summer school

Web Services and Identity

9 September 2010

XML Security

Thomas Roessler, W3C

@roessler / tlr@w3.org

http://tinyurl.com/xmlss10sec

Crypto primitives
XML Encryption
XML Signature
What could possibly go wrong?


CARRER
DEL
VEGUER
13
ANTIQUARI

He knows she said it:

authenticity

She knows only he heard it:

confidentiality

He doesn't know
from whom she got it.

She doesn't know
whom he'll tell.

channel properties


CARRER
DEL
VEGUER
13
ANTIQUARI


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 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
C
S
S
A

classical cryptography:

share a secret, exchange messages

channel transformation in time

confidential now

→ confidential then

authenticated +
confidential now
➔ authenticated then

e.g.
Caesar cipher

e.g.
Advanced Encryption Standard

primitive:
symmetric ciphers

cleartext + key

➔ ciphertext

ciphertext + key

→ cleartext

no key
no cleartext

1976
Diffie & Hellman
New directions in cryptography

different channel transforms

confidential now

confidential then

authenticated now

confidential then

public key cryptography

public key + cleartext

→ ciphertext

private key + ciphertext

→ cleartext

public key → private key:
hard problem

asymmetric cipher

e.g.,
Diffie-Hellman
RSA
...

hybrid ciphers

fast, symmetric cipher with random key

random key encrypted with public key

another primitive: signatures

private key + plaintext

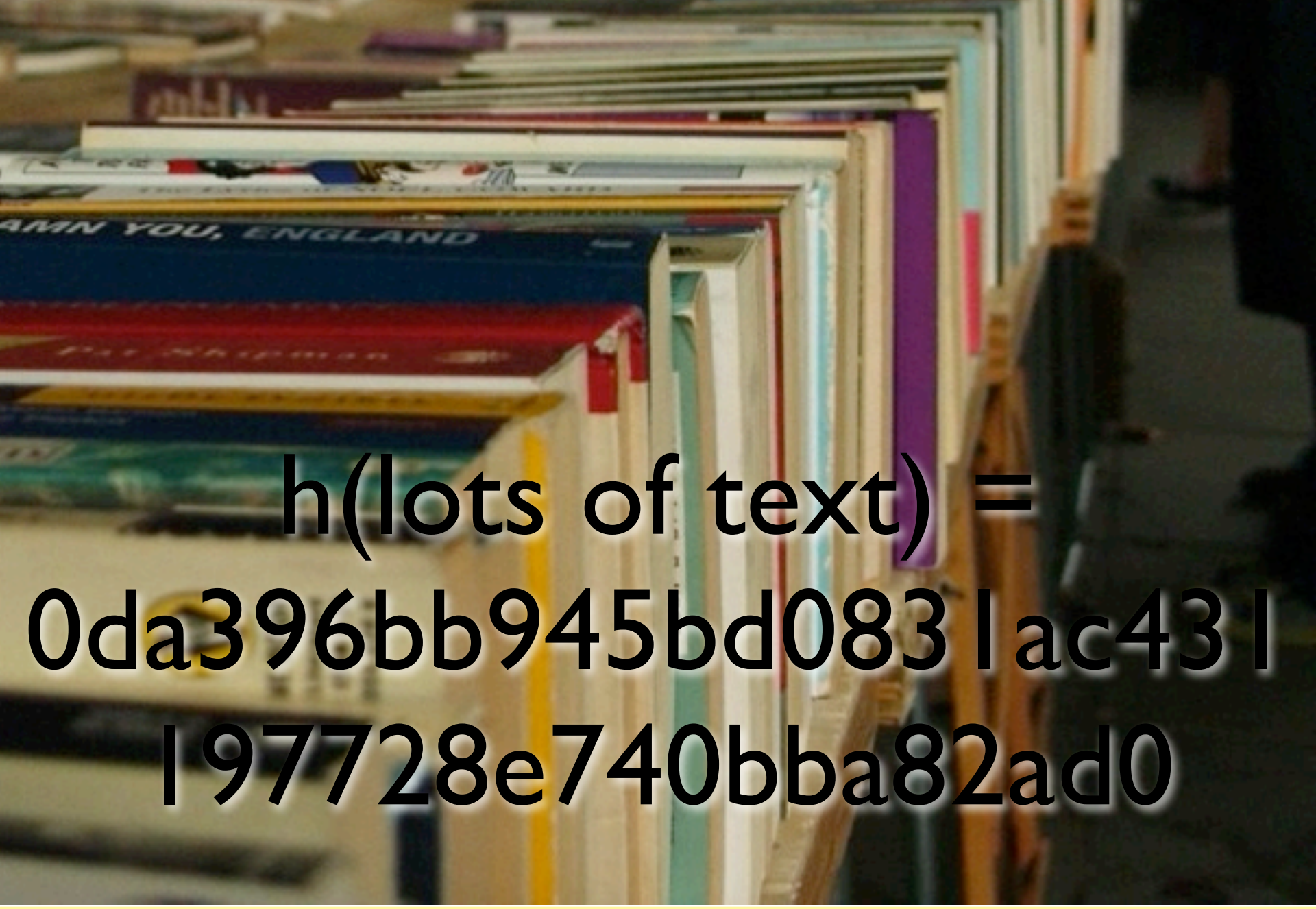
→ signature

signature + plain text + public key

→ ok (or not)

binding a message to a key

primitive:
hash functions


h(lots of text) =
0da396bb945bd0831ac431
197728e740bba82ad0

variable length input
fixed length output
fast

interesting properties:

pre-image resistance

2nd pre-image resistance

collision resistance

pre-image resistance:
hard to find
x with $h(x) = y$
(y is given)

2nd pre-image resistance: hard to find x' with $h(x') = h(x)$ (x is given)

collision resistance:
hard to find *x*, *x'* with
$h(x) = h(x')$

e.g.,

SHA-256

RIPEMD-160

…

sign(*key*, h(*message*))

instead of

sign(*key*,*message*)

$$\mathrm{HMAC}(K,m) = H((K \oplus \mathrm{opad}) \parallel H((K \oplus \mathrm{ipad}) \parallel m))$$

what have we learned so far?

crypto algorithms give us channels

they bind stuff to keys

public key crypto does interesting channel transforms

don’t brew your own algorithms

the keys are key

Signature Encryption

use XML to encapsulate crypto

encrypt and sign XML documents

http://www.w3.org/TR/xmlenc-core1/

encrypts:
elements
character data
stuff
keys

```
<PaymentInfo
    xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData
      Type='…/2001/04/xmlenc#Element'
      xmlns='…/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

EncryptedData: “Replace me after you’ve decrypted me.”

```
<EncryptedData
  xmlns='…/2001/04/xmlenc#'
  Type='…/2001/04/xmlenc#Element'>
 <EncryptionMethod
   Algorithm='…/xmlenc#tripledes-cbc'/>
 <ds:KeyInfo
    xmlns:ds='…/2000/09/xmldsig#'>
   <ds:KeyName>John Smith</ds:KeyName>
 </ds:KeyInfo>
 <CipherData>
   <CipherValue>DEADBEEF</CipherValue>
 </CipherData>
</EncryptedData>
```

```
<EncryptionMethod
  Algorithm='…/xmlenc#
                tripledes-cbc'/>
```

```
<ds:KeyInfo
  xmlns:ds='…/2000/09/xmldsig#'>
  <ds:KeyName>…</ds:KeyName>
</ds:KeyInfo>
```

identifying the key,
not transmitting it!

variant:
CipherReference
“pull the encrypted data from elsewhere”

also:

EncryptedKey

DerivedKey

caveat:
namespaces

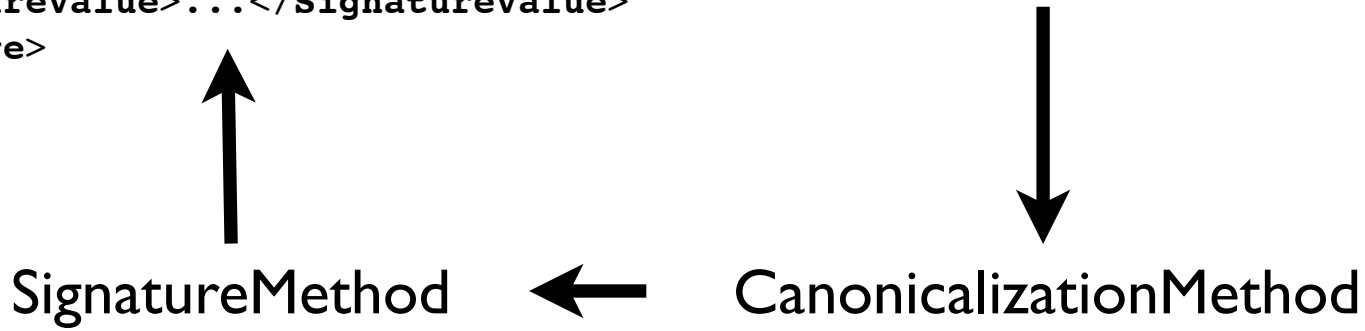
xmlns=”…”
is your friend

caveat:
schema validation

http://www.w3.org/TR/xmldsig-core1/

```
<Signature Id="MyFirstSignature"
   xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod
     Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
     Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  <Reference
     URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
    <Transforms>
      <Transform
         Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </Transforms>
    <DigestMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
  </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```

```
<Signature Id="MyFirstSignature"
   xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod
     Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
     Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  <Reference …>
    …
  </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```



```
<Signature Id="MyFirstSignature"
   xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod
     Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
     Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  <Reference
     URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
    <Transforms>
      <Transform
         Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </Transforms>
    <DigestMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
  </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```

```
<Reference
  URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
 <Transforms>
   <Transform
      Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
 </Transforms>
 <DigestMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
 <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
</Reference>
```

table of contents

W3C

XHTML™ 1.0: The Extensible HyperText Markup Language

A Reformulation of HTML 4 in XML 1.0

W3C Recommendation 26 January 2000

This version:

Latest version:

Previous version:

Authors:

Abstract

transform chain

(canonicalization)

DigestMethod

```
<DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
```

<Reference
URI="http://…">

external URI references (http://…)

→ octet stream

<Reference
URI="#…">

same-document reference (#…)

➔ node-set

you can't hash a nodeset

→ canonicalization

generates octet-stream

Canonical XML 1.0
Exclusive Canonicalization

Canonical XML 1.0
all namespaces that are in force
assorted attributes

1.0 inherited

xml:id

fixed in Canonical XML 1.1

subset issues:
Exclusive C14N 1.0
namespaces that are
actually used

```
<Reference
  URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
 <Transforms>
   <Transform
      Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
 </Transforms>
 <DigestMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
 <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
</Reference>
```

table of contents

W3C

XHTML™ 1.0: The Extensible HyperText Markup Language

A Reformulation of HTML 4 in XML 1.0

W3C Recommendation 26 January 2000

This version:

Latest version:

Previous version:

Authors:

Abstract

transform chain

(canonicalization)

DigestMethod

```
<DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
```

transforms can do

anything

XSLT
XPath filter v.*N*
xproc
your own

identified

by URI

A signature binds data to a key.

XML Signature binds SignedInfo to a key.

A Reference element binds the result of some transform chain into a signature.

table of contents

W3C

XHTML™ 1.0: The Extensible HyperText Markup Language

A Reformulation of HTML 4 in XML 1.0

W3C Recommendation 26 January 2000

This version:

Latest version:

Previous version:

Authors:

Abstract

transform chain

(canonicalization)

DigestMethod

```
<DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
```

the result

of a transform chain

use case:
enveloped signature

"remove <Signature> from parent before computing hash"

```
<Reference
  URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
 <Transforms>
   <Transform
      Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
 </Transforms>
 <DigestMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
 <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
</Reference>
```

to understand the signature, you MUST understand the transforms

XSLT is Turing-complete

```
<Transform
    Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:for-each select="//. | //@*">
        <xsl:for-each select="//. | //@*">
          <xsl:for-each select="//. | //@*">
            <foo/>
          <xsl:for-each>
      <xsl:for-each>
    <xsl:for-each>
  </xsl:stylesheet>
<Transform>
```

XSLT transform with inline stylesheet: flexible, but dangerous.

Consider minting your own URIs!

XML Signature binds SignedInfo to a key.

What does a signature *mean*?

“approved”
“noted”
“under penalty of perjury”
“vetoed”

semantics need to be part of the signature

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"
  Id="DistributorSignature">
 <SignedInfo>
  <CanonicalizationMethod
   Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
   Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
 …
  <Reference URI="#prop">
   <Transforms>
    <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
   </Transforms>
   <DigestMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
   <DigestValue>…</DigestValue>
  </Reference>
 </SignedInfo>
 <SignatureValue>…</SignatureValue>
 …
 <Object Id="prop">
  <SignatureProperties
   xmlns:dsp="http://www.w3.org/2009/xmldsig-properties">
   <SignatureProperty Id="profile" Target="#DistributorASignature">
    <dsp:Profile URI="http://www.w3.org/ns/widgets-digsig#profile"/>
   </SignatureProperty>
   <SignatureProperty Id="role" Target="#DistributorASignature">
    <dsp:Role
      URI="http://www.w3.org/ns/widgets-digsig#role-distributor"/>
   </SignatureProperty>
   <SignatureProperty Id="identifier" Target="#DistributorASignature">
    <dsp:Identifier>07425f59c544b9cebff04ab367e8854a</dsp:Identifier>
   </SignatureProperty>
  </SignatureProperties>
 </Object>
</Signature>
```

```
<Signature xmlns="…/2000/09/xmldsig#"
        Id="DistributorSignature">
```

```
<Reference URI="#prop">
 <Transforms>
  <Transform Algorithm="…/2006/12/xml-c14n11"/>
 </Transforms>
 <DigestMethod
  Algorithm="…/2001/04/xmlenc#sha256"/>
 <DigestValue>…</DigestValue>
</Reference>
```

```
<Object Id="prop">
 <SignatureProperties
  xmlns:dsp="http://www.w3.org/2009/xmldsig-properties">
  <SignatureProperty Id="profile"
      Target="#DistributorASignature">
   <dsp:Profile
      URI="http://www.w3.org/ns/widgets-digsig#profile"/>
  </SignatureProperty>
   …
  <SignatureProperty Id="identifier"
      Target="#DistributorASignature">
<dsp:Identifier>07425f59c544b9cebff04ab367e8854a</dsp:Identifier>
  </SignatureProperty>
 </SignatureProperties>
</Object>
```

e.g.,
W3C Widget Signatures
XADES

“it’s signed” – what’s wrong with this sentence?

“it’s signed *by XYZ*”

remember:
the key is key

<Signature>
can include
<KeyInfo>

key material
pointer at key
identifier for key
...

Just because the key is in the signature, it isn't the right one!

You need to know **why** you think the signature key is associated with the entity you think you talk to.

certificates

previous key exchange

...

What could possibly go wrong?

“it’s signed”

“it”?

by what key?

who controls that key?

powerful
complex
complicated

use as few features
as you can

don’t need arbitrary transforms?
don’t support them!

signatures can be attack vectors, too

```
<Transforms xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
 <Transform
   Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         xmlns:java="java">
   <xsl:template match="/" xmlns:os="java:lang.Runtime" >
    <xsl:variable name="runtime"
                  select="java:lang.Runtime.getRuntime()"/>
    <xsl:value-of select="os:exec($runtime, 'shutdown -i')" />
   </xsl:template>
  </xsl:stylesheet>
 </Transform>
</Transforms>
```

e.g.,
XSLT extensions to execute arbitrary commands

```
<Transform
    Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:for-each select="//. | //@*">
        <xsl:for-each select="//. | //@*">
          <xsl:for-each select="//. | //@*">
            <foo/>
          <xsl:for-each>
      <xsl:for-each>
    <xsl:for-each>
  </xsl:stylesheet>
<Transform>
```

don’t execute transforms that you don’t know

if you don't need the flexibility, don't use it

mint URIs instead!

verification order:

1. reference validation
2. SignedInfo
3. check if it's the right key

DON'T DO THAT!

verification order:

1. Acceptable signature key?
2. Does SignatureValue work out with SignedInfo?
3. reference validation

```
<Signature Id="MyFirstSignature"
   xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod
     Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
     Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
  <Reference
     URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
    <Transforms>
      <Transform
         Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </Transforms>
    <DigestMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
  </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```

know what information you trust

The HMAC algorithm (RFC2104 [HMAC]) takes the truncation length in bits as a parameter; if the parameter is not specified then all the bits of the hash are output. An example of an HMAC `SignatureMethod` element:

```
<SignatureMethod
   Algorithm=
     "http://www.w3.org/2000/09/xmldsig#hmac-sha1">
   <HMACOutputLength>128</HMACOutputLength>
</SignatureMethod>
```

The output of the HMAC algorithm is ultimately the output (possibly truncated) of the chosen digest algorithm. This value shall be base64 encoded in the same straightforward fashion as the output of the digest algorithms.

```
<SignatureMethod
   Algorithm="…/2000/09/xmldsig#hmac-sha1">
    <HMACOutputLength>64</HMACOutputLength>
</SignatureMethod>
```

```
<SignatureMethod
   Algorithm="…/2000/09/xmldsig#hmac-sha1">
    <HMACOutputLength>1</HMACOutputLength>
</SignatureMethod>
```

I bit

know what
you reference

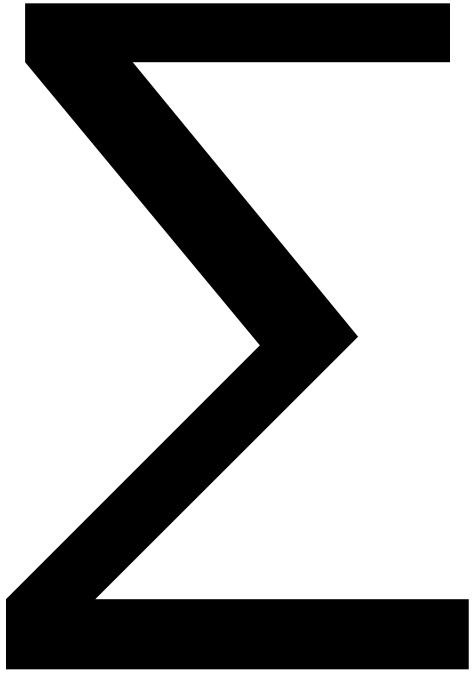
URI=”#foo”

```
<Doc>
  <Approval xml:id="…">...</Approval>
  <Signature>
      ...
        <Reference URI="…"/>
    ...
  </Signature>
</Doc>
```

```
<Doc>
  <Approval xml:id="ap">...</Approval>
  <Signature>
     ...
       <Reference URI="#ap2"/>
    ...
    <Object>
      <Approval xml:id="ap2">...</Approval>
    </Object>
  </Signature>
</Doc>
```

Where in the SOAP message is #foo?

“wrapping attacks”



cryptographic primitives give you channels and channel transforms

they give you certain guarantees,
bind data to keys

what keys?
who owns them?
why?

Signature Encryption

powerful building blocks
use with care

Don’t bake your own unless you absolutely have to.

Avoid complexity wherever you can.

If you need transforms, name them instead of transmitting them.

“it’s signed”

“it”?

by what key?

who controls that key?

References

XML Signature Syntax and Processing
http://www.w3.org/TR/xmldsig-core1/

XML Encryption Syntax and Processing
http://www.w3.org/TR/xmlenc-core1/

XML Signature Best Practices
http://www.w3.org/TR/xmldsig-bestpractices/