CS 259

Project: XML Security

March, 2004
Jun Yoshida
(Visiting Scholar from Hitachi Ltd.)

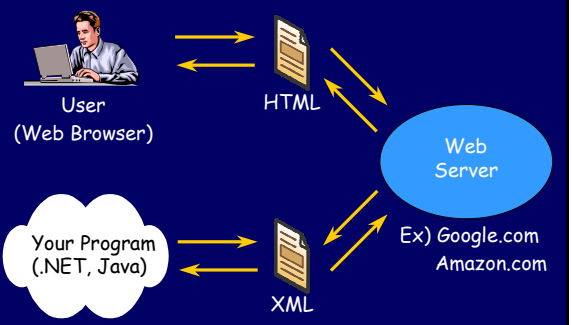
Project: XML Security

- protocol or system
  - XML Security (XML Encryption, XML Signature)
- properties which should be preserved
  - XML elements (ex. credit card number)
- kind of attacks
  - Authentication, Secrecy, Replay attack, ...
- tool or method
  - Muro
- self or team
  - myself

XML

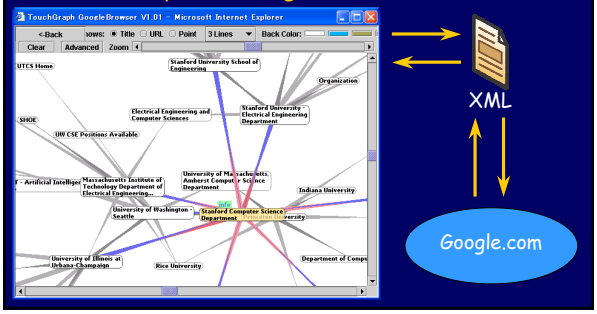
```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name> Alice </Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number> 1234 5678 9012 3456 </Number>
    <Issuer> Example Bank </Issuer>
    <Expiration> 01-05 </Expiration>
  </CreditCard>
</PaymentInfo>
```

XML Web Services (XWS)

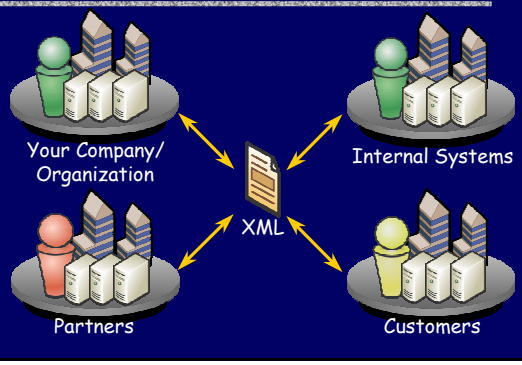


XWS Application Example

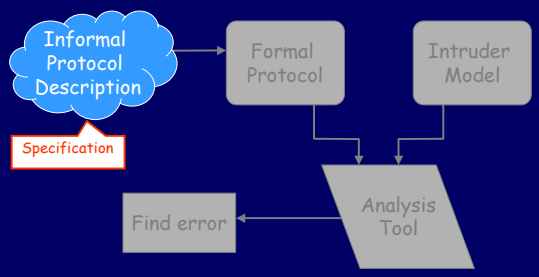
- TouchGraph.com GoogleBrowser



System Integration with XWS



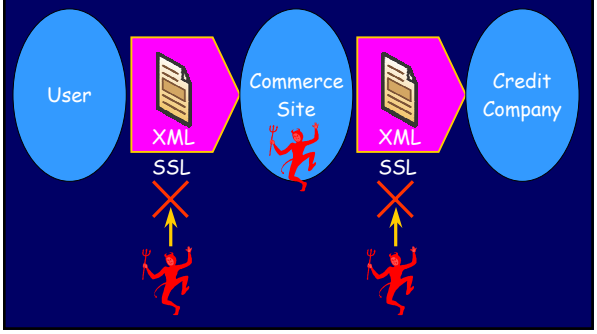
Getting Started...



Specification

- XML Encryption: 31 pages
  - http://www.w3.org/TR/xmlenc-core/
- XML Signature: 47 pages
  - http://www.w3.org/TR/xmldsig-core/
- WS-Security: 22 pages
  - http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

XML on SSL



XML Encryption --- before ---

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name> Alice </Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number> 1234 5678 9012 3456 </Number>
    <Issuer> Example Bank </Issuer>
    <Expiration> 01-05 </Expiration>
  </CreditCard>
</PaymentInfo>
```

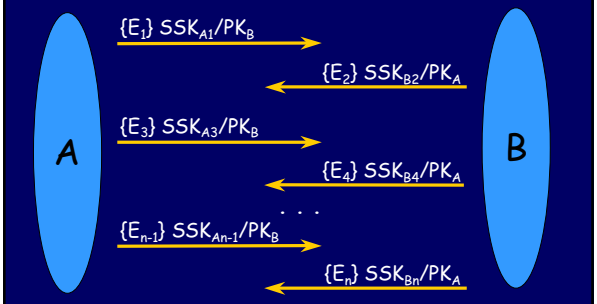
XML Encryption --- after ---

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name> Alice </Name>
  <EncryptionData
     type='http://www.w3c.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3c.org/2001/04/xmlenc#'>
     <CipherData>
       <CipherValue> A23B45C56... </CipherValue>
     </CipherData>
  </EncryptionData>
</PaymentInfo>
```

The Rule of XML Encryption

```
<EncryptionData Id? Type?>
  <EncryptionMethod/>? <!-- 3DES-CBC, AES128-CBC, ... >
  (<ds:keyInfo>
     <EncryptedKey> <!-- shared secret key encrypted with public key >
     <ds:KeyName>?
     <ds:RetrievalMethod>?
  </ds:keyInfo>)?
  <CipherData>
    <CipherValue> <!-- data encrypted with shared secret key >
  </CipherData>
<EncryptionData>
```

Basic Example

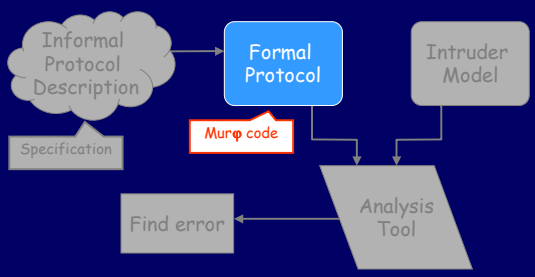


E_x: XML Element, SSK_x: Shared Secret Key, PK_x: Public Key

Shared Secret Key Example

E_x: XML Element, SSK_x: Shared Secret Key, PK_x: Public Key

From Spec. to Murφ Model



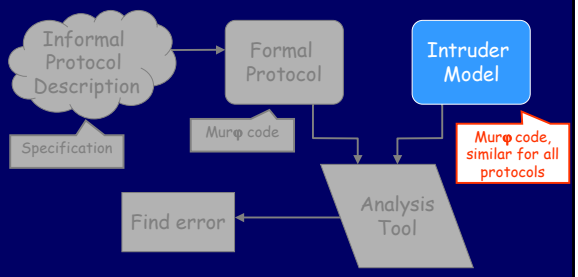
Send XML encrypted with Public Key

```
ruleset i: ClientId do
  ruleset j: ServerId do
    cli[i].state = C_SLEEP &  multisetcount (l:net, true) < NetworkSize
    ==>
    var
      outM: Message;              -- outgoing message
      cSSK: SharedSecretKeyId; -- shared secret key for client
    begin
     cSSK := GenSharedSecretKey();                    -- SSKA
      undefine outM;  outM.source := i;  outM.dest := j;  outM.mType := M_PK;
      outM.element1 := i;  outM.enKey1_1 := j;   -- {E1} PKB
      outM.element2 := cSSK;  outM.enKey2 := j; -- {SSKA} PKB
      multisetadd (outM, net);
      cli[i].state := C_WAIT_PK_MESSAGE;  cli[i].server := j;  cli[i].cSK := cSK;
    end;
  end;
end;
```

Send XML encrypted with SSKey

```
ruleset i: ClientId do
  choose j: net do
     cli[i].state = C_WAIT_PK_MESSAGE &  net[j].dest = i
    ==>
    var
      outM: Message;                -- outgoing message
      inM:  Message;                -- incoming message
      sSSK: SharedSecretKeyId; -- shared secret key for Server
    begin
      inM := net[j];  multisetremove (j, net);
      if inM.mType = M_PK then
        if inM.enKey1_1 = i & inM.enKey2 = i then       -- {E2} PKA, {SSKB} PKA
          sSSK := inM.element2;                                -- SSKB
          undefine outM;  outM.source := i;  outM.dest := cli[i].server;  outM.mType := M_SSK;
          outM.element1 := i;  outM.enKey1_2 := sSSK; -- {E3} SSKB
          multisetadd (outM, net);
          cli[i].state := C_WAIT_SSK_MESSAGE;  cli[i].sSSK := sSSK;
        end;  end;  end;  end;  end;
```

Intruder Model



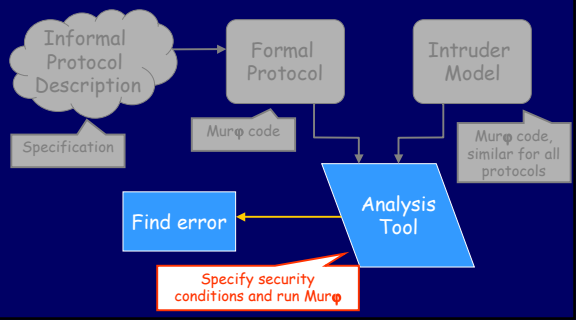
Intruder can Decrypt if Knows Key

```
ruleset i: IntruderId do
 choose j: net do
  ruleset intercept : boolean do
   rule "intruder overhears/intercepts"
    !ismember(net[j].source, IntruderId) -- not for intruder's message
   ==>
   begin
    alias msg: net[j] do -- message to overhear/intercept
     -- learn public key based messages
     if msg.mType = M_PK then
      if msg.enKey1_1 = i then  int[i].elements[msg.element1] := true;  end; -- {E_x} PK_I
      if msg.enKey2 = i then  int[i].keys[msg.element2] := true;  end;        -- {SSK_X} PK_I
     end;
     -- learn shared secret key based messages
     if msg.mType = M_SSK & int[i].keys[msg.enKey1_2] = true then        -- {E_x} SSK_I
      int[i].elements[msg.element1] := true;
     end;  end;  end;  end;  end;  end;
```

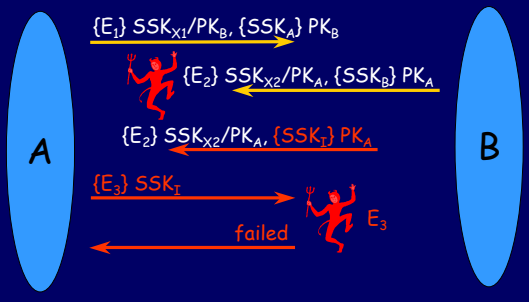
Intruder can Alter Messages

```
ruleset i: IntruderId do
 choose j: net do
  ruleset intercept : boolean do
   rule "intruder overhears/intercepts and alter ssk"
    !ismember(net[j].source, IntruderId) -- not for intruder's message
   ==>
   var
    iSSK: SharedSecretKeyId; -- shared secret key for intruder
   begin
    alias msg: net[j] do -- message to overhear/intercept
     if msg.mType = M_PK then
      iSSK := GenSharedSecretKey();
      msg.element2 := iSSK;            -- {SSK_A} PK_B => {SSK_I} PK_B
      int[i].keys[iSSK] := true;
     end;  end;  end;  end;  end;  end;
```

Running Murφ Analysis

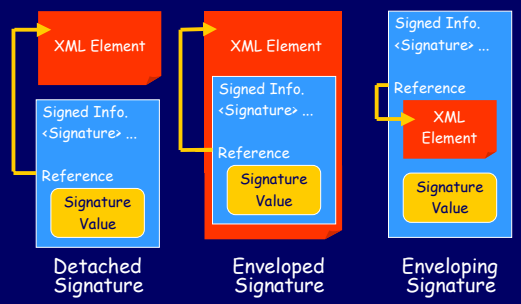


Found Error



E_x: XML Element, SSK_x: Shared Secret Key, PK_x: Public Key

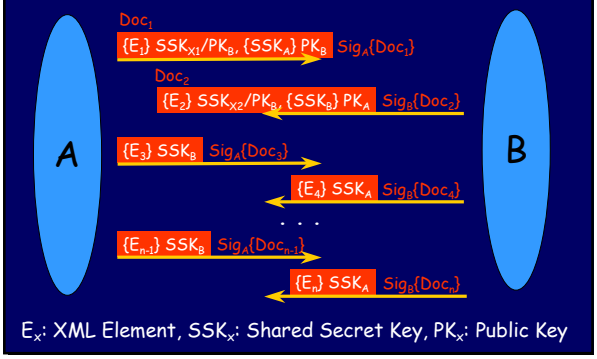
XML Signature



The Rule of XML Signature

```
<Signature ID?>
 <SignedInfo>
   <CanonicalizationMethod/>
   <SignatureMethod/>
   (<Reference (URI=)?> <!-- a data object using a URI-Reference >
     (<Transforms>)?
     <DigestMethod><DigestValue> <!-- SHA-1, SHA256, SHA512, ... >
   </Reference>)+
 </SignedInfo>
 <SignatureValue> <!-- digest encrypted with private key >
 (<KeyInfo>
   <KeyValue><X.509Data> <!-- public key and X.509 certificate >
 </KeyInfo>)
</Signature>
```

Fixed Shared Secret Key Example



E_x: XML Element, SSK_x: Shared Secret Key, PK_x: Public Key

Conclusion

- I couldn't find any error about the combination of XML Enc./Sig.
- But XML Enc./Sig. have the flexibility to use
- It's important to use correctly