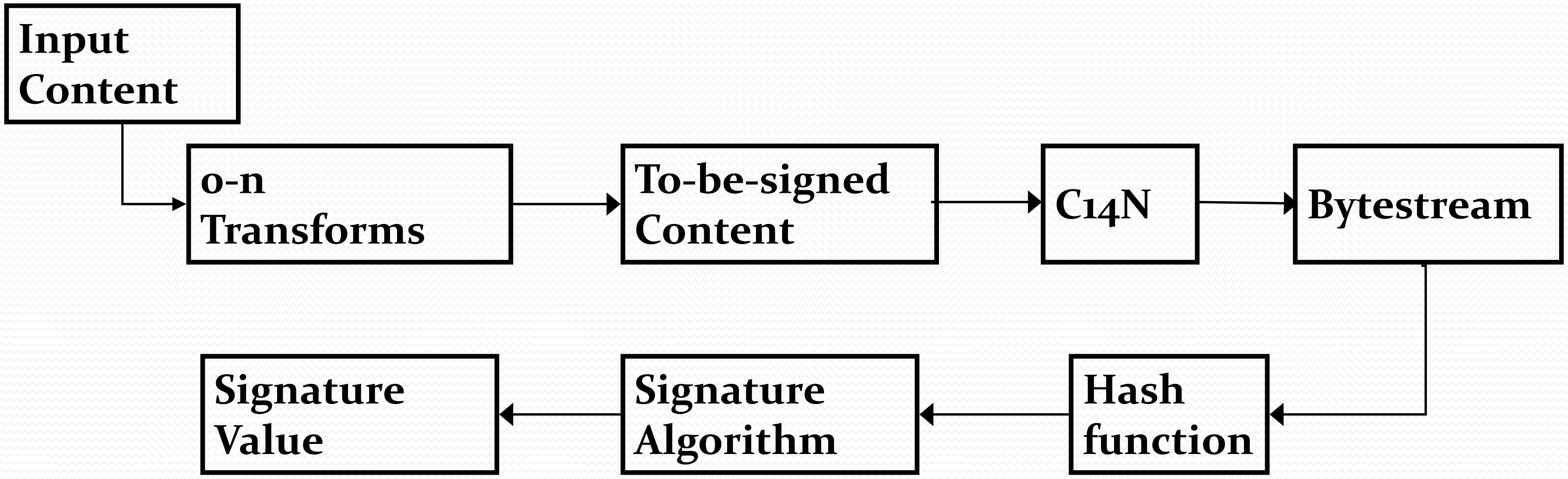
Bob completes his part and fills out the remainder of the form

Canonicalization (C14N)

- XMLDSIG introduced the notion of a “canonical form” for an XML object
  - C14N is an algorithm that converts an XML text representation into its canonical form bytestream.
  - All semantically-equivalent representations of an XML object have the same canonical form bytestream
    - That’s the ideal case – in practice for various technical reasons we don’t quite get there

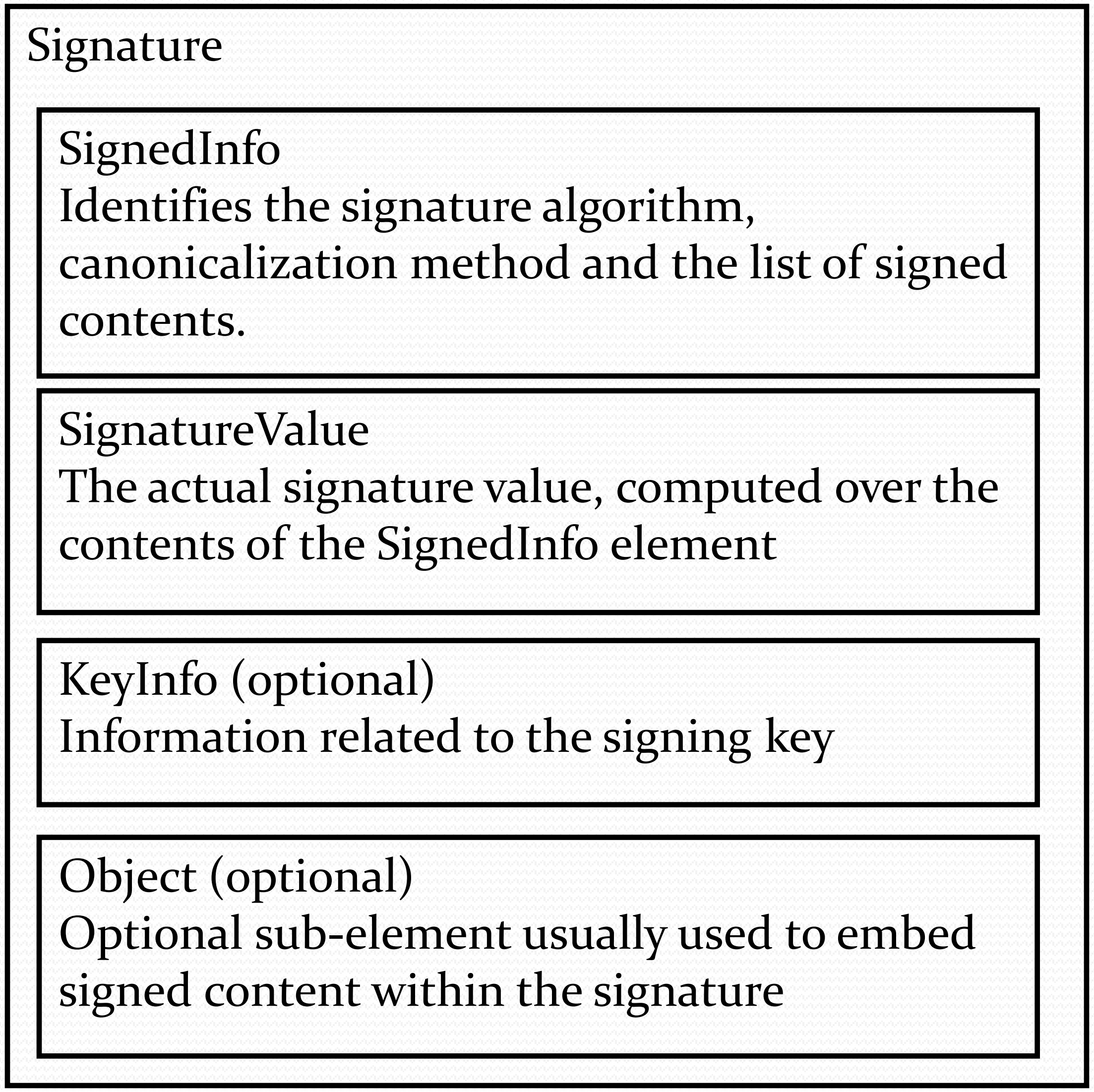
C14N and Signing

- In XMLDSIG, we compute the digital signature over the hash of the canonical form of whatever we want to sign



Structural Overview

- Top-level element is always a <Signature>
  - <SignedInfo> and <SignatureValue> are required sub-elements
  - <Keyinfo> and <Object> are optional



Signature

SignedInfo
Identifies the signature algorithm, canonicalization method and the list of signed contents.

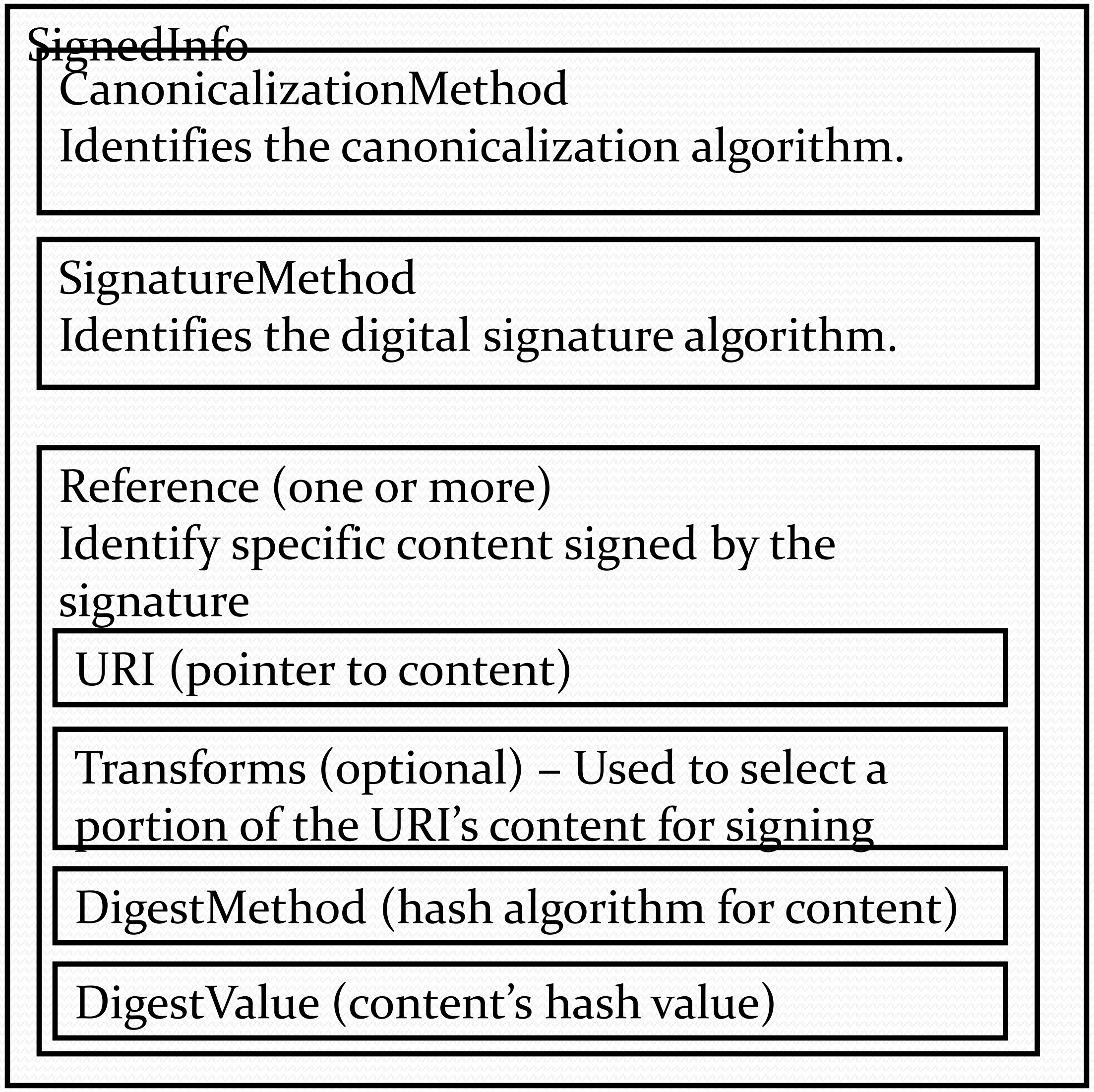
SignatureValue
The actual signature value, computed over the contents of the SignedInfo element

KeyInfo (optional)
Information related to the signing key

Object (optional)
Optional sub-element usually used to embed signed content within the signature

SignedInfo Details

- The <SignedInfo> element contains a list <Reference> elements
- Each <Reference> element points to a piece of signed content
    - <SignedInfo> is a manifest listing all the contents signed by the signature



SignedInfo

CanonicalizationMethod
Identifies the canonicalization algorithm.

SignatureMethod
Identifies the digital signature algorithm.

Reference (one or more)
Identify specific content signed by the signature

URI (pointer to content)

Transforms (optional) – Used to select a portion of the URI's content for signing

DigestMethod (hash algorithm for content)

DigestValue (content's hash value)

Sample Signature

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
  20010315" />
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="http://www.farcaster.com/index.htm">
      <DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>XoaHIm+jLKnPocR7FX0678DUOqs=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
  M5BhlrxPaOEYcCwSZ3WEDR6dfK5id/ef1JWK6OO5PEGHp9/JxrdA2xT5TYr5
  egArZGdVURpMVGUeViWoeHcGAyMNG9Cmc/I56sYd/TSV/MjLgb/mxq+6Fh/H
  WtVhjHIG+AdL4lA+ZxxEi147QVVzgCl4+dvIZaGo7oAFneDKv0I=
  </SignatureValue>
</Signature>
```

The XMLENC Standard

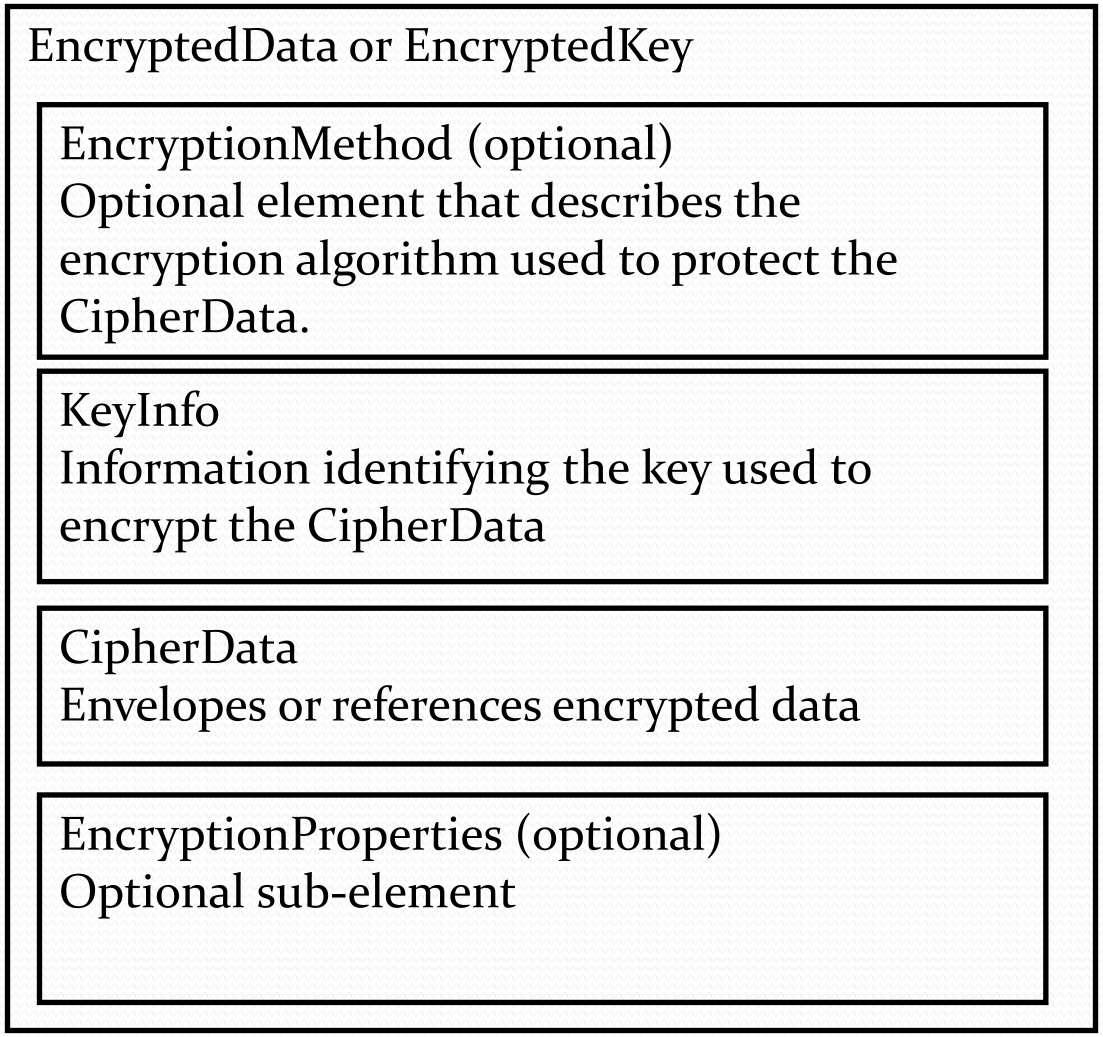
- XMLENC is a W3C Standard defining how to encrypt data and represent the result in XML
  - The data may be arbitrary data (including an XML document), an XML element, or XML element content.
  - The result of encrypting data is an XML Encryption element which contains or references the cipher data.

Key Features of XMLENC

- Wrapped or detached CipherData
  - Encrypted data may be enclosed within the metadata describing how it was encrypted, or sent separately
- EncryptedKey inside KeyInfo
  - Bulk data encryption keys wrapped in recipient public keys can be sent along with the data (a la S/MIME)
- Detached CipherData references use the same Transforms structure as XMLDSIG

Structural Overview

- Top-level element is either <EncryptedData> or <EncryptedKey>
- <EncryptedKey> has two additional properties over <EncryptedData>
  - <CipherData> always contains key material
  - An <EncryptedKey> may appear within an <EncryptedData>'s <KeyInfo> element.



EncryptedData or EncryptedKey

EncryptionMethod (optional)
Optional element that describes the encryption algorithm used to protect the CipherData.

KeyInfo
Information identifying the key used to encrypt the CipherData

CipherData
Envelopes or references encrypted data

EncryptionProperties (optional)
Optional sub-element

XMLENC Example

- Raw (unencrypted) XML: a simple payment structure with embedded credit card information

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
       <Number>4019 2445 0277 5567</Number>          3
       <Issuer>Example Bank</Issuer>                       2    1
       <Expiration>04/07</Expiration>
  </CreditCard>
</PaymentInfo>
```

XMLENC Example (1)

- Encrypting the entire <CreditCard> element including tag & attributes

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData
    Type='http://www.w3.org/2001/04/xmlenc#Element’
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <CipherData>
            <CipherValue>A23B45C56</CipherValue>
      </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XMLENC Example (2)

- Encrypting the contents of <CreditCard> element

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <EncryptedData
      xmlns='http://www.w3.org/2001/04/xmlenc#‘
      Type='http://www.w3.org/2001/04/xmlenc#Content’>
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </CreditCard>
</PaymentInfo>
```

XMLENC Example (3)

- Encrypting just the card number

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData
        xmlns='http://www.w3.org/2001/04/xmlenc#’
        Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/07</Expiration>
  </CreditCard>
</PaymentInfo>
```

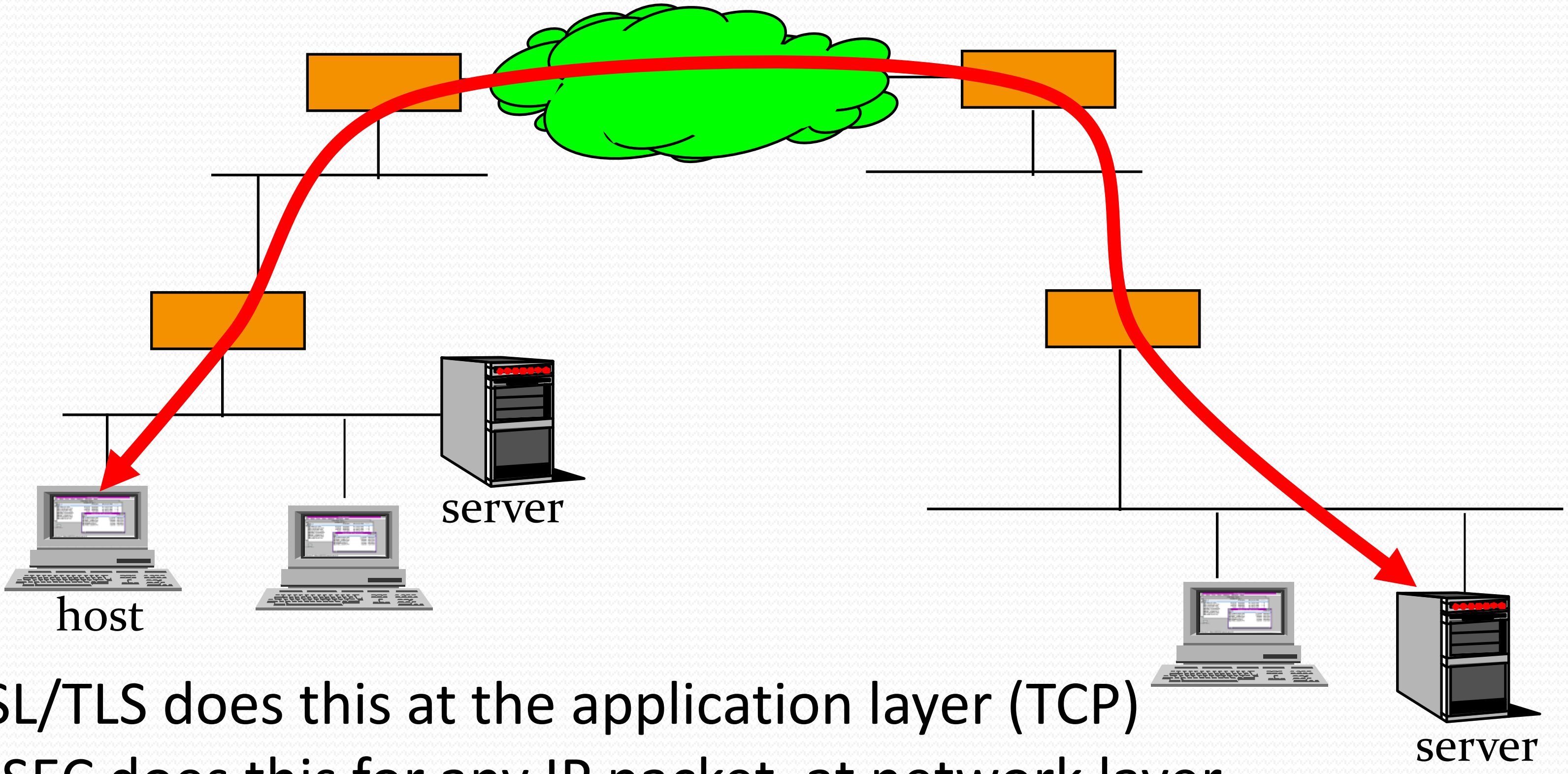
Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II

Protocol-Level Security: IPSEC

- Application-level security protocols work great for particular applications
  - But they only work for that application
- SSL/TLS requires lots of infrastructure to work; how many protocols can we do that for?
- Ideally, we'd like all the security features of SSL/TLS available for every Internet protocol/application
  - "Security at the IP layer"

Ideal Protection: End-to-End



- SSL/TLS does this at the application layer (TCP)
- IPSEC does this for any IP packet, at network layer
- Apps must be aware of/control SSL, don't have to be for IPSec

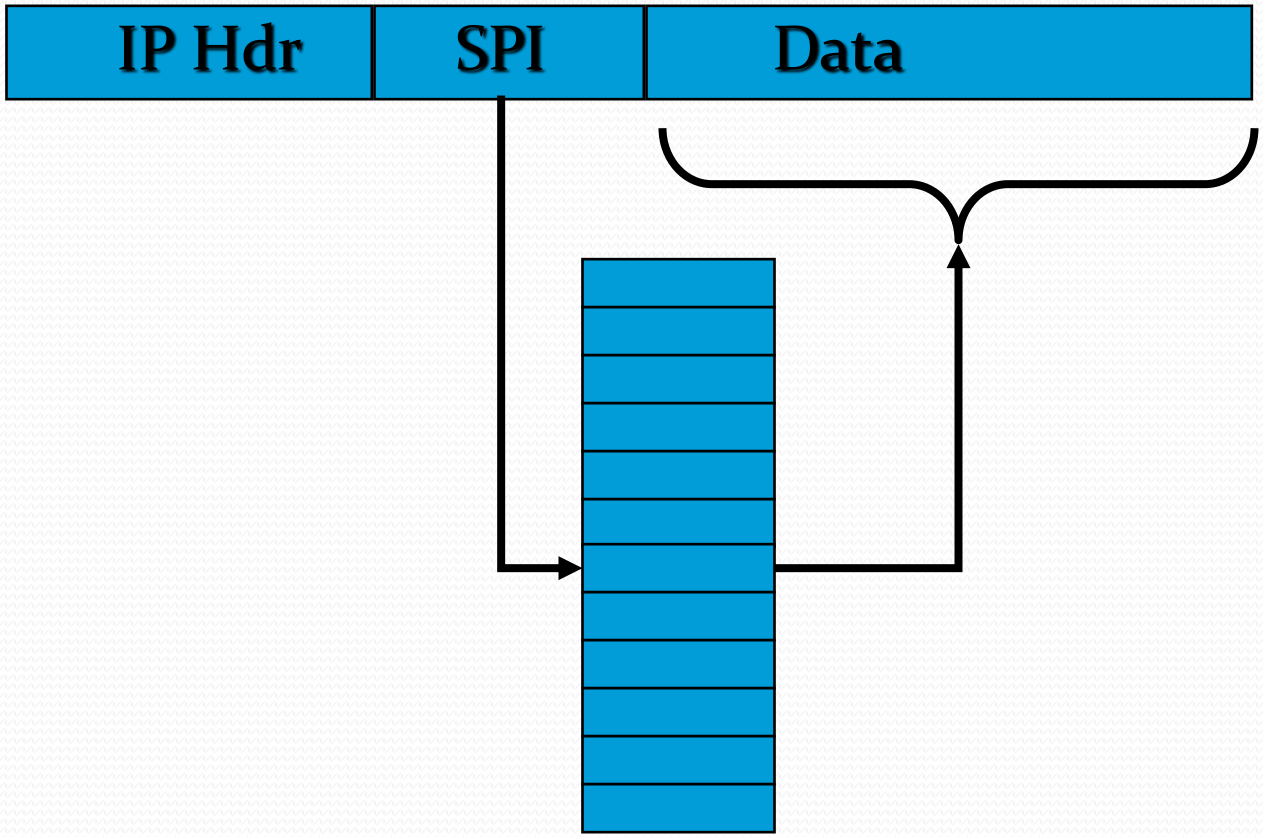
IPSEC

- IPSEC = IP (Internet Protocol) Security
  - Suite of protocols that provide encryption, integrity and authentication services for IP packets
  - Mandatory-to-implement for IPv6, optional (but available) for IPv4
- Consists of two main components:
  - IPSEC key management
  - IPSEC protection protocols
    - Encryption & auth of IP packets

IPSEC Key Management

- Establishes a Security Association (SA) for a session
  - Think “shared secret key” for each pair of communicating parties
  - SA used to provide authentication and confidentiality services for that session
  - SA is referenced via a security parameter index (SPI) in each IP datagram header

IPSEC Architecture



Security information maintained by host

IPSEC Protection Protocols

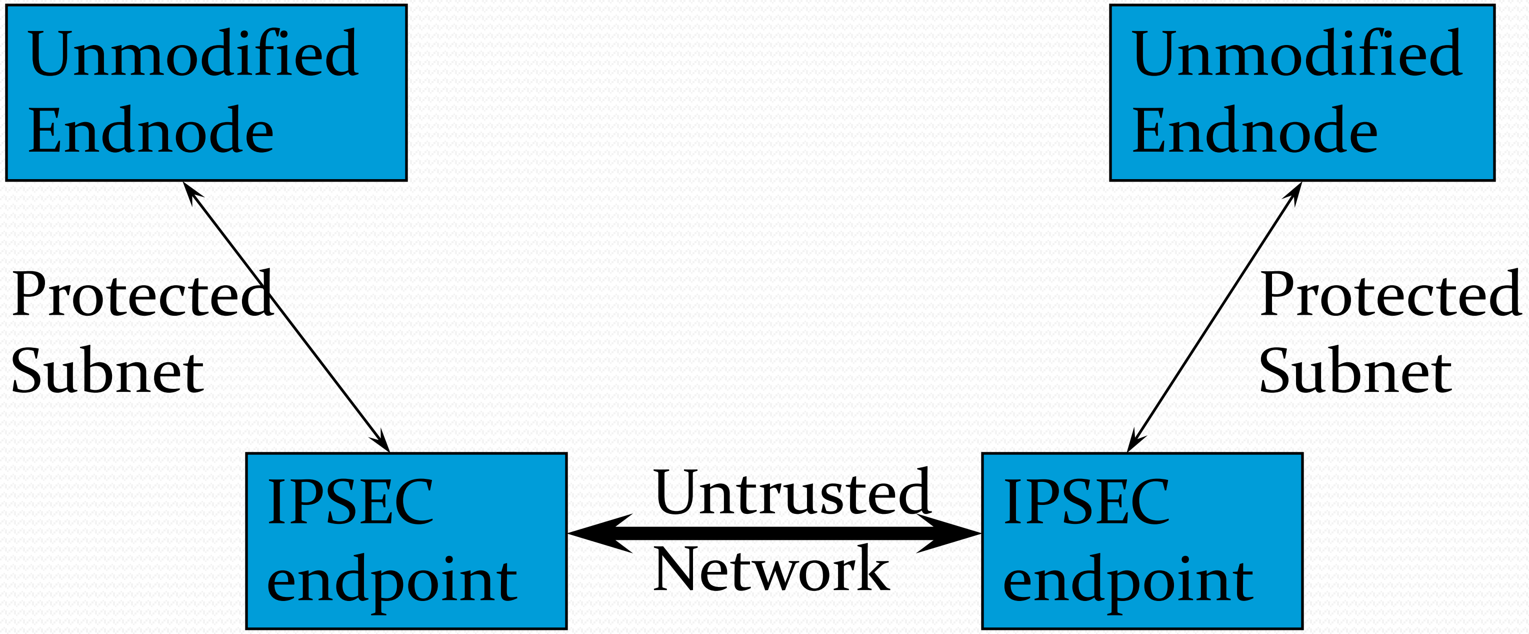
- Authentication Header (AH)
  - Authenticates payload data
  - Authenticates network header
  - Gives anti-replay protection
- Encapsulated Security Payload (ESP)
  - Encrypts payload data
  - Authenticates payload data
  - Gives anti-replay protection

IPSEC Modes of Operation

- Tunnel Mode
  - Encapsulates the entire IP packet within IPSC protection
  - Tunnels can be created between several different node types
    - Gateway to gateway
    - Host to gateway
    - Host to host
- Transport Mode
  - Encapsulates only the transport layer information within IPSEC protection
  - Can only be created between host nodes

IPsec Scenario 1
Firewall to Firewall

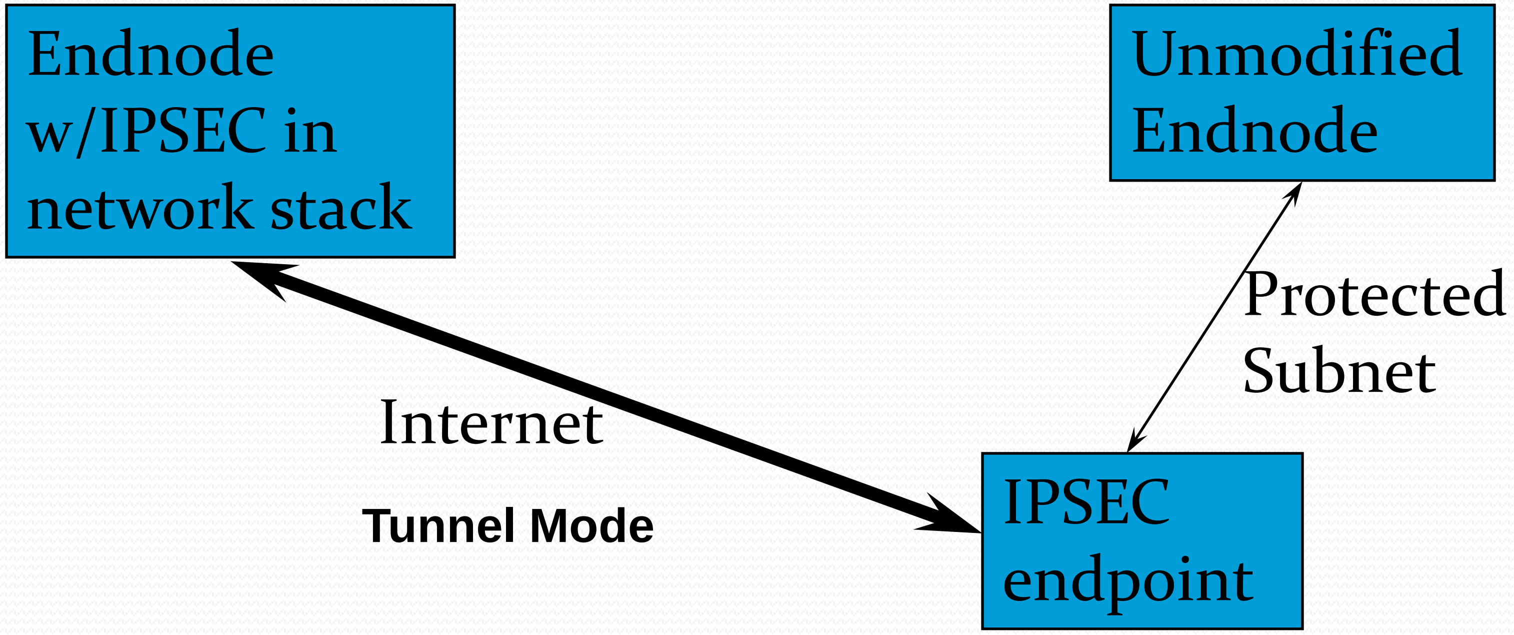
- Corporate network connected through Internet



Tunnel Mode

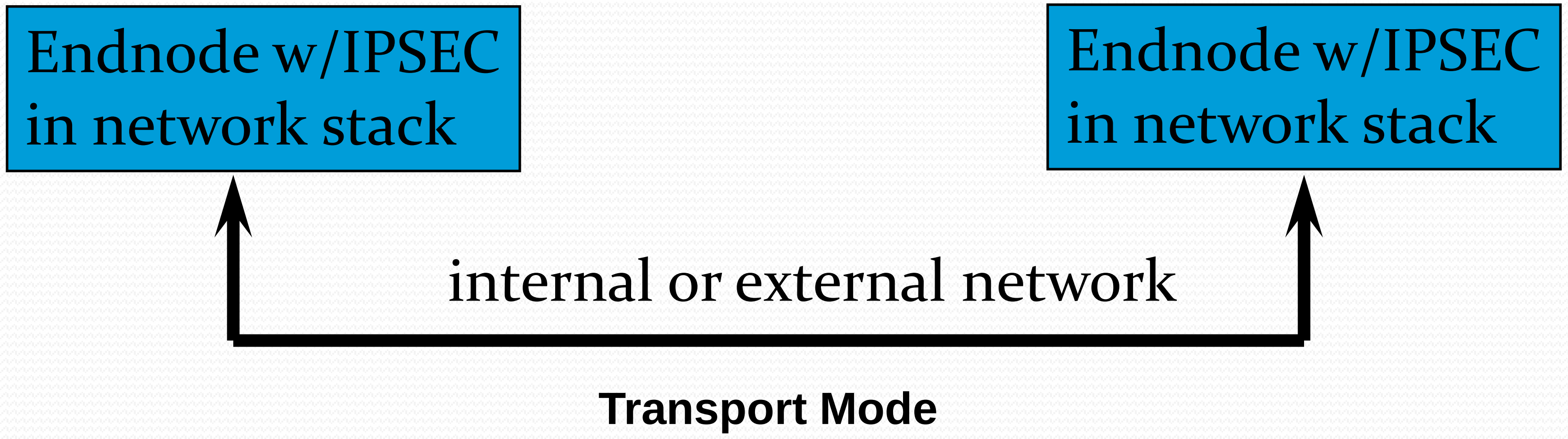
IPsec Scenario 2
Endnode to Firewall

- Mobile node connects home through Internet



IPsec Scenario 3
End to End

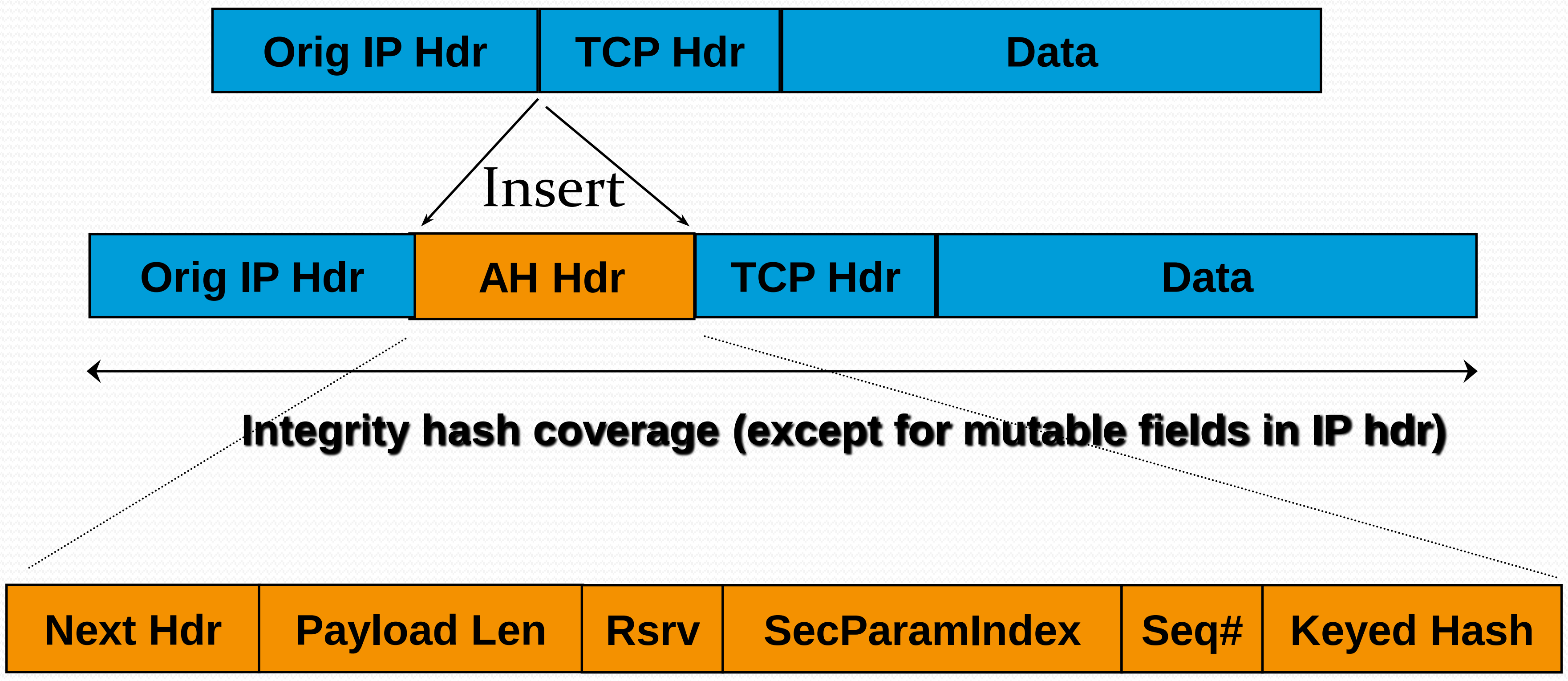
- Two nodes don’t need to trust the network


Endnode w/IPSEC in network stack
Endnode w/IPSEC in network stack
internal or external network
Transport Mode

Authentication Header (AH)

- Authentication is applied to the entire packet, with the mutable fields in the IP header zeroed out
- If both ESP and AH are applied to a packet, AH follows ESP

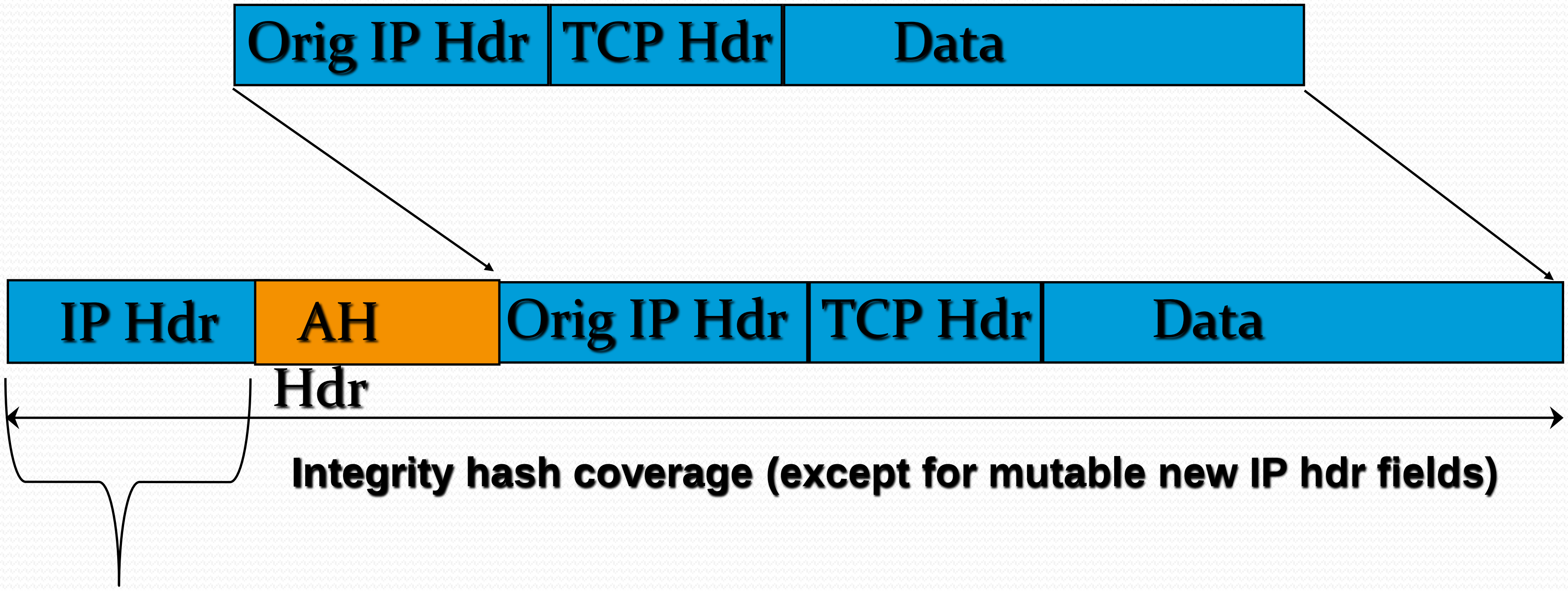
IPSEC Authentication Header (AH) in Transport Mode



AH is IP protocol 51

24 bytes total

IPSEC AH in Tunnel Mode

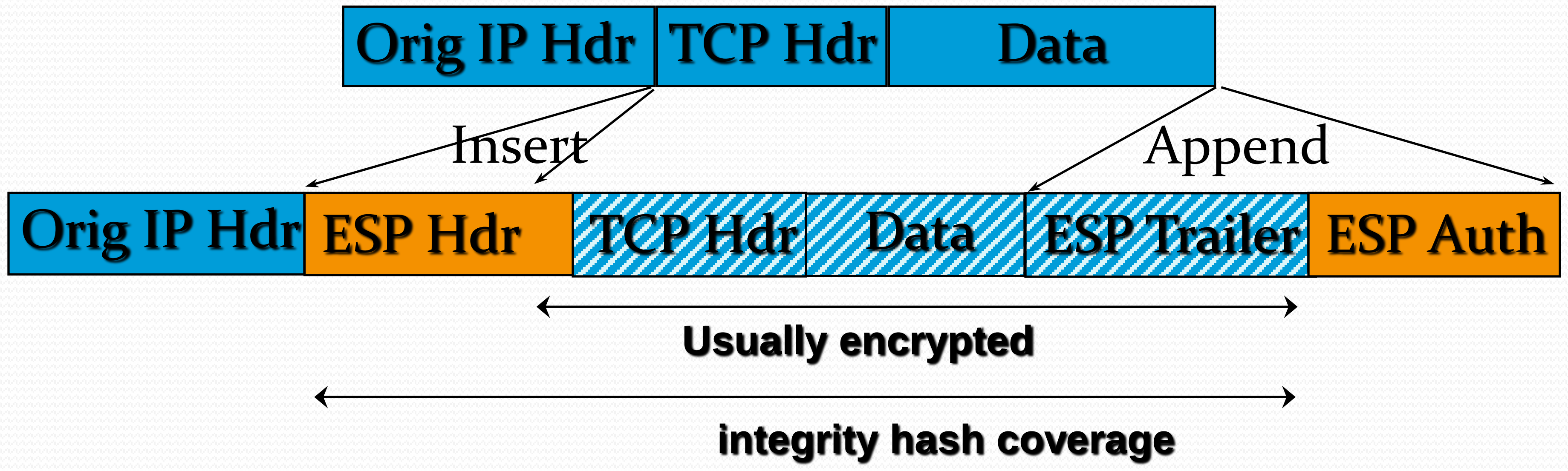

Orig IP Hdr
TCP Hdr
Data
IP Hdr
AH Hdr
Orig IP Hdr
TCP Hdr
Data
Integrity hash coverage (except for mutable new IP hdr fields)

New IP header with source & destination IP address

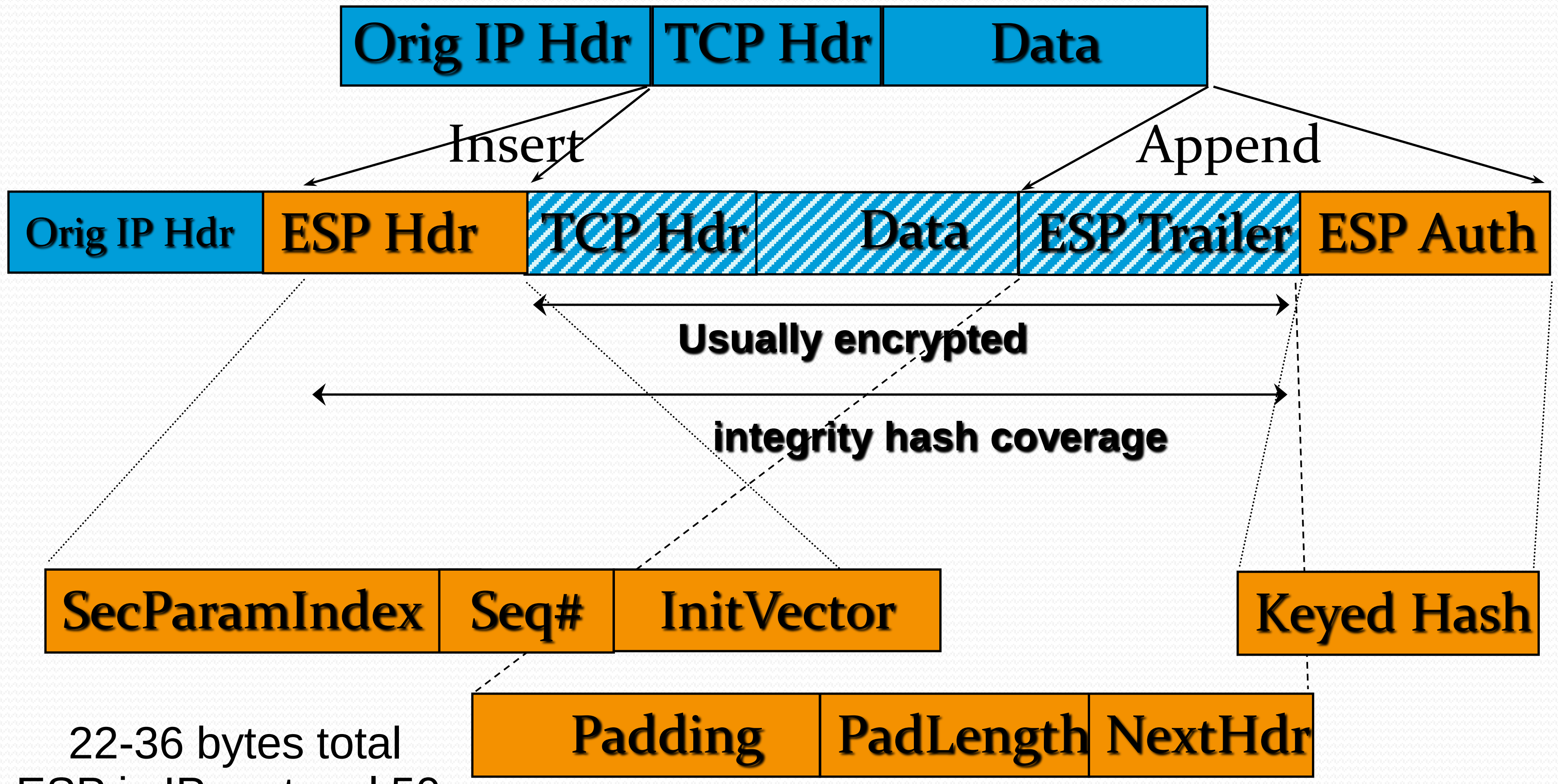
Encapsulated Security Payload (ESP)

- Must encrypt and/or authenticate in each packet
- Encryption occurs before authentication
- Authentication is applied to data in the IPSEC header as well as the data contained as payload

IPSEC ESP in Transport Mode

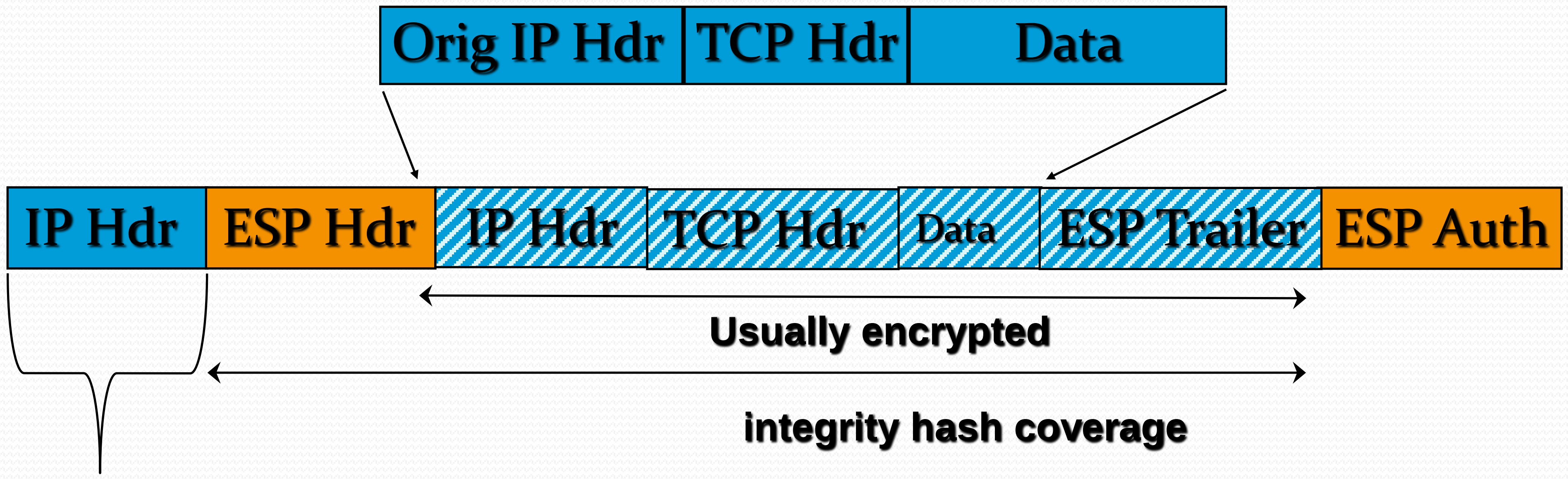


IPSEC ESP in Transport Mode



22-36 bytes total
ESP is IP protocol 50

IPSEC ESP Tunnel Mode



New IP header with source & destination IP address

IPSEC Key Management

- IPSEC Key Management is all about establishing and maintaining Security Associations (SAs) between pairs of communicating hosts

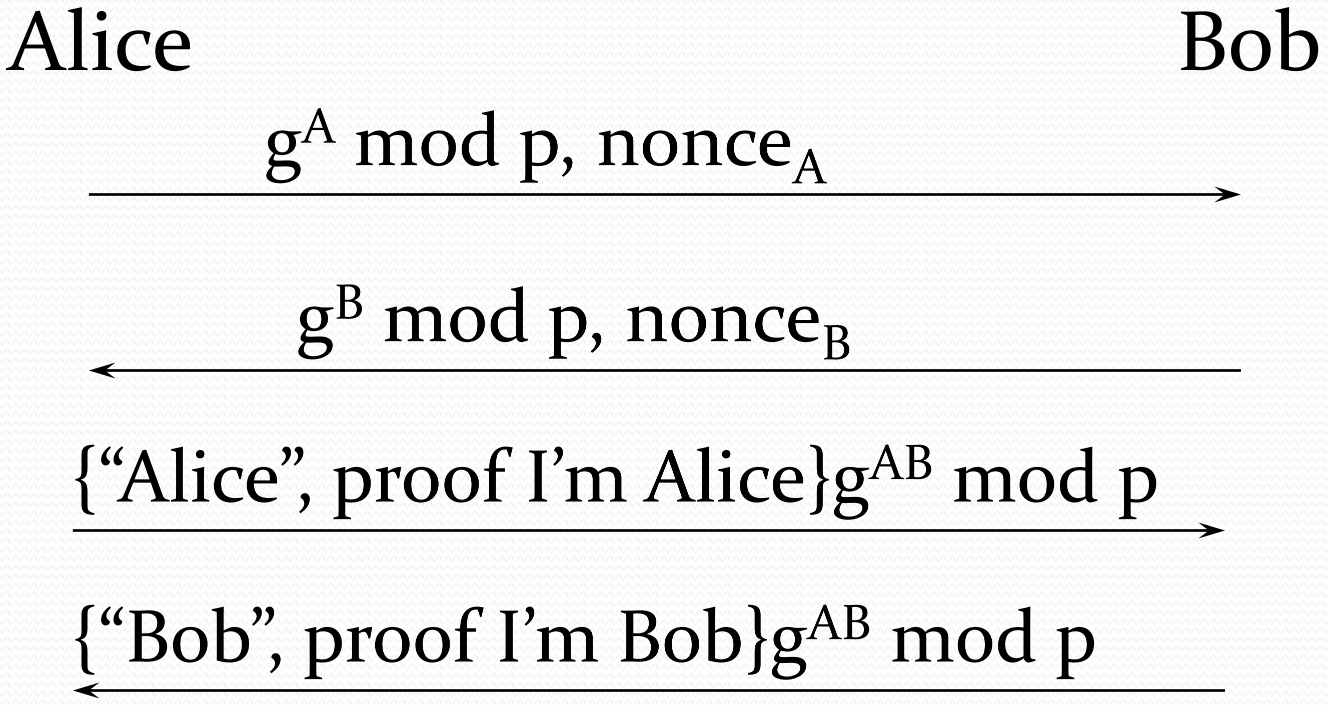
Security Associations (SA)

- New concept for IP communication
  - SA not a “connection”, but very similar
  - Establishes trust between computers
- If securing with IPSEC, need SA
  - IKE protocol negotiates security parameters according to policy
  - Manages cryptographic keys and lifetime
  - Enforces trust by mutual authentication

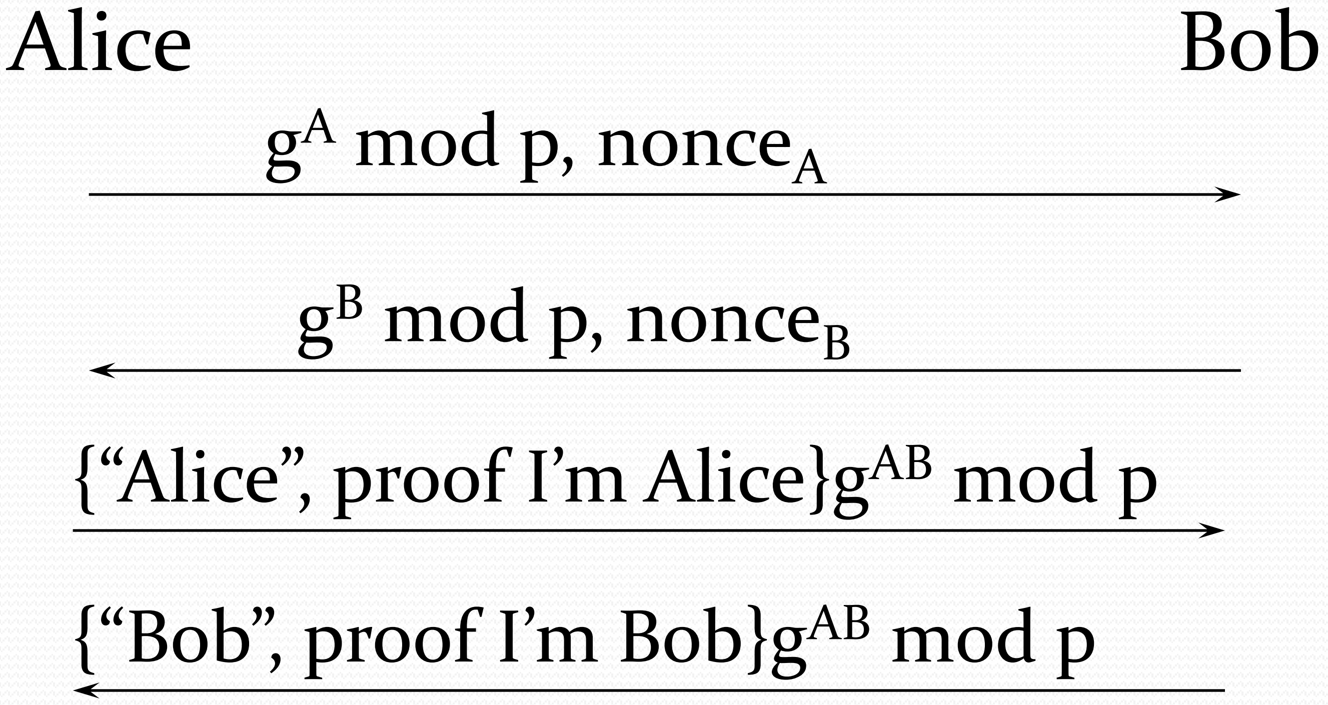
Internet Key Exchange (IKE)

- Resynchronize two ends of an IPsec SA
  - Choose cryptographic keys
  - Reset sequence numbers to zero
  - Authenticate endpoints
- Simple, right?
  - Design evolved into something very complex

General idea of IKEv2

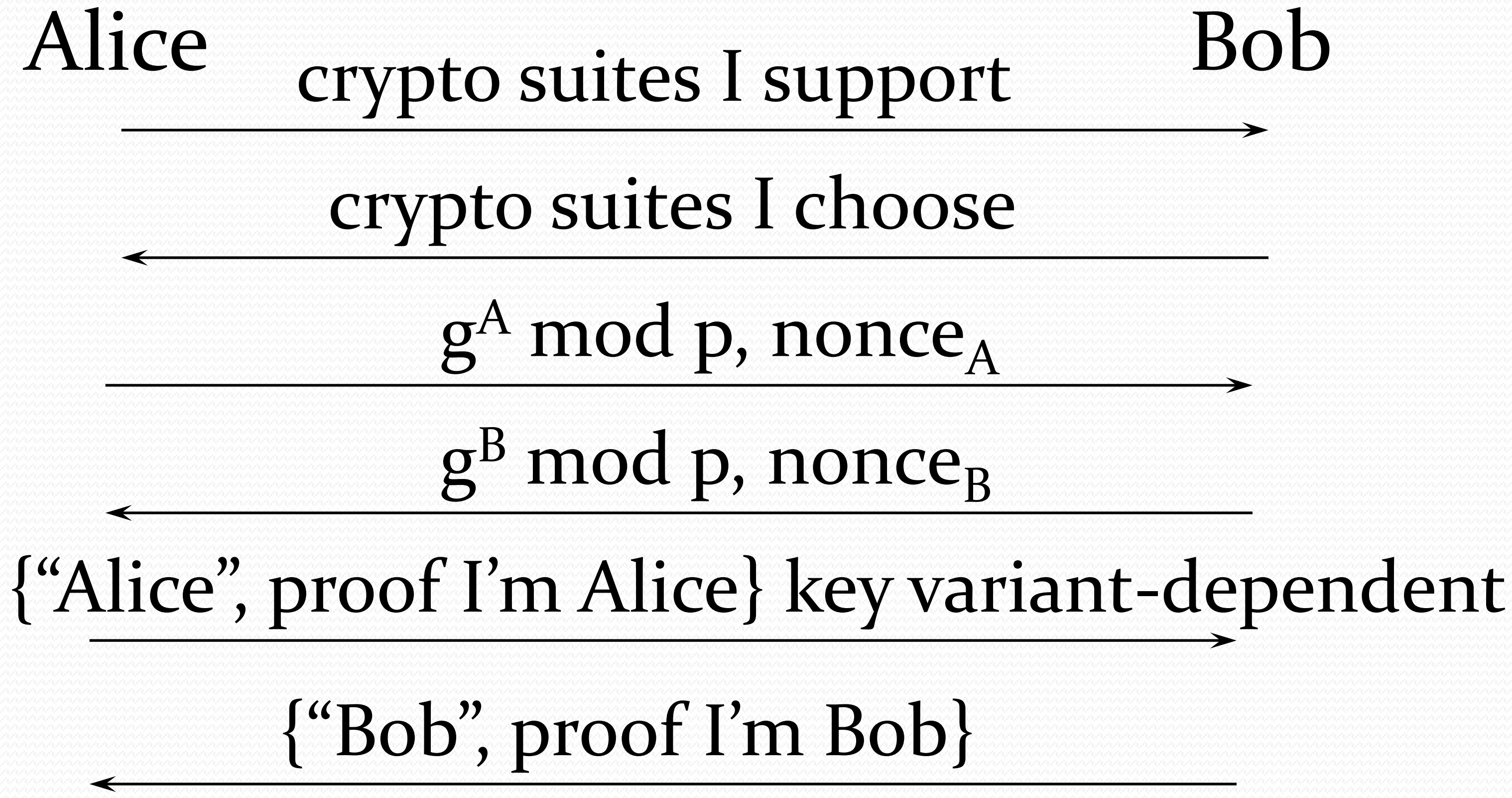

Alice
Bob
g^A mod p, nonce$_A$
g^B mod p, nonce$_B$
{"Alice", proof I'm Alice}g^{AB} mod p
{"Bob", proof I'm Bob}g^{AB} mod p

General idea of IKEv2

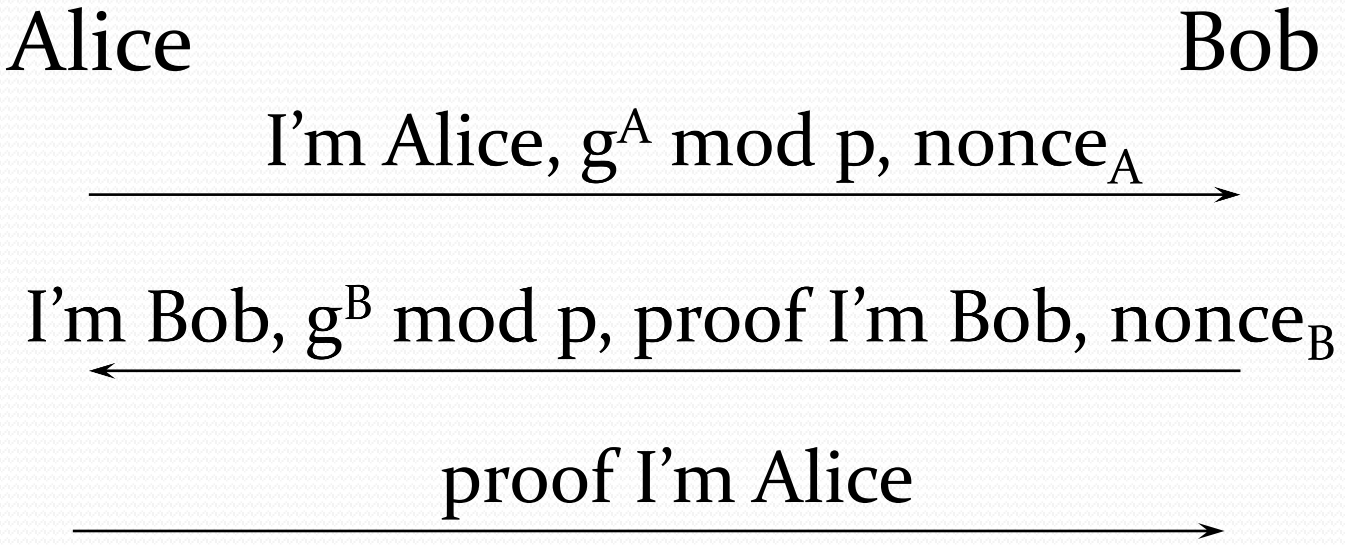

Alice
Bob
g^A mod p, nonce$_A$
g^B mod p, nonce$_B$
{“Alice”, proof I’m Alice}g^{AB} mod p
{“Bob”, proof I’m Bob}g^{AB} mod p

- It’s just Diffie-Hellman Key Exchange!

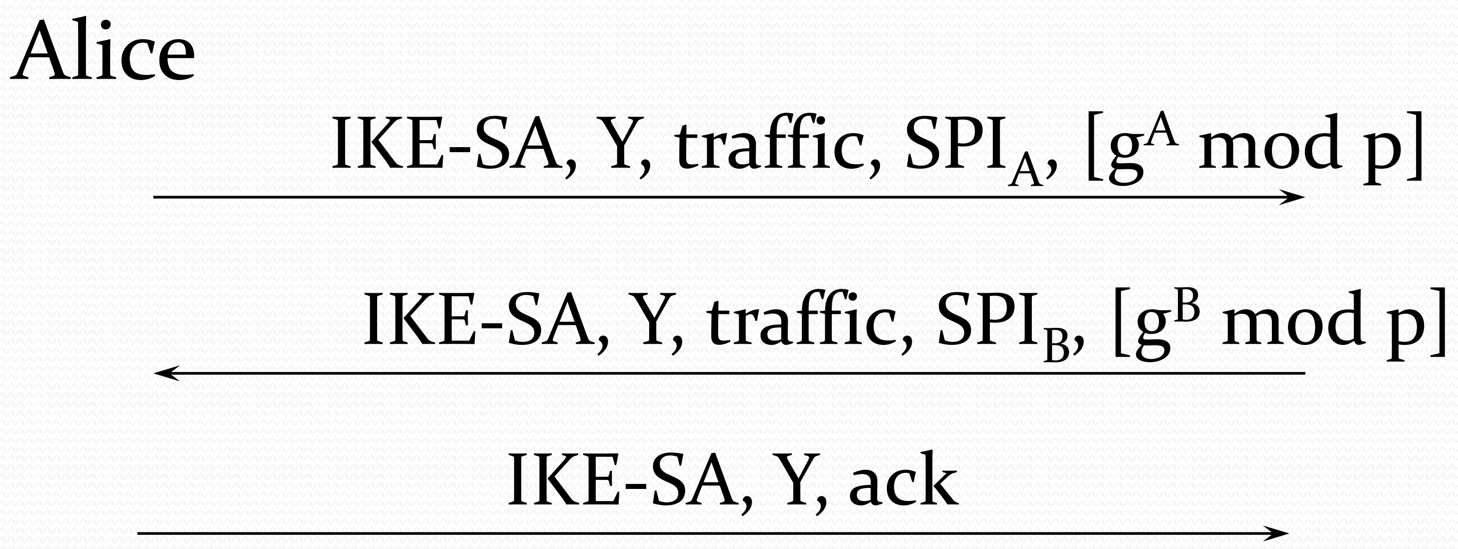
General Idea of Main Mode



General Idea of Aggressive Mode


Alice
Bob
I'm Alice, g^A mod p, nonce_A
I'm Bob, g^B mod p, proof I'm Bob, nonce_B
proof I'm Alice

General idea of Quick Mode


Alice
Bob
IKE-SA, Y, traffic, SPI_A, [g^A mod p]
IKE-SA, Y, traffic, SPI_B, [g^B mod p]
IKE-SA, Y, ack

Agenda

- Guest lecture: Christian Rechberger, KU Leuven
  - *Towards SHA-3*
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- IPsec (depending on time)
- Design Charrette Part II