Samoa: Formal Tools for Securing Web Services

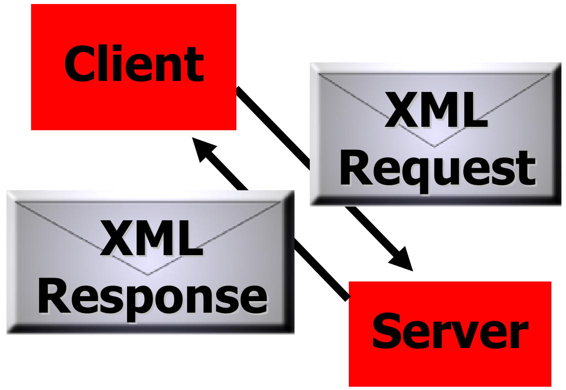
Karthik Bhargavan

Based on joint work with Andy Gordon, Ricardo Corin, Cédric Fournet, Greg O'Shea, and Riccardo Pucella

Microsoft Research

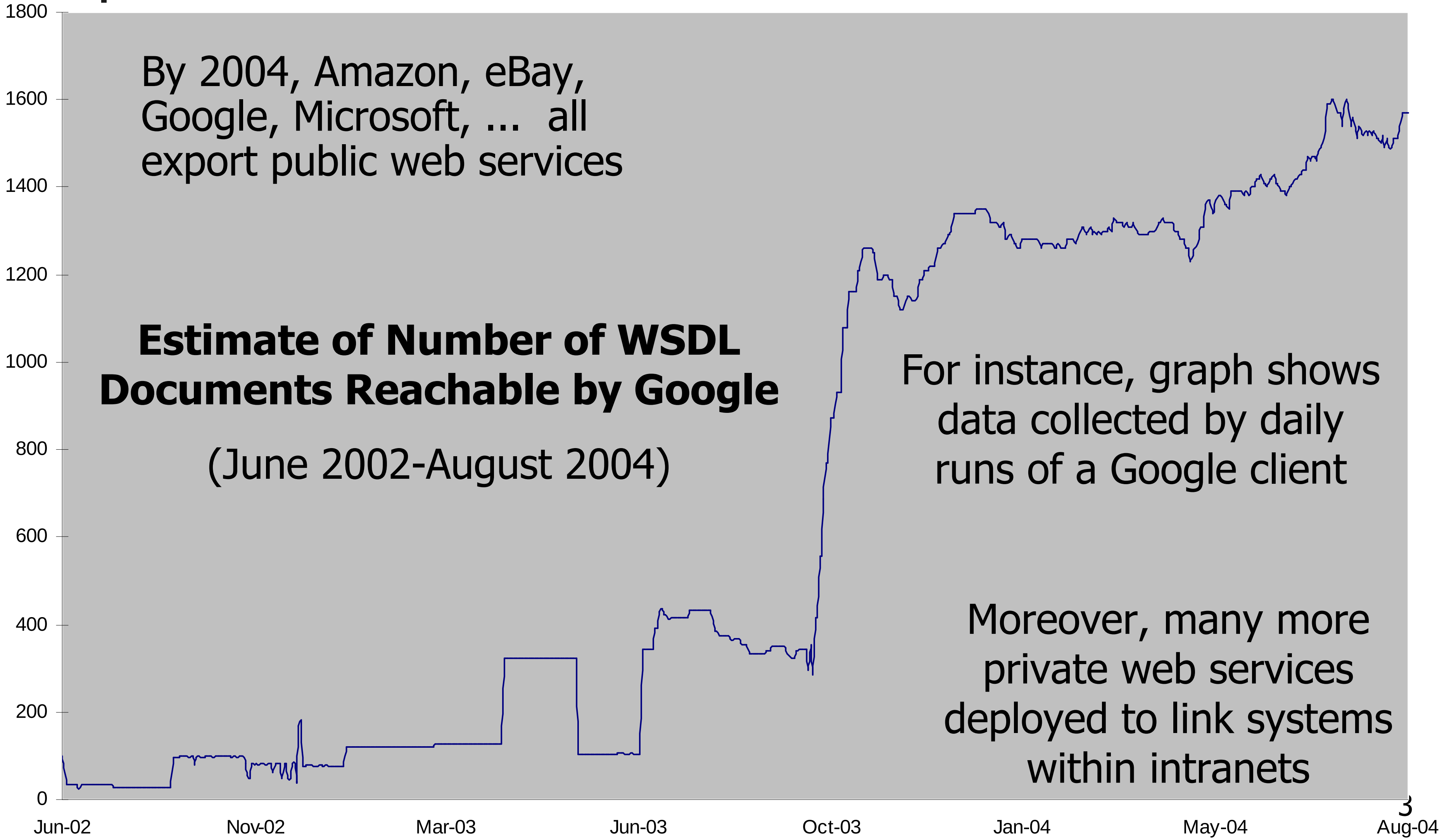
What's a Web Service?

- "A web service is a web site intended for use by computer programs instead of human beings." (Barclay et al)
- So XML not HTML



- Service messages in SOAP format:
  - Envelope/Header – addressing, security, and transactional headers
  - Envelope/Body – actual payload
- Service metadata in WSDL format:
  - For each SOAP endpoint, list of operations
  - For each operation, request and response types

Global Computing via SOAP



Securing SOAP Messages

```
<Envelope>
   <Header>
     <Security>
       <UsernameToken Id=1>
         <Username>"adg"
         <Nonce>"mTbzQM84RkFqza+lIes/xw=
         <Created>"2004-09-01T13:31:50Z"
       <Signature>
         <SignedInfo>
           <SignatureMethod Algorithm=hmac-sha
           <Reference URI=#2>
             <DigestValue>"U9sBHidIkVvKA4vZo0gGKxMhA1g=“
         <SignatureValue>"8/ohMBZ5JwzYyu+POU/v879R01s="
         <KeyInfo>
           <SecurityTokenReference>
             <Reference URI=#1 ValueType          nameToken>
   <Body Id=2>
     <StockQuoteRequest>
       <symbols>
         <Symbol>"FABRIKAM"
         <Symbol>"CONTOSO"
```

UsernameToken assumes both parties know adg’s secret password *p*

<Security> header defined by OASIS **WS-Security 2004** includes identity tokens, signatures, encrypted message parts

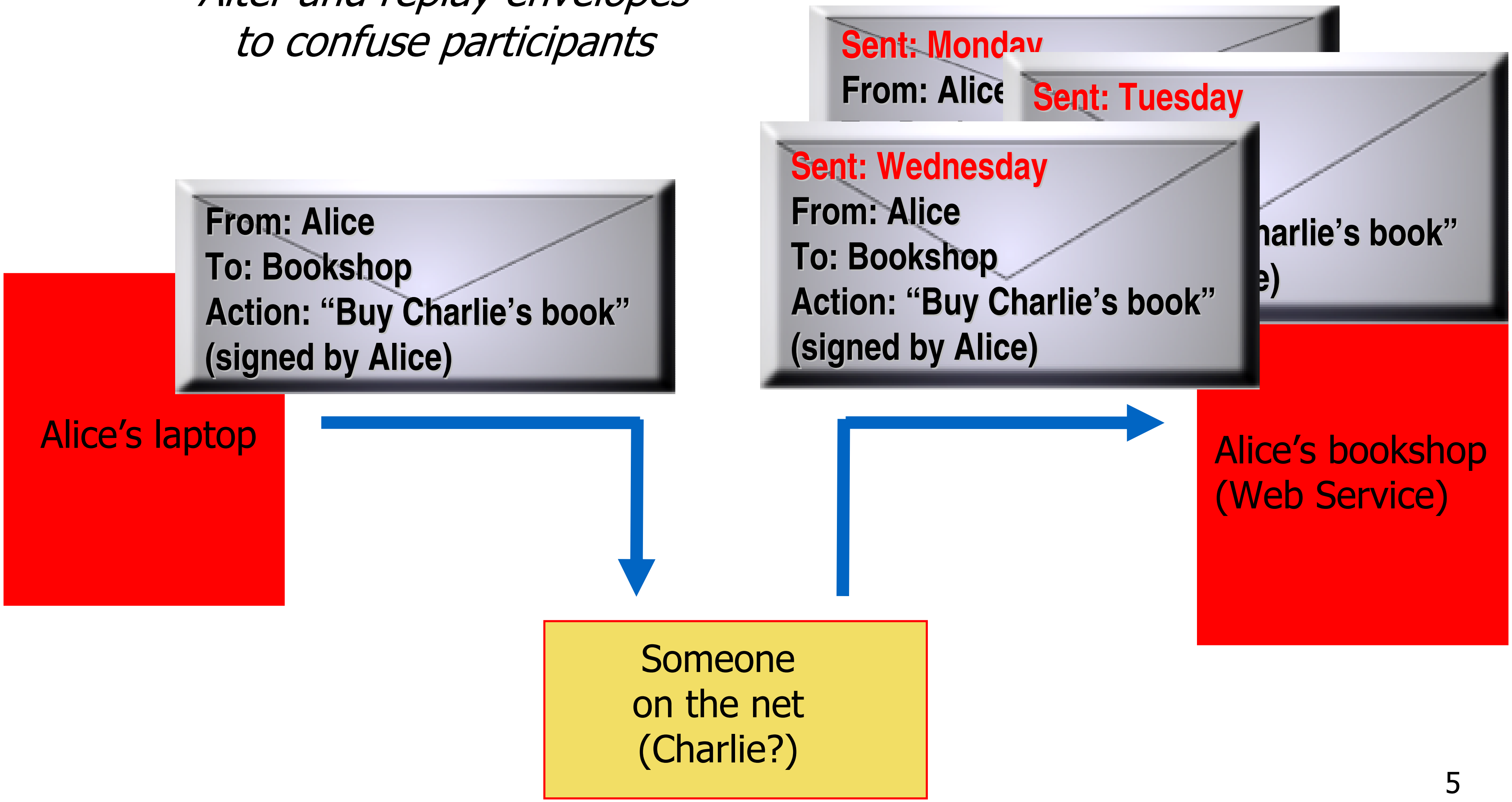
Each **DigestValue** is a cryptographic hash of the URI target

Dozens of implementations, including Microsoft Web Services Enhancements (WSE)

hmacsha1(*key*, **SignedInfo**) where *key≈psha1*(*p+nonce+created*)

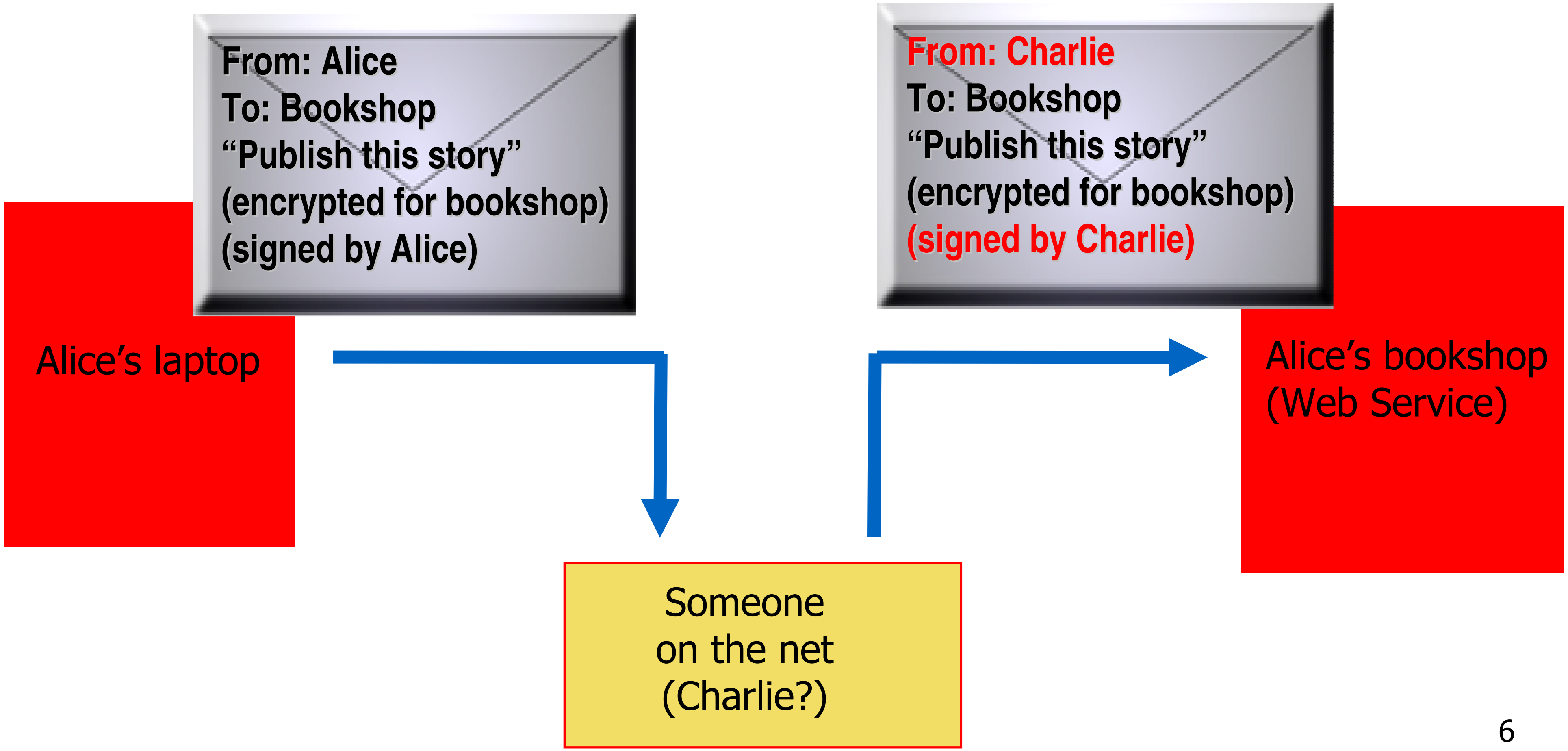
What Can Go Wrong?

Alter and replay envelopes to confuse participants

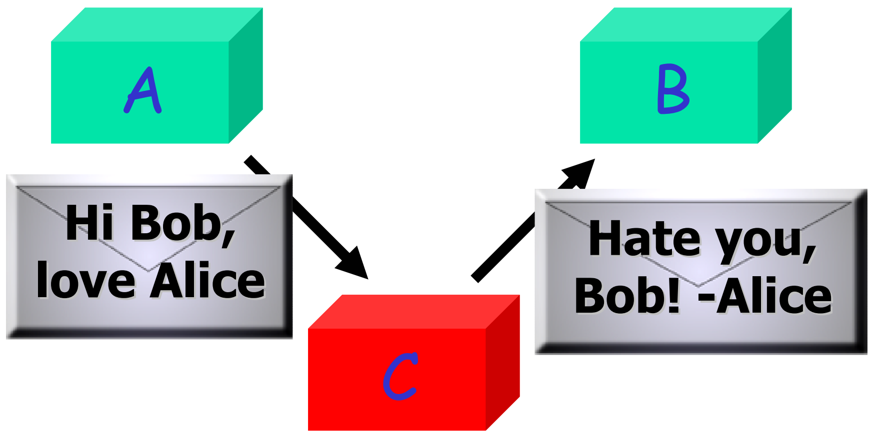


Another XML Rewriting Attack

Take credit for someone else's data...



Long History of Such Attacks



> We assume that an intruder can interpose a computer on all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. While this may seem an extreme view, it is the only safe one when designing authentication protocols.
>
> Needham and Schroeder CACM (1978)

1978: N&S propose authentication protocols for “large networks of computers”
1981: Denning and Sacco find attack found on N&S symmetric key protocol
1983: Dolev and Yao first formalize secrecy properties wrt N&S threat model, using formal algebra
1987: Burrows, Abadi, Needham invent authentication logic; neither sound nor complete, but useful
1994: Hickman (Netscape) invents SSL; holes in v2, but v3 fixes these, very widely deployed
1994: Ylonen invents SSH; holes in v1, but v2 good, very widely deployed
1995: Abadi, Anderson, Needham, et al propose various informal “robustness principles”
1995: Lowe finds insider attack on N&S asymmetric protocol; rejuvenates interest in FMs
circa 2000: Several FMs for “D&Y problem”: tradeoff between accuracy and approximation
circa 2005: Many FMs now developed; several deliver both accuracy and automation

The Pi-Calculus and Cryptography

The pi-calculus is a tiny yet highly expressive concurrent language, with precise semantics, rich theory, and several implementations

$P, Q ::=$	process
$\mathbf{out}\ x(y_1, \ldots, y_n)$	output
$\mathbf{in}\ x(z_1, \ldots, z_n);\ P$	input
$\mathbf{new}\ x;\ P$	fresh name
$P \mid Q$	parallel
$!P$	replication
$\mathbf{0}$	inactivity

- Milner, Parrow, Walker (1989); Milner (1999)
- Computation is name-passing between parallel processes on named channels. Each name has a mobile scope, that tracks the processes that can and cannot communicate on the name
- The spi-calculus (Abadi and Gordon 1997) adds Dolev-Yao style representation of cryptographic operations and protocols

So, Our Observation in 2003

Two parallel trends over past five years:

- Rapid invention and deployment of XML-based crypto protocols for securing web services (eg WS-Security)
  - Intended to scale from data centres down to devices
  - XML great help for interop
  - Security also important, but hard, XML or no XML
- Sustained and successful effort to develop formalisms and tools to check crypto protocols
  - Needham-Schroeder threat model: attacker can replay, redirect, rewrite messages, but cannot guess secrets
  - Hot Research Topic: approx 30 papers per year

Timely opportunity to develop tools for validating standards-based XML crypto protocols

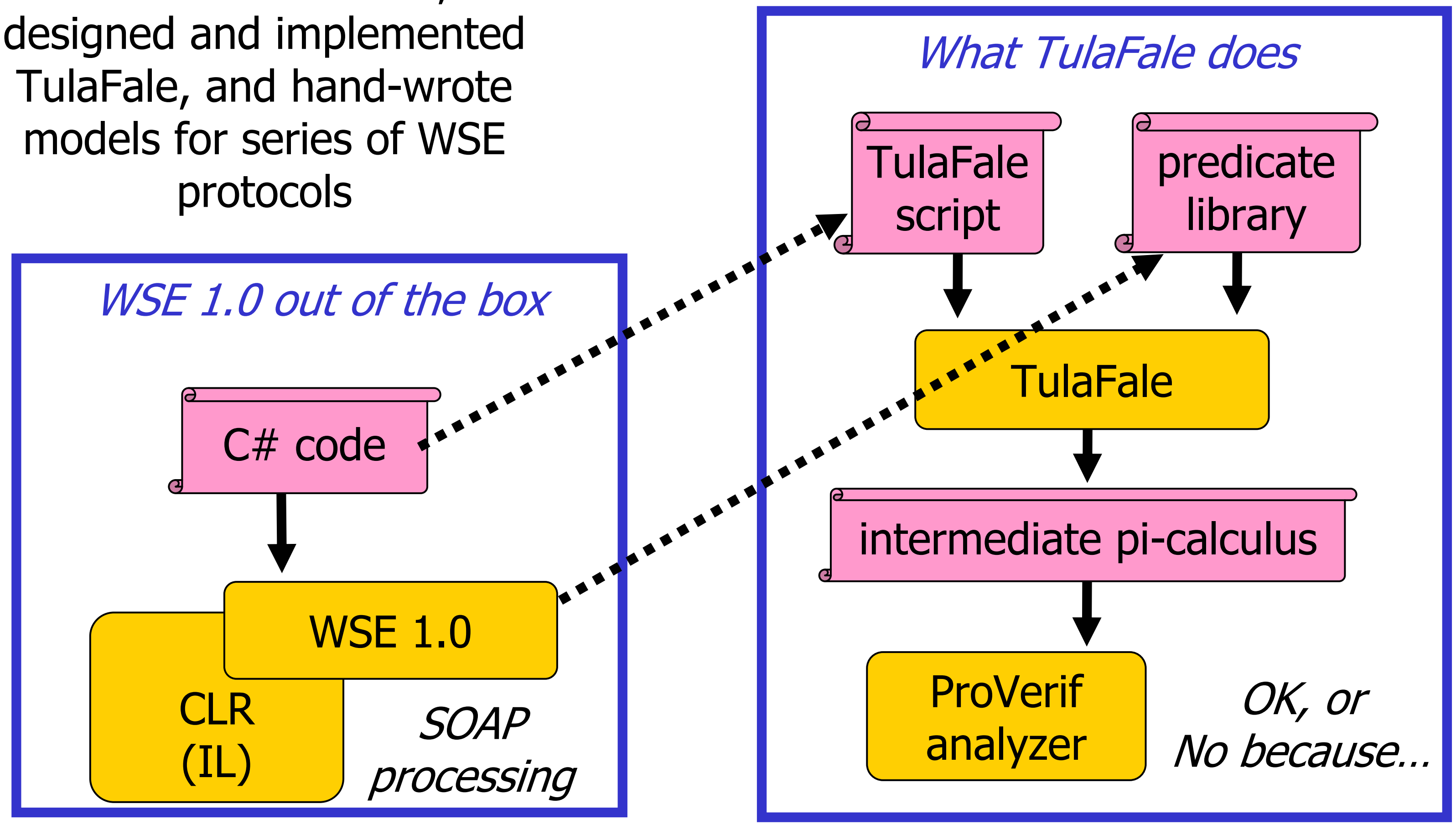
Samoa Project: Summary

- If misconfigured or mis-implemented, WS-Security protocols vulnerable to XML rewriting attacks
  - We found such attacks on code that uses MS WSE toolkit
- TulaFale tool – shows the absence of such attacks given a description of the protocol
  - First analysis tool for XML-based crypto protocols
  - Automatic analysis of hand-written models via ProVerif
- Generator and Analyzer tools – compile TulaFale scripts from declarative policy files that drive WSE2
  - We believe to be first source-based formal verification of interoperable implementations of crypto protocols
- WSE Policy Advisor – runs 30+ queries for security-related errors found in reviews of sample policies

Tool 1: TulaFale

TulaFale = pi + XML + predicates + assertions

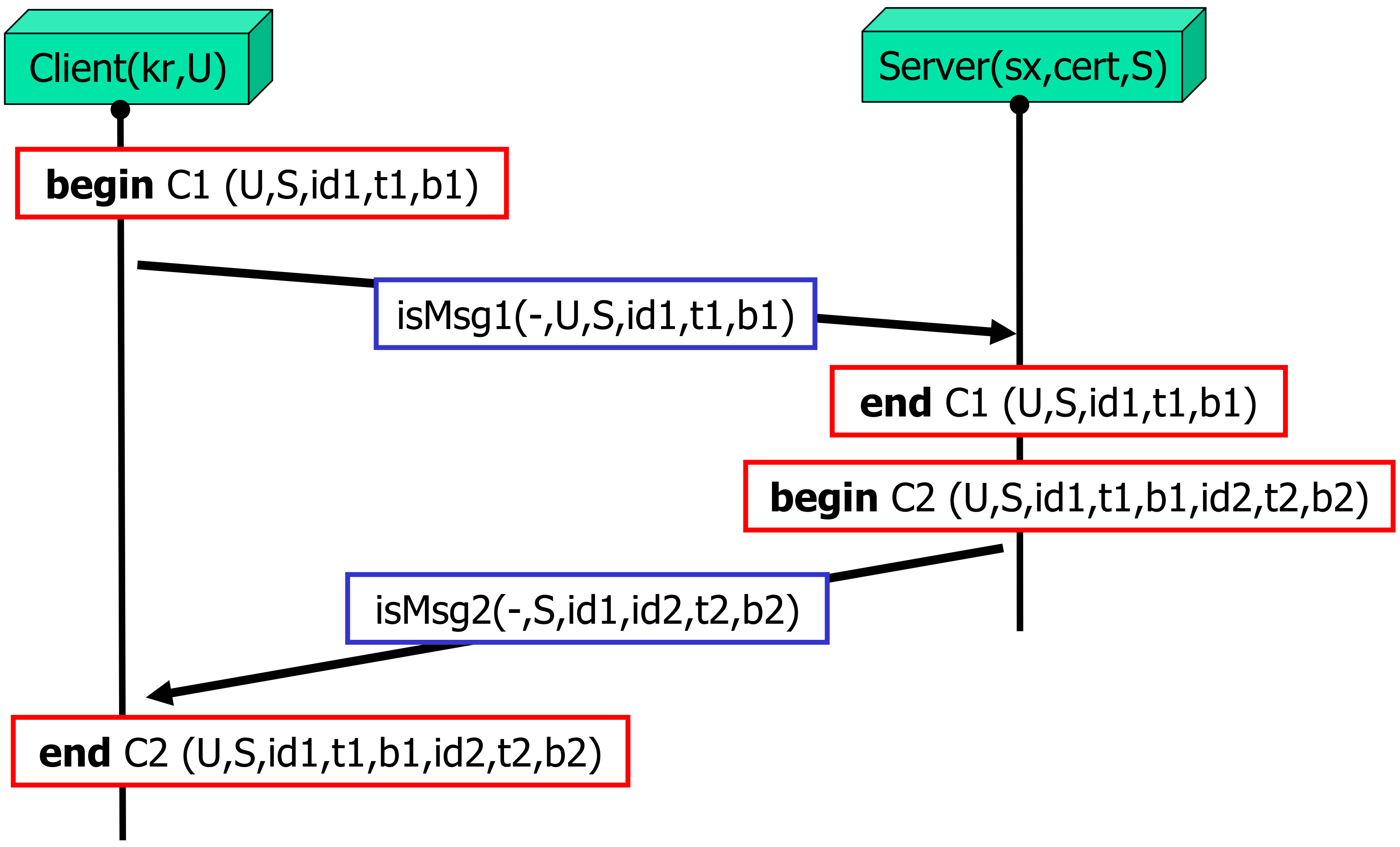
In work published at FMCO'03 and POPL'04, we designed and implemented TulaFale, and hand-wrote models for series of WSE protocols



Example: A Secure RPC

- A typical system model:
  - A single certification authority (CA) issuing X.509 public-key certificates for services, signed with the CA's private key.
  - Two servers, each equipped with a public key certified by the CA and exporting an arbitrary number of web services
  - Multiple clients, acting on behalf of human users
- Threat model: an active attacker, in control of network, but knowing none of:
  - The private key of the CA
  - The private key of any public key certified by the CA
  - The password of any user in the database
- Security goals: authentication of each message, and correlation of request and response, but not confidentiality

An intended run of the protocol



Msg 1 includes signature of S,id1,t1,b1 under key derived from username token for U

Msg 2 includes signature of id1,id2,t2,b2 under public key of S

pi+XML+predicates+assertions

```
predicate env1(msg1:item,uri:item,ac:item,id1:string,t1:string,
               eutok:item,sig1:item,b1:item) :–
  msg1 =
    <Envelope>
      <Header>
        <To>uri</>
        <Action>ac</>
        <MessageId>id1</>
        <Security>
          <Timestamp><Created>t1</></>
          eutok
          sig1</></>
      <Body>b1</></>.
```

TulaFale predicates defined by Horn clauses with message equalities

For example, this predicate used in two different modalities to construct and parse Message 1

TulaFale messages are terms in a many-sorted algebra with sorts: **item**, **items**, **att**, **atts**, **bytes**, **string**

pi+XML+predicates+assertions

```
predicate isMsg1(msg1:item,U:item,sx:bytes,cert:bytes,S:item,
                    id1:string,t1:string,b1:item) :–
  env1(msg1,uri,ac,id1,t1,eutok,sig1,b1),
  S = <Service><To>uri</> <Action>ac</> <Subject>subj</></>,
  isEncryptedData(eutok,utok,sx),
  isUserTokenKey(utok,U,n,t1,sk),
  isSignature(sig1,"hmacsha1",sk,
    <list>
      <Body>b1</>
      <To>uri</>
      <Action>ac</>
      <MessageId>id1</>
      <Created>t1</>
      eutok</>).
```

TulaFale library includes predefined predicates for XML signatures and encryption

For example, this predicate uses these predicates to check structure of Message 1

pi+XML+predicates+assertions

```
new sr:bytes; let kr = pk(sr);
new sx1:bytes; let cert1 = x509(sr,"BobsPetShop","rsasha1",pk(sx1));
new sx2:bytes; let cert2 = x509(sr,"ChasMarket","rsasha1",pk(sx2));
out publish(base64(kr));
out publish(base64(cert1));
out publish(base64(cert2));
( !MkUser(kr) |!MkService(sx1,cert1) |!MkService(sx2,cert2) |
   (!in anyUser(U); Client(kr,U)) |
   (!in anyService(sx,cert,S); Server(sx,cert,S)) )
```

The implicit attacker, running in parallel, can:

- Send and receive on the soap channel
- Generate arbitrarily many users and services
- Initiate arbitrarily many sessions

pi+XML+predicates+**assertions**

```
channel init(item,bytes,bytes,string,item).
process Client(k:bytes,U:item) =
   in init (S,certA,n,t1,b1);
   new id1:string;
   begin C1(U, <list>S id1 t1 b1</>);
   filter mkMsg1(msg1,U,S,k,certA,n,id1,t1,b1) → msg1;

   out soap(msg1);
   in soap(msg2);
   filter isMsg2(msg2,S,k,id1,id2,t2,b2) → id2,t2,b2;
   end C2(U, <list>U S id1 t1 b1 id2 t2 b2</>);
   done.
```

By sending a message on init, the attacker chooses arbitrary payloads and destination

Each **begin-event** marks the transmission of a message

Each **end-event** marks the apparently successful reception of a message

TulaFale Demo

Automatic verification of following reachability and safety properties via TulaFale/ProVerif

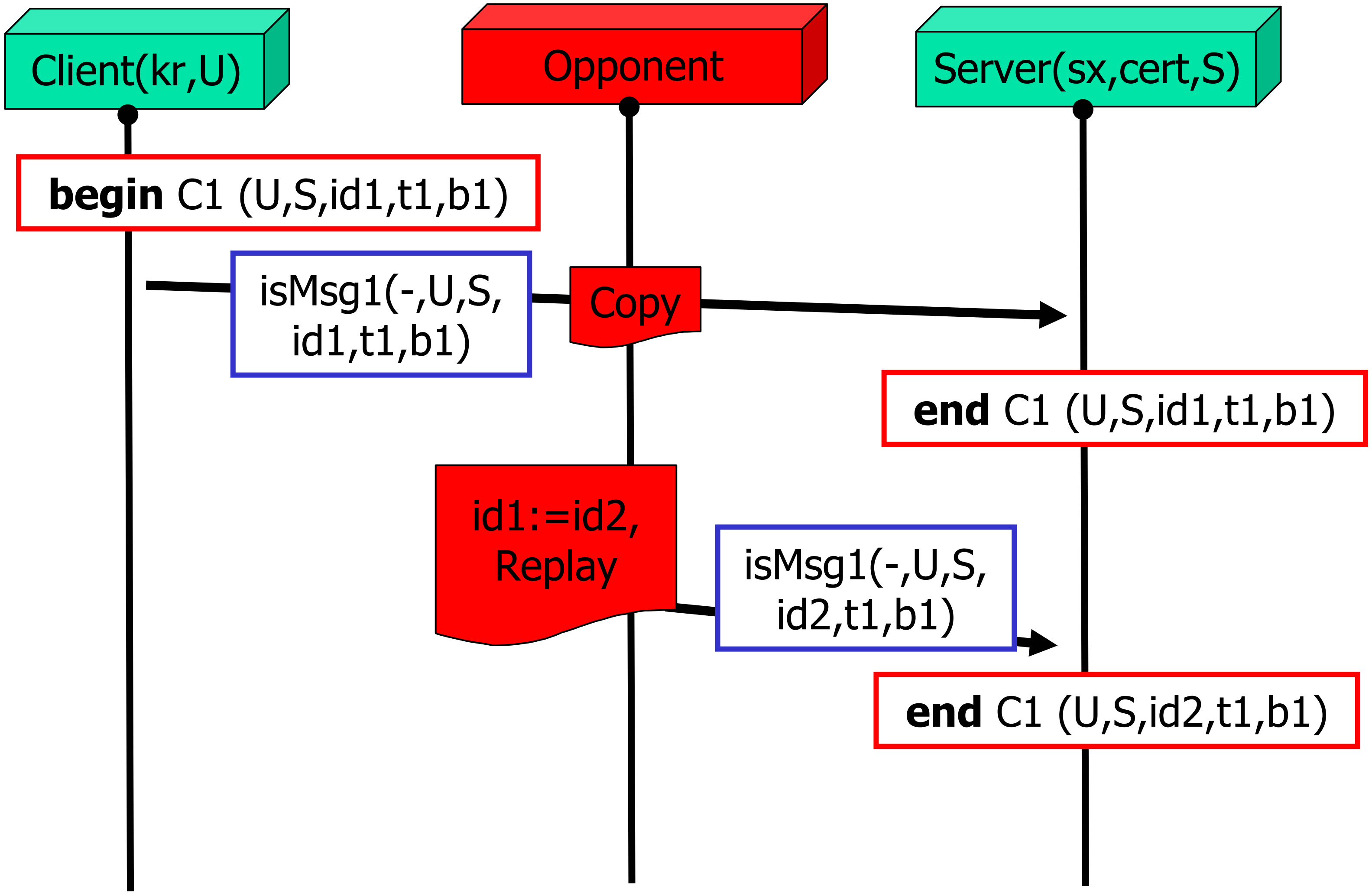
query **end**:C2(U,m12).

query **end**:C1(U,m1).

query **end**:C2(U,m12) $\Rightarrow$ **begin**:C2(U,m12).

query **end**:C1(U,m1) $\Rightarrow$ **begin**:C1(U,m1).

Suppose a client does not sign the message identifier id1...



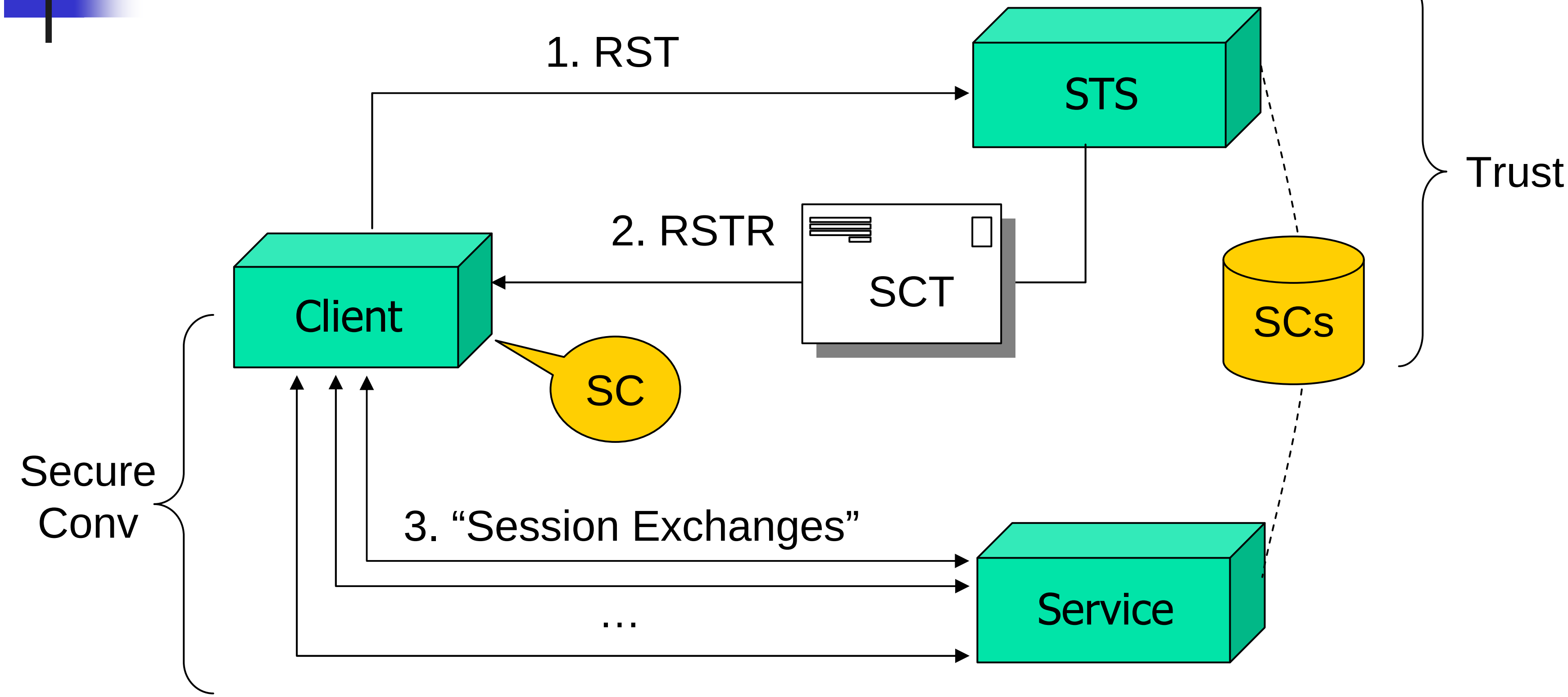
Pair (id1,t1) uniquely identifies the message only if id1 and t1 are signed

We found and fixed faults like this in preliminary WSE samples

A TulaFale Case Study

- WS-Security provides basic mechanisms to secure SOAP traffic, one message at a time
  - Signing and encryption keys derived from long-lived secrets like passwords or private keys
- If a SOAP interaction consists of multiple, related messages, WS-Security alone may be inefficient, and does not secure session integrity
  - Standard idea: establish short-lived session key
- Recent specs describe this idea at the SOAP-level
  - **WS-SecureConversation** defines *security contexts*, used to secure sessions between two parties

A Typical Scenario



STS = Security Token Server

RST = Request Security Token

RSTR = RST Response

SC = Security Context

SCT = SC Token

Discussion

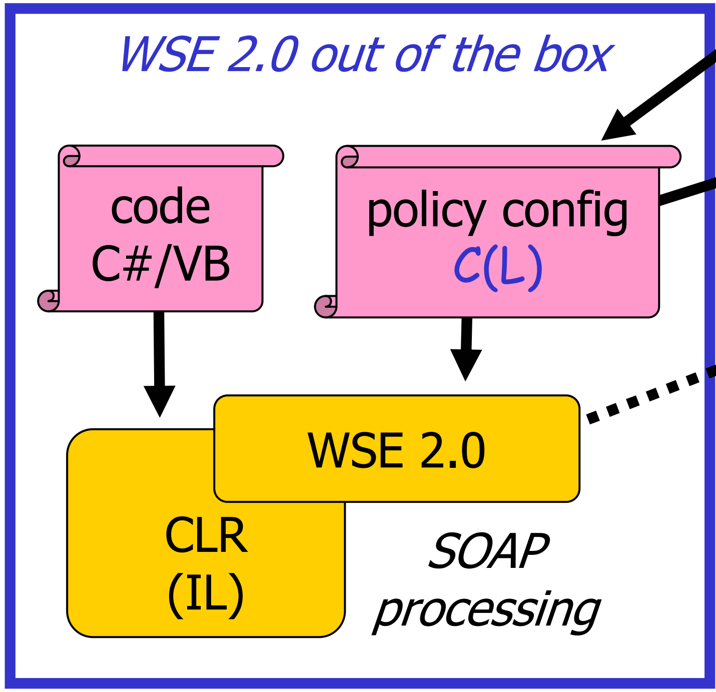
- First formal analysis of WS-Trust and WS-SecureConversation
  - XML syntax and automation very effective, against a demanding, realistic attacker model
  - Approx 1000 LOC - manual proofs we published at POPL'04 concerning one or two message protocols would not scale
  - Still, a theorem concerning open-ended sessions proved by combination of automated proof and short hand-proof
- As is common, these specs:
  - focus on message formats for interoperability
  - are non-committal regarding security, for example, no clear spec of contents of SCs
- By making modes, data, and goals explicit, we found design and implementation bugs

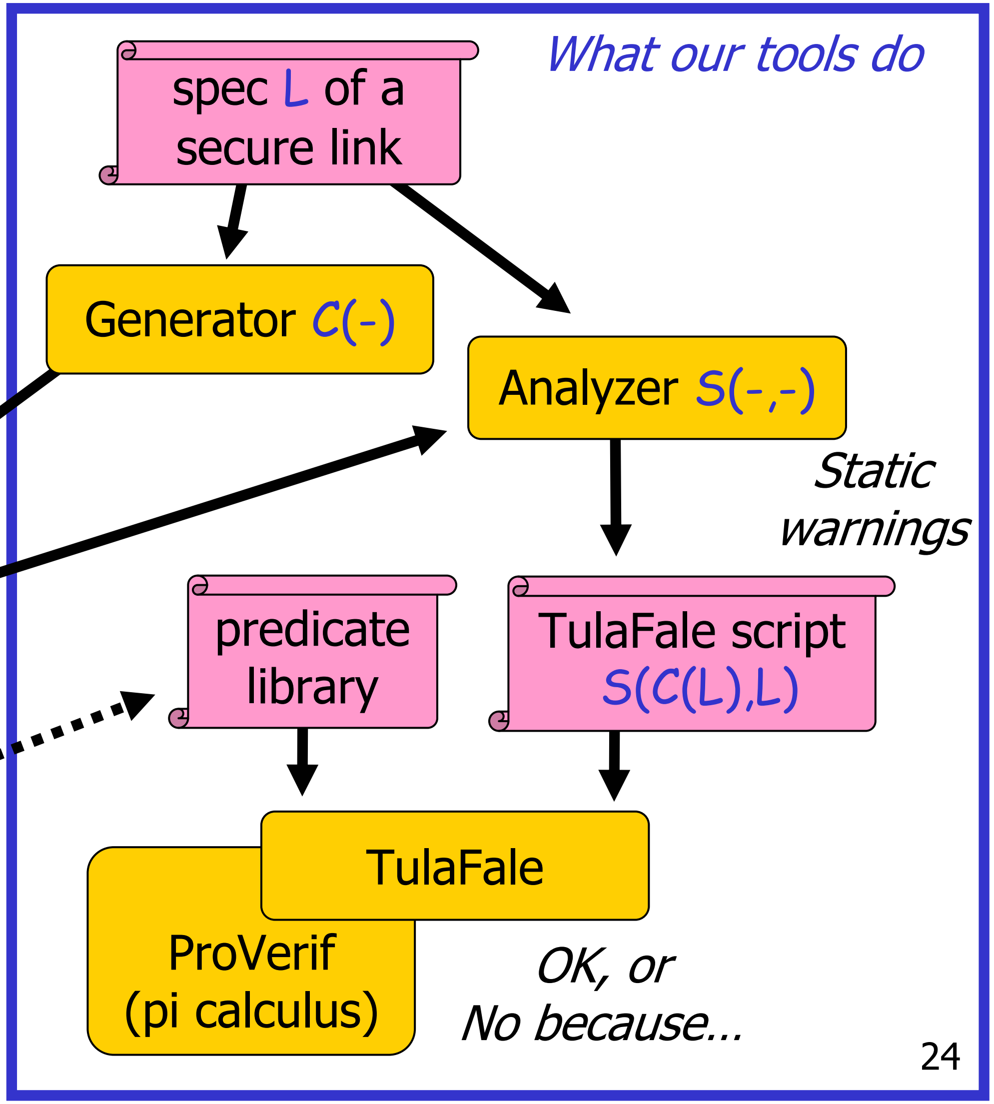
Tools for Policy-Based Security

- WSE2 security governed by declarative policies
  - Propositions on messages, separate from code
  - Stipulate integrity & confidentiality, token types
- Separation of policy and code good, but no panacea
  - Many errors found in reviews of sample policies
  - Vulnerabilities to range of passive and active attacks
- Tools offer some partial solutions
  - Generator – construct "hardened" policies – but errors creep in given "affection for copy and paste development"
  - Analyzer – prove +ve properties of deployed policies via TulaFale – good in lab, but low-level error messages limit

Tool 2: Policy Generator/Analyzer

In WSE 2.0, WS-SecurityPolicy files drive security; hence, we can generate TulaFale directly from implementation files (appears at CCS'04)





Translating Policies to Predicates

```
<Policy Id="Msg1">
   <All>
    <Confidentiality>
     <TokenInfo>
      <SecurityToken>
       <TokenType>X509v3</>
       <Claims><SubjectName>S</></>
     <MessageParts>Body()</>
    <Integrity>
     <TokenInfo>
      <SecurityToken>
       <TokenType>UsernameToken</>
       <Claims><SubjectName>U</></>
     <MessageParts>Body() Header("To")
                       Header("MessageId")</>
```

Conjunction

Encryption Requirement

Signature Requirement

```
predicate hasMsg1Policy(msg1:item,U:item,pwd:string,
                    S:item,skS:bytes,id1:string,req:item) :-
  msg1 =
   <Envelope>
    <Header>
     <To>S</>
     <MessageId>id1</>
     <Security>
      utok
      sig1</></>
    <Body>b1</></>,
  isEncryptedData(b1,req,skS),
  isUserTokenKey(utok,U,pwd,skU),
  isSignature(sig1,"hmacsha1",skU,
          [<Body>b1</> <To>S</> MessageId>id1</>]).
```

Tool 3: WSE Policy Advisor

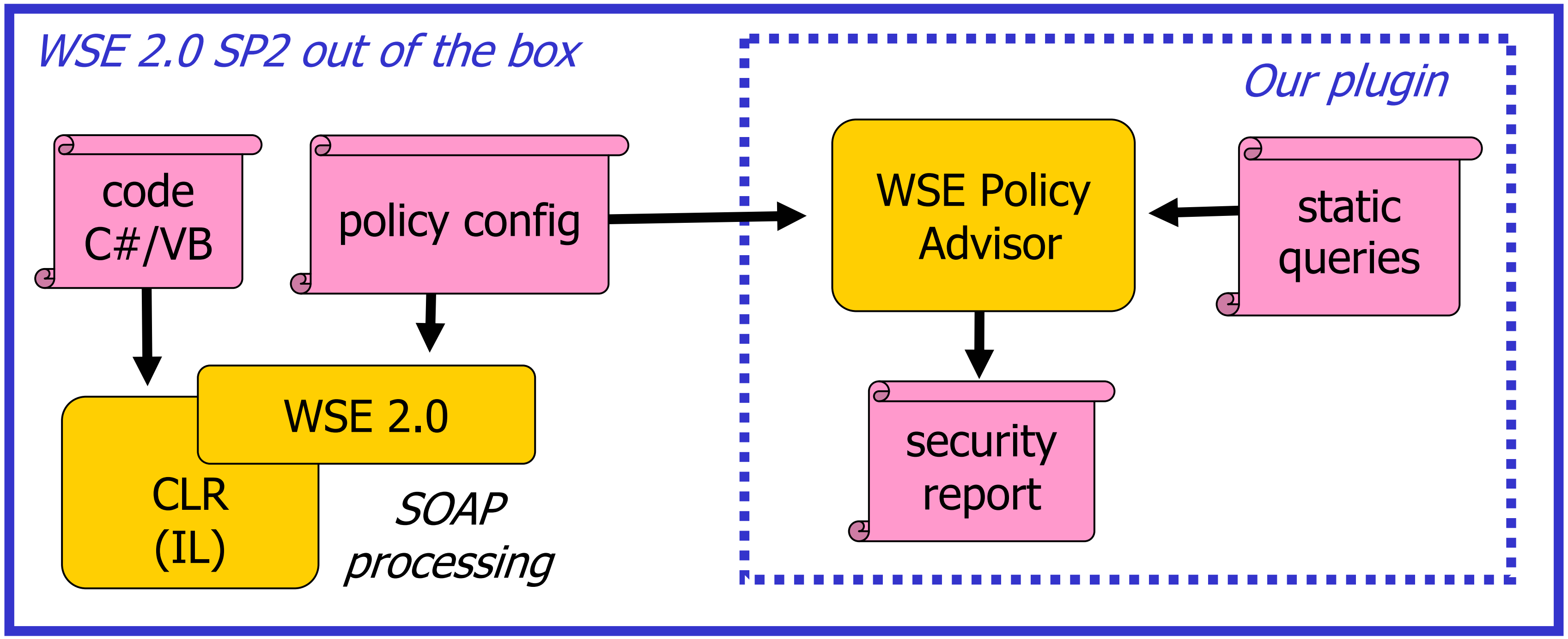
Advisor guesses intended goals and runs queries that check for:

(1) likely errors in configuration file settings

(2) conformance to conservative policy schema

(3) likely errors in (request,response,fault) mappings

(4) likely errors in particular policies



WSE Policy Advisor Demo

Related Work

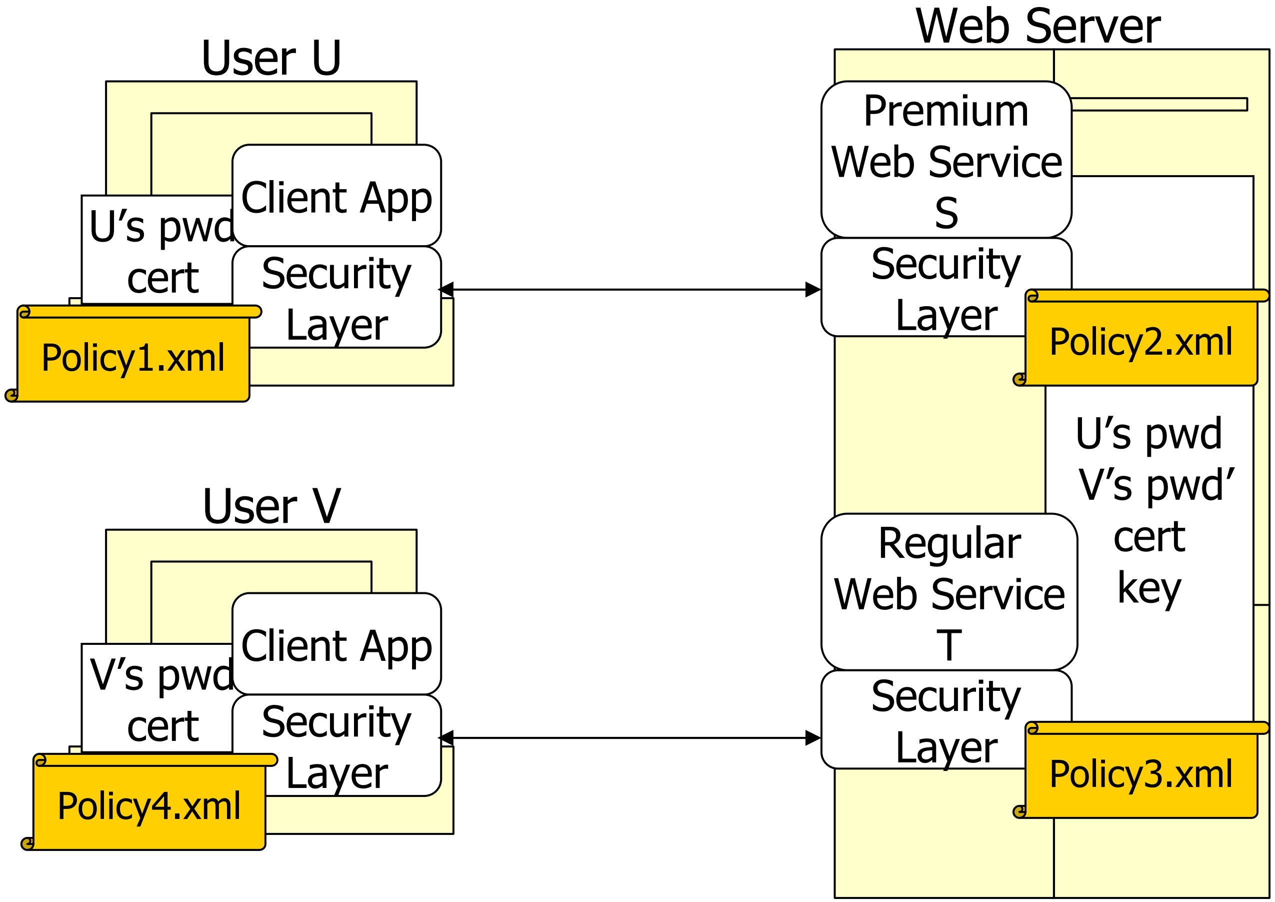
- Going in the opposite direction to our policy analyzer, several tools compile formal models to code:
  - Strand spaces: Perrig, Song, Phan (2001), Lukell et al (2003)
  - CAPSL: Muller and Millen (2001)
  - Spi calculus: Lashari (2002), Pozza, Sista, Durante (2004)
  - Apparently, the resulting code cannot yet interoperate with other implementations – an important future target
- Other Dolev-Yao modelling of web services
  - Type-based analysis of pre-WS-Security web services using Cryptyc: Gordon and Pucella (2002)
  - Model-checking of some example WS-Security specs using FDR, uncovering similar attacks: Kleiner & Roscoe (2004)
- Other formalizations of XML and web services specs
  - XPath, XSLT, XQuery: Wadler et al (since 1999)
  - WS-RM: Johnson, Langworthy, Lamport, Vogt (2004)

Summary

- Scaling from mobile devices to grid computations, SOAP-based web services are becoming an important substrate for Global Computing
- Like any websites, web services may be vulnerable to SQL injection, buffer overruns, etc; moreover, they may be vulnerable to *XML rewriting attacks*
- TulaFale, a dialect of the pi-calculus, forms the basis of a set of tools to detect and prevent such attacks
  - Analysis of specs helps uncover problems during the standardization process
  - Policy generator, analyzer, and advisor tools help secure a particular web services installation

End of Talk

Analyzing Policy Configurations



Automated tools for collecting, parsing policies from IIS Servers, Clients

Config = [Policy1, Policy2, Policy3, Policy4]

Link Specifications

- Link: Security spec for a single web service
- Spec = [Link1, Link2]
- Link1 =
  - {ServiceURI = "http://server/servicePremium",
  - ClientPrins = [U],
  - ServicePrin = S,
  - SecrecyLevel = Encrypted}
- Link2 =
  - {ServiceURI = "http://server/serviceRegular",
  - ClientPrins = [U, V],
  - ServicePrin = S,
  - SecrecyLevel = Clear}
- Links translate to security goals in TulaFale
  - All requests and responses on Link1 and Link2 must be secure

Web Location of Service

Allowed Users

Service Cert Subject Name

Request/Response Secrecy

Secrecy not required