Secure Global Computing with XML Web Services: Theory and Practice

Andy Gordon
Microsoft Research

EEF Global Computing Summer School
Edinburgh, Scotland, July 7-11, 2003
Lecture 1, Tuesday 11:00-12:30

Syllabus
- Lecture 1, Tuesday 11:00-12:30
  - Web services
- Lecture 2, Tuesday 16:30-18:00
  - Security protocols
  - Type systems for secrecy and authenticity properties
- Lecture 3, Wednesday 9:00-10:30
  - Analysing web services security

This course reflects joint work with M. Abadi, K. Bhargavan, C. Fournet, A. Jeffrey, and R. Pucella

1: Web Services
- Basics of Web Services
- Demo: Constructing a Web Service
- SOAP-Level Security
- Abstract vs XML Views of Simple Sample
- Demo: Signing Message using WS-Security

Part I: Basics of Web Services
Websites for computers not humans; XML versus HTML; SOAP, WSDL, and HTTP; current state of adoption

1993: Websites for Humans
- HTTP: very simple access protocol
  - Text-based is bulkier than binary: latency insensitive
- URLs: pointers to remote documents
  - Lack referential integrity of familiar pointers
- HTML: document model
  - Mixes raw data with presentational markup
- MIME: very coarse type system for web documents
  - (The Semantic Web initiative: typed pointers, roughly)
- CGI: remote procedure calls
  - But streaming data model, unlike local-area RPC

From a programming language perspective, an unusual, for 1992, model of computation, but rather effective

Websites for Computers B.X.E.
- “A web service is a website intended for computer programs rather than human beings” (Barclay et al)
- Can trace the origins of web services before the XML Era:
  - Much work on software to access the web programmatically
    - Programmatic browsing: spiders, Cardelli and Davies’ service combinators, ...
      - Every algorithmic behaviour of web browsing should be scriptable
    - URL = pointer + bandwidth
    - Programmatic data access: “screen-scraping”, Perl, Marsid’s Web
- Widely downloaded, but didn’t take off
  The thing you have to remember about pioneers is that a lot of them got shot
1998: XML
- Standard syntax for labelled ordered trees
  - Two kinds of label: elements and attributes
  - `<MyElement MyAttrib="fred">chas</MyElement>`
- Namespaces for modularity
  - URI qualifying element and attribute names
- Type systems: regular expressions for trees, roughly
  - DTDs – early, simple, but no namespaces
  - XML Schema – later, complex, but standard
  - The one that matters for SOAP web services
- Relax NG – simpler, has human readable syntax
- Query languages:
  - XPath – W3C standard
  - Many PhDs and papers...

Essential XML
- Resembles the official W3C data model, the Infoset
- XML intended originally as standard semi-structured data model for database integration
- XML as a general-purpose messaging format came later

Websites for Computers X.E.
- “XML Web Services” refers to SOAP stack of specifications:
  - SOAP – message format
    - Syntax of request, response, fault messages
  - WSDL – service description
    - Interface: function name, parameter and return types
  - UDDI – service discovery
    - Search for service by attributes (like Yellow Pages)
    - Not yet widely used in practice
  - BPEL4WS – service composition
    - Programming language for automating business processes, such as B2B order processing
    - Some sort of merger of IBM WSFL, Microsoft XLANG, and Sun WSCI... so quite complex
    - Opportunity for clean and simple alternative...?

SOAP
- Simple Object Access Protocol, early version in 1999
- Descends, in part, from earlier XML-RPC proposal
  - Deployed commercially, but lacked extensibility
  - SOAP envelope – request, response, or fault - sent from one end-point to another, possibly via intermediaries
    - Has optional header and mandatory body
    - Conventions for encoding requests and responses
    - Most commonly, SOAP sent over HTTP transport, but other underlying transports possible, such as HTTPS/SSL
    - Usually request/response pattern, others possible
    - Many implementations
    - .NET is Microsoft's implementation of SOAP web services

A Sample SOAP Request
- Says: “get me status of order 20”
- XML not meant to be read by humans, so we'll omit namespace info, and trailing brackets...
A Sample SOAP Request

- Envelope
- Body
- GetOrder
- <OrderId>20</OrderId>

Says: "get me status of order 20"
XML not meant to be read by humans, so we'll omit namespace info, and trailing brackets...that's better

A Sample SOAP Response

- Envelope
- Header
- Timestamp
- Created: 2003-03-11T23:36:06Z
- Body
- GetOrderResponse
- <orderId>20</orderId>
- <date>2003-03-11</date>
- <userId>adg</userId>

Unlike the client making the request, the server has included a timestamp in the optional Header

WSDL

- Web Services Description Language, early version in 2000
- Like IDL in CORBA/COM, etc, published by a server, and consumed by client to construct proxy
- Most of what you need to know to consume a service
- But nothing about security, for example
- A WSDL document has 5 kinds of named description
  - Type: most commonly an XML Schema
  - Message: type for the body of a SOAP envelope
  - Port type: set of operations (function signatures) with input/output message types
  - Binding: concrete transport protocol for a port type, e.g., SOAP over HTTP, HTTP GET, HTTP POST
  - Service: set of ports, each a binding plus address

AddInt ::= <AddInt> <a>int</a> <b>int</b></AddInt>
AddIntResponse ::= <AddIntResult>int</AddIntResult>

(message name="AddIntSoapIn")
<part name="parameters" element="AddInt"/>
<message name="AddIntSoapOut">
<part name="parameters" element="AddIntResponse"/>

(portType name="AddNumbersSoap")
<operation name="AddInt">
<input message="AddIntSoapIn"/>
<output message="AddIntSoapOut"/>

(binding name="AddNumbersSoap" type="AddNumbersSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<operation soapAction="http://microsoft.com/.../AddInt"/>
<operation names="AddIntSoapAction"/>
<service name="AddNumbers">
<port name="AddNumbersSoap" binding="AddNumbersSoap">
<soap:address location="http://localhost/.../usernamesigning.asmx"/>

Demo: Constructing a Web Service

High-level RPC model; SOAP messages; WSDL descriptions

Next: Three ways in which SOAP web services are being used, and what's actually new?

(1) On the Public Internet

- Aim: outsource user-interface to specialists
- Specialist storefronts
  - Ex: Products sold through Amazon, but associate site gets commission
  - 25k developers since Jul’02
- Specialist visualizations
  - Ex: TouchGraph re Google
- Specialist smart clients
  - Ex: SourceForge project
(2) Within a Private Intranet

- **Aim:** XML as common language to integrate systems obtained from different vendors
- **Ex:** single service to calculate quotes uniformly no matter how customer contacts an insurance company
  - call centre
  - sales force
  - website
- Today, biggest growth area for web services, in part because there are few new security concerns

(3) Between Private Intranets

- **Aim:** support inter-institution workflows
- **E-business transactions**
- **E-science GRID**
- This is the hardest of the three cases to secure
- **Single sign-on:** scientist may access 100 sites
- **Federation:** e.g., can company A permit just certain company B employees to access shared service?
- **Delegation:** e.g., can I permit online travel agent to consult my calendar service, make bookings?

Web Services: What’s New?

- Though their core is roughly XML-encoded RPC – rather old! – what’s new about SOAP web services is the combination of:
  - Vendor-neutral, Internet-scale, high-level tools
- Signs of fervour,
  - Wide support from commercial & OSS suppliers
  - Weekly news of progress at OASIS and W3C
- yet reasons for caution,
  - Cost of SOAP encoding?
  - Lack of SOAP security?
- and some competition,
  - Fielding’s REST: HTTP-based web services
  - ebXML: XML version of earlier UN EDI format

Part II: SOAP-Level Security

Transport- versus application-level security; WS-Security for embedding a range of security tokens within SOAP headers

The 2002 Security Story

- The 2002 best practice was to build secure web services using an SSL (as in https) transport
- SSL gives transport- not application-level security
  - Messages secured point-to-point not end-to-end
  - Messages cannot be filtered or routed
  - Messages not encrypted in files or databases
  - Moreover, SSL has scalability problems
- Party line (aka Web Services Security Roadmap) security within SOAP envelopes is better:
  - For end-to-end, application-level security, independently of underlying transports

WS-Security: Syntax Summary

```
SecurityToken ::= 
  <UsernameToken
    <Username>String</>
  ?<Password Type="PasswordType">String</>
  ?<Created>String</>
  ?<Nonce>Base64Binary</>
  | <BinarySecurityToken>Base64Binary</>
  | <SecurityTokenReference>
    <Reference URI="Uri">
      <KeyInfo>*KeyInfoItem</>
      <Signature>SignedInfo SignatureValue</>
    <Reference URI="Uri"></
    <ReferenceList>+<DataReference URI="Uri"></
  | EncryptedKey
  | EncryptedData
```

**Security** element is child of SOAP Header

- **UsernameToken** identifies particular user
- **BinarySecurityToken** embeds an existing format such as an X509 public-key certificate, or a Kerberos certificate
Outline Security Architecture

- A wants to talk to B but may have insufficient security tokens to satisfy B's security policy
  - Trust policy tells which tokens are valid
  - Authorization policy tells if valid tokens suffice for request
- So A gets tokens from Security Token Server STS
- Abstracts from underlying crypto technologies
- Will consider three party protocols (but not in XML form) in Lecture 2

Security Spec Overview

- Several specs released since summer 2002:
  - **WS-Security**: message integrity, confidentiality, authentication; security token attachment, both XML (SAML, X.509) and binary (Kerberos, X.509)
  - **WS-Trust**: request and issue security tokens, manage trust relationships
  - **WS-SecureConversation**: establish and share security contexts, derive session keys
  - **WS-SecurityPolicy**: security requirements
- Plus several implementations:
  - Microsoft WSE (Web Service Enhancements)
    - RTW Dec 2002, free plugin for VS.NET 2002 and later
    - Product implementing WS-Security, WS-Routing, and DIME attachments
  - Others from IBM, Verisign, etc

Sample Security Goals

- Suppose a human A with password p uses a client I to invoke a web service at URL S
  - S = http://www.bobspetshop.com/ws/orderstatus.asmx
- Without some kind of authentication, anybody could request the private details of anyone else's order
- Simple solution to require p-based signature of:
  - Message body
    - to show request from A, and has not been modified
  - Timestamp
    - to detect replays, with cache of recent messages
  - Web server S
    - to detect redirection from another server

Part III: Abstract vs XML Views of Simple Sample

We illustrate WS-Security by explaining the design of a simple but typical authentication protocol

Get Order

Order Info

Scope of our Threat Model

- The threat model is an attacker who can replay, redirect, assemble new messages, but cannot brute force secrets such as passwords
  - Formal statement usually credited to Dolev and Yao 1983, but basic ideas in Needham and Schroeder's pioneering 1978 work on crypto protocols
  - Can verify that crypto protocols establish various safety properties in spite of such an attacker:
    - Message authentication - against unauthenticated access
    - Message integrity - against parameter manipulation
    - Message confidentiality - against eavesdropping
    - Message freshness - against replays
- Like all formal or informal methods, certain threats lie outside the model, and must be addressed separately
  - Disclosure of configuration data
  - Unauthorized access via SQL injection or cross-site scripting

An Abstract Protocol

- Security goal expressed as a correspondence
  - Each end-event corresponds to a begin-event
- In Lecture 2, we will explain how to formalize diagrams like this within the pi calculus, and verify safety with respect to any Dolev-Yao opponent
  - who can replay, redirect, assemble new messages, but cannot brute force secrets such as passwords
  - To justify this assumption, p must be a strong password, stored on Client I, and not just a weak memorizable password subject to dictionary attacks
  - Next, a wire-level view of the implementation...
Aside: Secure Hash Functions

- A hash function is a pseudo-random function mapping an arbitrary length input to n bits.
- Additionally, a secure hash function satisfies:
  - One-way: For given y, computationally infeasible to find x with y = h(x).
  - Weak collision resistance: For given x, computationally infeasible to find x' with h(x) = h(x').
  - Collision resistance: It is computationally infeasible to find any x, x' with h(x) = h(x').
- Examples: MD5 (n=128), SHA-1 (n=160).
- In Dolev-Yao formal models, a secure hash function is represented as a symbolic constructor with no inverse.

Aside: WS-Routing

- Client requests a service from server A, which in fact routes it on to server B.
- Routing determined by a referral cache at A.
- If B needs to be taken offline, we can update the cache to point at C, transparently to the client.
- Aside to an aside: DIME enables non-XML, binary attachments.
Demo: Signing with WS-Security

To try this at home, you need:
Windows XP Pro (not Home Edition)
Visual Studio .NET or Visual Studio .NET 2003

1: Summary

- SOAP and WSDL implement a fairly standard RPC mechanism on top of HTTP
- but that has achieved unprecedented interoperability, and works on a global scale
- XML-DSIG, XML-ENC, and WS-Security are a basis for end-to-end security guarantees, from encryption, signatures, and embedded security tokens;
- novel features include abstraction from underlying crypto technologies, and flexibility of signatures
- XML or not, new crypto protocols are often wrong
  - Lecture 2: type-theoretic techniques for verification
  - Lecture 3: application to SOAP-level security protocols

1: Resources

- Cardelli and Marais websites, for service combinators, and WebL
  - http://www.luca.demon.co.uk/
- Standards tracks and whitepaper
  - http://www.w3.org/2002/ws/
  - http://www.oasis-open.org
- Abliteboul of INRIA and Lehman of IBM, on web services:
- General introduction to computer security (one of many…)
- My Top Three Web Service Blogs
  - http://weblogs.cs.cornell.edu/AllThingsDistributed/index.rdf
  - http://www.scottishlass.co.uk/rss.xml

End of Part 1

Secure Global Computing with XML Web Services: Theory and Practice

Andy Gordon
Microsoft Research
EEF Global Computing Summer School
Edinburgh, Scotland, July 7-11, 2003
Lecture 2, Tuesday 16:30-18:00

2: Types for Crypto Protocols

- Basic ideas of crypto protocols
- How the Dolev-Yao threat model can be rephrased within a process calculus
  - Attacker can modify, decrypt, replay network traffic, but cannot simply guess keys (even by brute force)
- How secrecy properties can be proved by typing
  - Continuation of long literature on security levels
- How authenticity properties can be proved by typing
  - Continuation of long literature on effect systems
Basic Ideas of Crypto Protocols

Protocols are quite short, and are often specified by message sequences.
Even assuming perfect crypto algorithms, replay and impersonation attacks are possible.

A Little History

- 1978: Needham and Schroeder invent authentication protocols; descendants eg SSL now widely deployed
- 1981: attack found on symmetric key protocol (!)
- 1987: Burrows, Abadi, and Needham invent a formal logic of authentication (“BAN logic”)
- 1995: attack found on asymmetric key protocol (!!!)
- 1997: Several formal methods for “BAN problem”
  - Tradeoff between accuracy and approximation: theorem-proving versus model-checking
- Today: Many formal methods now developed
  - Several deliver both accuracy and automation
  - Examples: Cohen’s TAPS, Blanchet’s Proverif

Ex I: Sending Messages

Message 1: B → A: NB
Message 2: A → B: \( (\text{msg,NB})_{\text{KAB}} \)

- A and B share a secret key \( \text{KAB} \)
- A wishes to send a series of instructions to B
- We wish to secure against tampering and replays, and authenticate A’s identity to B
- Encryption prevents tampering, and helps guarantee authenticity
- A nonce challenge prevents replay attacks

Ex II: Server-Based Login

Principal A wishes to prove its presence to principal B, via an authentication server S
Although A and B have no keys in common, the protocol can exploit secret keys KAS and KBS that A and B share with S

Ex II: Message Sequence

Message 1: A → B: A
Message 2: B → A: NB
Message 3: A → B: \( (\text{NB})_{\text{KAS}} \)
Message 4: B → S: B \( (A,\text{NB})_{\text{KCS}} \)_{\text{KAS}}
Message 5: S → B: \( (\text{NB})_{\text{KBS}} \)

Message 5 meant to prove to B that A is currently running the protocol
But it doesn’t mention A, so by manipulating parallel sessions, an attacker C may login as A

Attacking Ex II

1. C → B: A
2. B → C: NB
3. C → B: \( (\text{NB})_{\text{KCS}} \)
4. B → S: B \( (A,\text{(NB)KCS})_{\text{KBS}} \)
5. S → B: \( (\text{NB})_{\text{KBS}} \)

Here A is offline, but insider C runs two parallel sessions which end with B believing A has logged in.
To fix, include the identity of A in messages 3 and 5.
Discussion

- We include the additional names in Ex II to prevent impersonation attack.
- Infamously, crypto protocols are vulnerable to attack, without breaking the underlying crypto algorithms.
- Ongoing need for protocol verification: new technologies force invention of new protocols, and change security assumptions.
- The message notation is abstract, but not completely precise; e.g., authentication goals left implicit.
- So, how might we specify properties formally?

Formally Specifying Security Properties

We might use any one of a great many formalisms. As it’s the basis of several type systems for security, we pick the untyped spi-calculus of Abadi and Gordon (CCS’97, Concur’97).

Related process calculi include the sjoin-calculus, the applied π-calculus, and several others.

The Spi-Calculus in One Page

The statement $\text{decrypt } M = (x)i; P$ means:
- “If $M$ is $(x)i$, for some $x$, run $P$“.

Decryption evolves according to the rule:
$\text{decrypt } (L)_i = (x)i; P \rightarrow P(x-L)$

- Decryption requires having the key $N$.
- Decryption with the wrong key gets stuck.
- There is no way to extract $N$ from $(L)_i$.
- Abstraction introduced by Dolev and Yao (1983).

Specifying Authenticity

<table>
<thead>
<tr>
<th>Event</th>
<th>A begins</th>
<th>B receives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message 1</td>
<td>A sends</td>
<td>(msg, NB)</td>
</tr>
<tr>
<td>Message 2</td>
<td>B receives</td>
<td>(msg, NB)</td>
</tr>
<tr>
<td>Event 1’</td>
<td>B ends</td>
<td>(A sending msg to B)</td>
</tr>
</tbody>
</table>

Each end-assertion to have distinct, preceding begin-assertion with same label.

Attacks (replays, impersonations) show up as violations of these assertions.

Named correspondence assertions by Woo and Lam, but also injective agreements by Lowe.

Authenticity Specified in Spi

```
sys(msg1, …, msgn) =
  (send(msg1, k) | … | send(msgn, k) | repeat recv(k))
  imp net(no);
begin “Sender sent msg”;
out net((msg, no));
```

Secrecy Specified in Spi

For all $(msg_1, …, msg_i)$, $(msg_{i+1}, …, msg_n)$,
$\text{sys(msg}_1, …, \text{msg}_i) = \text{sys(msg}_{i+1}, …, \text{msg}_n)$.

- No opponent $O$ should be able to distinguish runs carrying different messages.
- We interpret $P=Q$ as may-testing equivalence.
  - A test is a process $O$ plus a channel $c$.
  - A process passes a test $(O, c)$ if $P|O$ may eventually communicate on $c$.
  - Two processes equivalent iff they pass the same tests.
- In fact, our example fails this spec…
**A Small Information Leak**

- Consider $\text{sys}(\text{msg}_1, \text{msg}_2)$. The opponent certainly cannot obtain either of the messages in the clear, but can it tell whether they are equal?
  - One may reason that A's inclusion of the nonces always distinguishes $(\text{msg}_1, \text{no}_1)$ and $(\text{msg}_2, \text{no}_2)$.
  - But the opponent may feed its own nonce $\text{no}$ twice to A, cause A to emit $(\text{msg}_1, \text{no})$ and $(\text{msg}_2, \text{no})$, and hence can tell whether $\text{msg}_1 = \text{msg}_2$.
  - To fix this, A sends $(\text{msg}, \text{no}, \text{co})$, for some fresh confounder $\text{co}$ instead of simply $(\text{msg}, \text{no})$.

**Many Variations Are Possible**

- There is a vast literature on equationally defined information flow, e.g., “non-interference properties”
  - Focardi and Gorrieri (JCS 1994) were pioneers in the setting of process calculi
  - As usual, the formalism (choice of equivalence and operational semantics) may abstract too much
  - Our spec is insensitive to covert timing channels
  - Mitchell et al study more refined calculi (CCS98...) still, we now have specs of authenticity and secrecy...

**Secrecy Types**

Introduction to “Secrecy by Typing in Security Protocols” by Abadi (JACM 1999)

Draws on earlier work on security levels to control information flow (Denning,...), and on types for $\pi$ (Milner, Pierce and Sangiorgi,...)

**Origins of Type Theory**

Cambridge 1901: Russell uncovers a paradox in Frege's system of arithmetic

Later, 1908, he proposes his Theory of Types to patch the bug

Whoops!

**Three Security Levels**

- There are three types $T$ for data
  - $\text{Un}$ for public data known to the opponent
  - $\text{Secret}$ for private data assumed not to leak to the opponent
  - $\text{Top}$ for arbitrary data, either $\text{Un}$ or $\text{Secret}$
- Judgment $E \vdash M : T$ means message $M$ has type $T$
- Judgment $E \vdash P \text{ ok}$ means process $P$ well-typed

**Typing Public Channels**

- The rules for communication on public channels
  - allow communication of public $\text{Un}$ data
  - but prevent communication of $\text{Secret}$ or $\text{Top}$ data
- Informally:
  - If $M:\text{Un}$ and $N:\text{Un}$ then $\text{out} M N \text{ ok}$
  - If $M:\text{Un}$ and $x:\text{Un} \vdash P \text{ ok}$ then $\text{inp} M (x:\text{Un}); P \text{ ok}$
Typing Secret Keys

- The rules for cryptography using secret keys
  - preserve types while sending data on public channels
  - force inclusion of suitable confounders

Informally:
- If $M_1:\text{Secret}$ and $M_2:\text{Top}$ and $M_3:\text{Un}$ and $N:\text{Secret}$ and $n$ a fresh confounder then $(M_1,M_2,M_3,n)\text{Un}$
- If $M:\text{Un}$ and $N:\text{Secret}$ and $x_1:\text{Secret},x_2:\text{Top},x_3:\text{Un},x_4:\text{Top}$ \(\vdash P\text{ok}\) then decrypt $M$ as $(x_1:\text{Secret},x_2:\text{Top},x_3:\text{Un},x_4:\text{Top})\text{Un}$

Secrecy for Ex I

<table>
<thead>
<tr>
<th>Event 1</th>
<th>A begins</th>
<th>(A sending msg to B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message 1</td>
<td>$B \rightarrow A$</td>
<td>NB</td>
</tr>
<tr>
<td>Message 2</td>
<td>$A \rightarrow B$</td>
<td>(msg, NB)_{	ext{local}}</td>
</tr>
<tr>
<td>Event 1’</td>
<td>B ends</td>
<td>(A sending msg to B)</td>
</tr>
</tbody>
</table>

Each end-assertion to have distinct, preceding begin-assertion with same label
Attacks (replays, impersonations) show up as violations of these assertions
Named correspondence assertions by Woo and Lam, but also injective agreements by Lowe.

Secrecy by Typing

Theorem (Abadi)

If $E=x_1:Un…,x_n:Un$ and $E,y:Top \vdash P(y)\text{ ok}$ and $E \vdash M,M':Top$ then $P(M) \approx P(M')$.

- Beware, does not say no flow from $\text{Secret}$ to $\text{Un}$
- Instead, provided public channels and secret keys have types Un and $\text{Secret}$, respectively, then Top data, e.g., the protocol payload, not observable
- In our example, net:Un,msg:Top \vdash sys(msg)\text{ ok}$, so we obtain desired secrecy property (also n-ary case)
- Proof relies on Opponent Typability; any $O$ can be typed if all its variables of type $\text{Un}$

Authenticity Types for Symmetric-Key Crypto

All the work on authenticity types is by Gordon and Jeffrey.
Work on types for correspondence assertions and symmetric-key crypto at MFPS’01, CSFW’01.
Draws on earlier work on effect systems for resource control in functional languages.

Authenticity Specified in Spi

<table>
<thead>
<tr>
<th>Event 1</th>
<th>A begins</th>
<th>(A sending msg to B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message 1</td>
<td>$B \rightarrow A$</td>
<td>NB</td>
</tr>
<tr>
<td>Message 2</td>
<td>$A \rightarrow B$</td>
<td>(msg, NB)_{	ext{local}}</td>
</tr>
<tr>
<td>Event 1’</td>
<td>B ends</td>
<td>(A sending msg to B)</td>
</tr>
</tbody>
</table>

Each end-assertion to have distinct, preceding begin-assertion with same label
Attacks (replays, impersonations) show up as violations of these assertions
Named correspondence assertions by Woo and Lam, but also injective agreements by Lowe.
The Two Main Judgments

- Judgment $E \vdash M : T$ means message $M$ has type $T$.
- Judgment $E \vdash P : \{L_1, \ldots, L_n\}$ means process $P$ has effect $\{L_1, \ldots, L_n\}$, a (multiset) bound on the events that $P$ may end but not begin.

- If $L : T$ then $\text{end} \ L : \{L\}$.
- If $L : T$ and $P : e$ then $\text{begin} \ L; P : e$.

Metaphor: $\text{end}$'s and $\text{begin}$'s like costs and benefits that must be balanced.

Typing Symmetric Crypto

- We drop $\text{Secret}$ and $\text{Top}$ and add some new types, including pairing and tagged unions.
- Messages of type $\text{Un}$ are data known to the untyped opponent, as before.
- Messages of type $\text{Key}(T)$ are names used as symmetric keys for encrypting type $T$.

If $M : T$ and $N : \text{Key}(T)$ then $\text{decrypt} \ M$ as $(x : T) \{x\} N : e$.

Typing a Nonce Handshake

- Messages of type $\text{Nonce}[L_1, \ldots, L_n]$ prove begin-events labelled $L_1, \ldots, L_n$ have previously occurred.
- Example nonce lifecycle:
  1. Receiver publishes new $\text{Un}$.
  2. Sender receives $\text{Un}$, asserts $\text{begin} \ L$, casts nonce into $\text{Nonce}[L]$, returns within ciphertext.
  3. Receiver decrypts message, checks just once for presence of $\text{Nonce}[L]$, then asserts $\text{end} \ L$.
- Effect in the $\text{Nonce}[L]$ type allows transfer.
- Each $\text{cast}$ is a cost; each $\text{check}$ is a benefit.

Semantics of $\text{cast}$

The process $\text{cast} x$ to $(y : \text{Nonce} e) : P$ evolves into the process $P(y \leftarrow x)$.

- Only way to make name of type $\text{Nonce} e$.
- The name is a proof events in $e$ have happened.

It "costs" the effect $e$:

If $E \vdash x : \text{Un}$ and $E \vdash y : \text{Un}$ and $E \vdash P : e'$ then $E \vdash \text{cast} x$ to $(y : \text{Nonce} e) : P : e + e'$.

Only kind of type-cast in the system.

Semantics of $\text{check}$

Process $\text{check} x = y : P$ evolves into process $P$ if $x = y$; but otherwise gets stuck.

It "pays for" the effect $e$ in $P$:

If $E \vdash x : \text{Nonce} e$ and $E \vdash y : \text{Un}$ and $E \vdash P : e'$ then $E \vdash \text{check} x = y : P : e - e'$.

For each $\text{new}(y : \text{Un}) : P$, we require that the name $y$ be used in a $\text{check}$ at most once.

Enforced by adding a new kind of effect; details omitted.

Typing Ex I

- $\text{MyKey} : \text{Key}(m : \text{Msg}, \text{MyNonce}(m))$.

```plaintext
send(msg:Msg,MyKey)[] &
new(no:Un); out net(no); inp net(u:Un);
```

```plaintext
decrypt u is (msg:Msg, no:MyNonce(msg));
```

```plaintext
check no is no; end "Sender sent msg";
```
### Authenticity by Typing

A process $P$ is **safe** iff in every execution trace, there is a distinct `begin` $L$ for every `end` $L$.

A process $P$ is **robustly safe** iff for all `begin`- and `end`-free opponents $O$, $P|O$ is safe.

**Theorem (Robust Safety)**

If $x_1, \ldots, x_n : Un \int sys(x_1, \ldots, x_n)$ then $P$ is robustly safe.

For Ex I, we can check the following:

$$net, msg_1, \ldots, msg_n : Un \int sys(msg_1, \ldots, msg_n) : []$$

### Typing Ex II

**Assertion 1** $A$ begins "A proving presence to $B"

**Message 1** $A \rightarrow B$: $A$

**Message 2** $B \rightarrow A$: $NB$

**Message 3** $A \rightarrow B$: $(tag_3(B, NB))_{tag_4}$

**Message 4** $B \rightarrow A$: $(tag_4(A, tag_3(B, NB))_{tag_5})$

**Message 5** $A \rightarrow B$: $(tag_5(A, NB))_{tag_6}$

**Assertion 2** $B$ ends "A proving presence to $B"

$PrincipalKey(p) \triangleq Key(Cipher3(p) + Cipher4(p) + Cipher5(p))$

$Cipher3(A) \triangleq (B, Un, NB, Nonce("A proving presence to $B")$

$Cipher4(B) \triangleq (A, Un, cipher-Un) -- seems redundant$

$Cipher5(B) \triangleq (A, Un, NB, Nonce("A proving presence to $B")$

### Typing Ex II, again

**Assertion 1** $A$ begins "A proving presence to $B"

**Message 1** $A \rightarrow B$: $A$

**Message 2** $B \rightarrow A$: $NB$

**Message 3** $A \rightarrow B$: $(tag_3(B, NB))_{tag_4}$

**Message 4** $B \rightarrow A$: $(tag_4(A, tag_3(B, NB))_{tag_5})$

**Message 5** $A \rightarrow B$: $(tag_5(A, NB))_{tag_6}$

**Assertion 2** $B$ ends "A proving presence to $B"

$PrincipalKey(p) \triangleq Key(Cipher3(p) + Cipher4(p) + Cipher5(p))$

$Cipher3(A) \triangleq (B, Un, NB, Nonce("A proving presence to $B")$

$Cipher4(B) \triangleq (A, Un, cipher-Un) -- seems redundant$

$Cipher5(B) \triangleq (A, Un, NB, Nonce("A proving presence to $B")$

### Implementation

- Checked a standard suite of symmetric key protocols
- Re-discovered known bugs, found redundancies

### Abstracting Asymmetric Crypto

Terms $Enc_k$ and $Dec_k$ extract the two parts of an asymmetric key-pair, the name $k$

Term $(M)_N$ is $M$ encrypted with key $N$

Process $decrypt L$ is $(X)_N:P$ attempts to decrypt $L$ with key $N$

$decrypt (M|Enc_k)$ is $(X)_Dec_k:P \rightarrow P(x \leftarrow M)$

Fairly standard model; has known limitations

Same operational semantics models both public-key crypto and digital signature applications
Typing Asymmetric Crypto

Names of type \texttt{KeyPair}(T) represent a key-pair for transforming T data.

Terms of type \texttt{EncKey}(T) and \texttt{DecKey}(T) are encryption and decryption keys, respectively.

- If : \texttt{KeyPair}(T) then \texttt{Enc} p:\texttt{EncKey}(T).
- If \texttt{p} : \texttt{KeyPair}(T) then \texttt{Dec} p:\texttt{DecKey}(T).
- If : T and : \texttt{EncKey}(T) then \texttt{M}_{\texttt{Enc}}\texttt{Un}.
- If : T and \texttt{N}: \texttt{DecKey}(T) and \texttt{x}: T+ P well-typed, then so is \texttt{decrypt} M as \texttt{M}_{\texttt{Enc}}\texttt{P}.

Jargon: Public versus Tainted

We (re-) introduce a subtype order T : U

- If : T and T : U then U : T

Hence, we characterize data that may flow to or from the opponent:

- Let a type T be public iff \texttt{T}_{\texttt{Un}}
- Let a type T be tainted iff \texttt{Un}_{\texttt{T}}

Ex: \texttt{Un} is both public and tainted
Ex: \texttt{Top} is tainted but not public

Subtyping Asymmetric Keys

Variance rules reminiscent of types for input and output channels (see Pierce and Sangiorgi)

- If T : U then \texttt{EncKey}(U) : \texttt{EncKey}(T) (contravariant)
- If T : U then \texttt{DecKey}(T) : \texttt{DecKey}(U) (covariant)
- \texttt{KeyPair}(T) neither co- nor contravariant.

For both Opponent Typability and to allow publication of keys for \texttt{Un}, both \texttt{EncKey(\texttt{Un})} and \texttt{DecKey(\texttt{Un})} are tainted and public.

Analysing our Abstraction

Can prove the following:

- (PK) \texttt{EncKey}(T) public iff \texttt{T} tainted
- (DS) \texttt{DecKey}(T) public iff \texttt{T} public

So how can we apply key-pairs of type \texttt{KeyPair}(T)

- If (PK) but not (DS): public-key crypto
- If (DS) but not (PK): digital signature
- If both (PK) and (DS): have \texttt{T}_{\texttt{Un}} dangerous!
- If neither, model degenerates to symmetric key

Ex III: Authentication by Certs

Server A authenticates to client B via certificate from CA

| Assertion 1 | A begins | (A sending msg to B) |
| Message 1 | B \rightarrow A | \texttt{NB} |
| Message 2 | A \rightarrow B | \texttt{A,KKA}_{\texttt{KA}} \texttt{1(msg,B,\texttt{NB}_{\texttt{KA}} \texttt{1})} |
| Assertion 2 | B ends | (A sending msg to B) |

- pKCA, pKA -- key-pairs
- KCA \triangleq \texttt{Dec} pKCA -- CA’s verification key (known to B)
- KCA \triangleq \texttt{Enc} pKCA -- CA’s private signing key
- KA \triangleq \texttt{Dec} pKA -- A’s verification key (initially unknown)
- KA \triangleq \texttt{Enc} pKA -- A’s private signing key

Ex III: Types for the Key-Pairs

As before, a name of type \texttt{Nonce}(A sending msg to B) bears witness to a distinct preceding begin-event labelled \texttt{A sending msg to B)}

- (DS) applies to both key-pairs, since \texttt{AuthMsg(A)} and \texttt{DecKey(AuthMsg(A))} are public (assuming T public)
- So verification keys public, signing keys private

Type-checking verifies that A can authenticate to B
Authenticity Types: Two Refinements

With symmetric-key protocols, nonces can be public.
With public-key, nonces may need to be private.
Hence, we need new nonce types and new effects.

Public Nonces Insufficient

| Assertion 1 | A begins | (A sending msg to B) |
| Message 1   | B → A: NB |
| Message 2   | A → B: (A, msg, NB) |
| Assertion 2 | B ends | (A sending msg to B) |

Replaced symmetric encryption with asymmetric
B has now no reason to believe message 2 from A
Unsafe, and indeed fails to type-check
(DS) rather than (PK) holds
since payload type is public but untainted

Ex IV: Encrypt Outgoing Nonce

| Assertion 1 | A begins | (A sending msg to B) |
| Message 1   | B → A: NB |
| Message 2   | A → B: (A, msg, NB) |
| Assertion 2 | B ends | (A sending msg to B) |

Now, B reasons that since only A can obtain NB from (NB)K,A, A must have sent Message 2.
This protocol is safe.
To type-check it, we need new secret but tainted types for the nonce challenge and response.

Typing Private, Tainted Nonces

\[ \text{AuthMsg}(P) \triangleq \text{msg}1(N: \text{PrivChall}[]) + \text{msg}2(\text{msg: Top, Q: Un, N:PrivResp(Q sending msg to P))} \]
\[ \text{KA: EncKey(AuthMsg(A))} \]
\[ \text{KB: EncKey(AuthMsg(B))} \]

Names of type \text{PrivChall}[] are private but tainted challenges.
Names of type \text{PrivResp}[L] are private but tainted responses, witness to a distinct begin-event L.
With these typings, can verify the protocol
For (PK), taint \text{AuthMsg}(P), by assuming \text{msg: Top}

Adding Trust Effects

\[ \text{AuthMsg}(P) \triangleq \text{msg}1(N: \text{PrivChall}[]) + \text{msg}2(\text{msg: Top, Q: Un, N:PrivResp(Q sending msg to P), msg: T}) \]
\[ \text{KA: EncKey(AuthMsg(A))} \]
\[ \text{KB: EncKey(AuthMsg(B))} \]

The effect \text{msg: T} asserts the existing name \text{msg} has type T
Before checking the nonce, B knows only that \text{msg: Top}
If the nonce-check fails, B knows nothing more about \text{msg}
If it succeeds, B can downcast \text{msg} to type T

Assessment of Authenticity Types

Benefits
- Familiar program/type-check/debug cycle
- Little human effort per protocol
- No bound on size of opponent or protocol
- Types are as intuitive as BAN formulas
- Directly check implementations

Limitations
- No automatic discovery of attacks
- Type inference problem still open
- Usual Dolev-Yao perfect encryption assumptions
- Incompletenesses, like any type system
2: Conclusions

The Dolev-Yao threat model of crypto protocols may be formalized within process calculi. Secrecy and authenticity properties may be formalized using behavioural equivalences and events. Suitable type systems may establish both, even in the presence of an untyped attacker. There are many other formal methods; see our papers for a discussion.

DePaul/MSRC Cryptyc Project working to develop type-checked protocol implementations.
DePaul funded from NSF Trusted Computing program.

2: Resources

- Cryptyc
  - http://cryptyc.cs.depaul.edu
  - http://research.microsoft.com/~adg/cryptyc.htm
- Blanchet's Proverif verification tool
  - http://www.mpi-sb.mpg.de/~blanchet/

End of Part 2

3: Analysing web services security
- Validating a security abstraction by typing
- XML with symbolic crypto
- Username signing example
- X509 signing example
- All part of our Samoa project...

MSRC Samoa Project
- Goal: exploit advances in the analysis of security protocols in the practical setting of XML web services.
- Outcomes so far:
  - An implementation of declarative security attributes for web services
  - Design of a logic-based approach to checking SOAP-based protocols
    - K. Bhargavan, C. Fournet, A. Gordon, A semantics for web services authentication, in preparation
  - Actionable feedback to internal group
- http://Securing.WS
Part I: A Web Service Security Abstraction

With Riccardo Pucella

An informal design, and pre-WS-Security implementation

We formalize the application-level within an object calculus, and the SOAP-level within the spi-calculus.

The validation is a type-preserving semantics of the object calculus in the spi-calculus.

A Security Abstraction

class BankingServiceClass {
    string callerid;
    [WebMethod] [SecurityLevel(Level=Auth)]
    public int Balance (int account)
    [WebMethod] [SecurityLevel(Level=AuthEnc)]
    public string Statement (int account)
}

A SOAP-Level Implementation

we assume key KAB shared between A and B
We also consider key establishment with certs
Messages 1/2 establish security context: fresh nonce
Could avoid first roundtrip by including timestamps
Messages 3/4 are the actual call/return
Implemented using SOAP extensions in VS.NET

An AuthEnc Envelope

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope …>
<soap:Header>
<DSHeader …>
<callerid>Alice</callerid>
<calleeid>Bob</calleeid>
<np>13</np>
<nq>-1</nq>
<signature>4E:00:6F:00</signature>
</DSHeader>
<soap:Body>
CA:CE:00:33:64:8C:03:F3:30:0C:94:39:04:CB:39:
</soap:Body>
</soap:Envelope>

What Do We Have So Far?

We have outlined a new “security abstraction”
Defined by custom attributes on web methods
Implemented by SOAP extensions

Next, to validate using types:
We formalise the abstraction as an object calculus
We specify its semantics by translation into spi
Since the translation preserves typings, attacks representable in spi are impossible
Verification of formal model, not running code
Still, verified implementations are coming within reach

Back to our SOAP protocol

Specify authenticity via event correspondences

Verify via generic types for crypto keys and nonces
A Calculus of Web Services

- Object calculi are OO-langs in miniature
- Small enough for formal proof
- Big enough for study of specific features
  - Abadi and Cardelli “A Theory of Objects”
  - Igarashi, Pierce, and Wadler “Featherweight Java”
  - Gordon and Syne BIL; ...
- We include an application-level view of a web service
- A service is neither an object nor a value
- WSDL neither object-oriented nor higher-order
- But a service implemented via a server class
- Recall the BankingServiceClass
- And may be accessed directly or via a proxy class

An Informal Semantics

- How to evaluate a body $b$ as principal $p$:
  - To evaluate $v$, terminate with $v$ at once.
  - To evaluate $\text{let } x := \alpha \text{ in } b(x)$, first evaluate $\alpha$ as $v$ to $p$, then evaluate $b(v)$ as $p$.
  - To evaluate $\text{if } u \text{ then } a_{\text{new}} \text{ else } a_{\text{new}} \text{ evaluate } a_{\text{new}}$, as $p$.
  - To evaluate $v.f$ when $v\text{ new } c(v_1, \ldots, v_n)$ and $f$ is the $i$th field of $c$, terminate with $v_i$.
  - To evaluate $v.l(u_1, \ldots, u_n)$ when $v\text{ new } c(v_1, \ldots, v_n)$ and $l$ in $c$ has signature $B(A_1, A_2, \ldots, A_3)$, and body $b(these(x_1, \ldots, x_n))$, evaluate $b(v, u_1, \ldots, u_n)$ as $p$.
  - To evaluate $\text{new class}(w, p)$, evaluate the method call $\text{new class}(w)(p)$ at $\text{owner}(w)$.

A Formal Semantics

- We map type $B$ to spi message type $\langle B \rangle$.
- We map value $v$ to spi message $\langle v \rangle$.
- We map method $c$ to spi message $\langle c \rangle$.
- We map body $b$ running as $p$ to spi process $\langle b \rangle$ where $k$ is a continuation channel.
- We represent SOAP envelopes as spi messages.
- We represent security guarantees by embedding begin- and end-assertions.
- These security guarantees (that is, robust safety) follow as a corollary of type preservation.

**Theorem (Type Preservation)**

If $E : B : B$ then $E \text{ new } c(\text{Ch}(B)) = \langle b \rangle : \langle k \rangle$.  

Summary of Part I

- Coding experiments in 2002, pre-WS-Security, showed we could implement an abstraction of SOAP-level security.
- Formal model shows Cryptytc can verify basic design
  - A novel approach to verifying secure RPC
  - Exposes some limitations of the type theory
  - No good model of compromised insiders
  - A criticism of this abstract approach to protocol verification is that it’s not clear which details are safe to omit.
  - Hence, we are developing a version of the pi calculus that directly embeds XML messages with crypto.

Part II: XML with Symbolic Crypto

With K. Bhargavan and C. Fournet

We develop a symbolic Prolog-like notation for XML and predicates on XML.

For example, we can write predicates defining envelopes, username tokens, and signatures.

Hence, we will subsequently be able to specify security protocols that use such tokens.
Our XML Model Part 1

- Represents valid, parsed XML
- Sorts string, att, atts, item, items, plus some others
- Adapted from Siméon and Wadler’s model (POPL’03)
- Resembles the W3C Infoset recommendation

Logical Predicates

- A Horn logic over our many-sorted algebra
  - primitive formulas for equality and list membership, but no recursively-defined predicates
  - Given certain implementability constraints, logic programs may be compiled into Abadi and Fournet’s applied pi calculus
  - Much like spi, but parametric on the algebra of values

Ex I: A SOAP request

- The predicate hasBody(item, b) below means b is the body of envelope env (the wildcard _ matches anything):
  - hasBody(env, item, b, item) :- env is the body of item:
  - Next, a body requesting info on OrderId:
  - Overall:

Ex II: A SOAP Response

- Body of the response concerning OrderId owned by user u:
  - Overall:

Need for Symbolic Crypto

- To specify interesting security properties, our predicates need to talk about cryptographically significant byte arrays encoded as strings within XML, i.e., security tokens
- For example, a UsernameToken may be added to a security header to identify the origin of a message
  - Nonce contains the Base64-encoding of random number
  - Password contains the Base64-encoding of a secure hash of the shared password, the nonce, and the timestamp
  - To do this, we add new sort bytes, and extend string

Our XML Model Part 2
Inverses, Equations

Following Dolev and Yao, model includes no inverses for cryptographic "one-way" functions
- sha1, psha1, hmac-sha1
- Nor for the function modelling user/password databases
  - principal

Ex: User Token with Digest

<UsernameToken>
  <Username>adg</Username>
  <Password>Ouywn2V6ikNNtWYL29gl9R3CPBk=</Password>
  <Nonce>cGxr8w2AnBUzuhLzDYDoW==</Nonce>
  <Created>2003-02-04T16:49:45Z</Created>
</UsernameToken>

Ex: A Security Header

User tokens occur within an envelope's security header:
- <UsernameToken>
  - <Username>adg</Username>
  - <Password>Ouywn2V6ikNNtWYL29gl9R3CPBk=</Password>
  - <Nonce>cGxr8w2AnBUzuhLzDYDoW==</Nonce>
  - <Created>2003-02-04T16:49:45Z</Created>

Hence, we can stipulate that an envelope contains a user token with a password digest:

Discussion: Password Digests

- We are assuming underlying transport is unencrypted
- What does hasUserTokenDigest(e, u, pwd, n, t, b) tell us?
  - Not very much in itself:
    - That u recently possessed the password pwd
    - Assuming that t is recent, that S doesn't itself generate such tokens, and that only S and u possess pwd
    - Nothing in sha1(concat(u, concat(utf8(t), utf8(pwd)))) specific to body b, so cannot infer u meant to send b to S
  - Worse, since u and t are public, the digest invites a dictionary attack to recover the pwd (outside our model)
- Explains why, to authenticate the message, we need to add a signature of the body based on a key derived from pwd

Signing Key from User Token

- Following stipulates the key k derived from the password, timestamp, and nonce; unlike the digest, k must be secret

Formalizing Document Refs

The group bound together in a signature is given by a finite sequence of references
- URI points to an item i, typically a node in the envelope
- DigestValue is a secure hash of the item
- ref(t,i) means that r is such a reference to i

When checking a signature, we know what's to be signed; the URI attribute is an untrusted processing hint
Formalizing Signatures

<Signature>
  <SignedInfo>
    <Reference URI="#..."><DigestValue>dFGb...</DigestValue></Reference>
  </SignedInfo>
  <SignatureValue>vSB9JU/Wr8ykpAlaxCx2KdvjZcc=</SignatureValue>
  <KeyInfo>
    <SecurityTokenReference>
      <Reference URI="#.../>
    </SecurityTokenReference>
  </KeyInfo>
</Signature>

Username Signed Message

- We need one final predicate:

WS-Security Protocol 1

| Event 1 | logs | begin(A,n,orderid) |
| Message 1 | → | c_1 where hasUserSignedBody(e_1,A,A,p,n,t,orderid) and isGetOrder(b,orderid) |
| Event 1 | ∪ | logs |
| Message 2 | → | c_2 where soapGetOrderResponse(e_2,orderid,A) |

- Authentication formalized as a correspondence; authorization decision not formalized
- We describe this protocol as a process \( Q \), and take the opponent \( O \) to be any arbitrary process in parallel
- Details omitted; this is much as in Lecture 2
- **Theorem:** \( Q | O \) is safe, that is, in every run, every end-event corresponds to a preceding begin-event
- Proofs use a combination of process calculus techniques, and are compositional

Summary of Part II

- We propose a method for analysing SOAP-level protocols
  - Identify the principals taking part in the protocol
  - Describe the exchange of messages, and the events that are to be in correspondence
  - Formalize security checks using predicates in our XML model with symbolic crypto
  - Define applied pi processes to represent behaviour of principals, by interpreting predicates as processes
  - Prove robust safety
    - In our paper, we use standard process calculus techniques
    - We are also exploring automatic techniques

Part III: X509 Security Tokens

We can handle protocols based on public-key signatures, and also protocols relying on SOAP intermediaries

Our XML Model Part 3

Additions for Public-key Certificates and Signatures:

- \( pk(B,A,B,B,B,B) \)
- \( check=B,A,B,B,B,B \)
- \( rsa-sha1(B,B,B,B,B) \)
- \( rsa-sha1-sha1(B,B,B,B,B) \)
- \( rsa-sha1-sha1-sha1(B,B,B,B,B) \)
- \( rsa-sha1-sha1-sha1-sha1(B,B,B,B,B) \)
- \( x509-user(B) \)
- \( x509-cert(B) \)

Our XML Model Part 3

- Now, we can define well-formed X509 security tokens:

  X509 Signed Message

  - We add a new clause to the definition of \( \text{isSigVal} \) and define a new top-level predicate

WS-Security Protocol 2

- As before, we describe this protocol as a process \( Q \), and take the opponent \( O \) to be any arbitrary process in parallel
- \textbf{Theorem:} \( Q \mid O \) is safe, that is, in every run, every end-event corresponds to a preceding begin event.

Contributions of the Paper

- Details and theorems for examples:
  - Username/password signatures
  - X509 signatures
  - Firewall-based authentication
- Application of three standard security principles:
  - Use explicit syntax for cryptographic transforms
  - Identify explicit goals (not just secrecy...)
  - Separate verification from discovery of evidence
- Some advice: beware weak passwords, verify all headers have been signed, use short frag URLs

3: Conclusions, Futures

- Successfully bridged gap between theoretical pi threat model and XML used in WS security protocols
  - Began with abstract view of SOAP, but was advantageous to work with direct XML model
  - Put effort into real samples, eg, MS Pet Shop
  - Found attacks within threat model
  - Proved wire-level theorems about protocols
- Next step, automated analyses within our new symbolic model of XML security protocols
  - Many potential users of WS-Security
    - BPEL4WS, OGSA, ...
  - Not many “best practices” just yet
    - Lots of standard syntax, little standard semantics

Overall Summary

- Unlike other initiatives, safe bet that web services will be widely deployed
  - If Grid happens, it will be WS-based
  - If Semantic Web happens, ditto!
- Moreover, web service security engineering presents additional challenges
  - Subtlety and lack of standardized semantics for WS-Security great opportunity to exploit successful development of formal methods for crypto
End of
Part 3