A language for specifying cryptographic protocols [Abadi, Fournet '00]
- Builds on pi calculus [Milner '92], spi calculus [Abadi, Gordon '99]
- Protocol roles encoded as concurrent processes that communicate by sending messages over public and private channels
  - Names represent channels, keys, nonces
- Equational theories on messages specify cryptographic functions
  - Term rewriting systems are a way of specifying equational theories
  - Convergent term rewriting systems provide algorithms for equality
- Structural congruence defines behaviorally identical processes
- Internal reduction enables interaction between protocol processes
- Labeled reduction enables interaction with the attacker environment
Recap: Term Syntax

- $\mathcal{N}$: an infinite (countable) set of names $a, b, c, \ldots$
- $\mathcal{X}$: an infinite (countable) set of variables $x, y, z, \ldots$
- $\mathcal{F}$: a finite signature of function symbols $f, g, h, \ldots$
  - Includes constructors and destructors $\mathcal{F} = \mathcal{F}_C \cup \mathcal{F}_D$
- Terms represent messages that may be sent between processes
  - $M, N, O, \ldots ::=$ Terms
  - $a$ name
  - $x$ variable
  - $f(M_1, \ldots, M_n)$ function application
- $T(\Sigma)$: terms constructed from the symbols in $\Sigma$
  - $\Sigma$ contains names, variables, and functions
Recap: Process Syntax

- **Processes**: $P, Q, R, \ldots$
  
  $P, Q, R ::=\quad$ Processes
  
  0 \quad$null process
  
  new $a.P \quad fresh name generation$
  
  in$(c, x).P \quad message input (continue as P)$
  
  out$(c, M).P \quad message output (continue as P)$
  
  if $M = N$ then $P$ else $Q \quad conditional$
  
  $P \parallel Q \quad parallel composition$
  
  !P \quad replication

- **Extended Processes**: $A, B, C, \ldots$
  
  $A, B, C ::=\quad$ Extended processes
  
  $P \quad process$
  
  new $a.A \quad fresh name generation$
  
  $A \parallel B \quad parallel composition$
  
  $\{M/x\} \quad active substitution$
Recap: Security Goals

- **Notation:**
  - Structural congruence $P \equiv Q$, $A \equiv B$
  - Internal reduction $P \xrightarrow{\tau} Q$, transitive closure $P \xrightarrow{\tau^*} Q$
  - Labeled reduction $A \xrightarrow{l} B$, transitive closure $A \xrightarrow{\bar{l}} B$

- **Frames** $\phi$ represent attacker knowledge
  - $\phi = \textbf{new} \ \vec{a}.\sigma$ where $\sigma$ is a substitution of the form:
    
    $\sigma = \{M_1/x_1, M_2/x_2, \ldots, M_n/x_n\}$

  - $\phi(A)$: the frame of $A$, replace every plain process $P$ in $A$ by 0
  - Example: $\phi(\textbf{new} \ a.(P \parallel \{M/x\} \parallel Q)) \equiv \textbf{new} \ a.\{M/x\}$

- **Deduction** $\phi \vdash M$ enables an attacker to derive terms from a frame

- **Secrecy goals** in terms of what the attacker can deduce
  - $a$ is secret in $A$ if for all $A \xrightarrow{\bar{l}} B$, $\phi(B) \not\vdash a$
  - A syntactic notion of secrecy, weaker than indistinguishability and computational secrecy.
Recap: Finding Invariants

- A method to prove syntactic secrecy
- Find an invariant on the shape of the extended process as it evolves
  - For all $B$ such that $A \xrightarrow{\bar{t}} B$, $B$ must have the shape $I$
  - Example: $B \equiv \textbf{new } \bar{a}.\{\text{senc}(M, a_i)/x\} \parallel P$
    for some $M, i, P$ such that $M, P$ do not mention $a_i$
- Show that the frame of this invariant establishes your secrecy goal
  - The attacker cannot deduce a secret from the frame of $I$
  - $\phi(I) = \textbf{new } \bar{a}.\{\text{senc}(M, a_i)/x\}$
    The attacker cannot deduce the secrets $a_k$ or $M$ from this frame
Today

- Case study: Web Services Security
  - Introduction and motivation
  - Writing a WS-Security protocol in the applied pi calculus
  - Proving its security by demonstrating invariants
- Towards automated proof of protocol models
  - Deducibility constraint systems for bounded protocol sessions
  - Decision procedure for secrecy
  - Undecidability result for unbounded protocol sessions
    (Next lecture and later in the course: tackling undecidability)
APIs for Cloud-based Web Services APIs

Web Apps on AWS
Discover the benefits of building your web app on the AWS Cloud.
Watch the video »

Get Started for Free » Launch virtual machines and apps in minutes.
To secure the request, we add the BinarySecurityToken element.

The secure version of the request begins with the following:
Web Services Security Specifications and Implementations

• A series of new specifications seeks to standardize cryptographic mechanisms for use with web services
  - WS-Security: message signatures, encryption
  - WS-Trust: token issuance, key establishment
  - WS-SecureConversation: secure sessions
  - WS-SecurityPolicy: specifying policies

• Web services developers can combine these mechanisms to build a custom security protocol
  - in Java: Apache WSS4J, IBM Websphere
  - in C#: Microsoft Windows Communication Foundation Web Services Enhancements
WS-Security

- SOAP Envelope/Header/Security header includes:
  - Timestamp
    - To help prevent replay attacks
  - Tokens identifying principals and keys
    - Username token: name and password
    - X509: name and public-key
    - Others including Kerberos tickets, and session keys
  - Signatures
    - Syntax given by XML-DSIG standard
    - Bind together list of message elements, with key derived from a security token
  - Encrypted Keys
    - Syntax given by XML-ENC standard

- Various message elements may be encrypted
Password-based Authentication

\[
C \to S: \text{account}_C, \text{HMAC-SHA1}(pwd_C, \text{account}_C) \\
S \to C: \text{balance}_C
\]

- Assume \(C\) has a username “C” & password \(pwd_C\) at \(S\)
- Request Authentication
  At \(S\), only accepts an \(\text{account}_C\) after checking \(pwd_C\)
- \(C\) MACs \(\text{account}_C\) using the shared password
- \(S\) checks the MAC on \(symbol\) before responding

\(\text{HMAC-SHA1} = \text{Keyed Hash, Message Authentication Code}\)
Example 1: Password-based Auth

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

Each DigestValue is the sha1 hash of the URI target

$hmacsha1(key, \text{SignedInfo})$ where $key \approx psha1(p+\text{nonce}+\text{created})$
Example 2: Signing Multiple Elements

To prevent redirections, need to sign To and Action

To prevent replays, need to sign Timestamp and MessageId

Actually, to prevent various XML rewriting attacks, it’s necessary to co-sign other message parts with the body.
Example 3: X.509 Mutual Auth

\[ C \rightarrow S : TS | \]
\[ \text{RSA-SHA1}\{sk_C\}[symbol | TS] | \]
\[ \text{RSA-Encrypt}\{pk_S\}[symkey_1] | \]
\[ \text{AES-Encrypt}\{symkey_1\}[symbol] \]
\[ S \rightarrow C : \text{RSA-SHA1}\{sk_S\}[quote | \text{RSA-SHA1}\{sk_C\}[symbol | TS]] | \]
\[ \text{RSA-Encrypt}\{pk_C\}[symkey_2] | \]
\[ \text{AES-Encrypt}\{symkey_2\}[quote] \]

- Assume C and S have key-pairs: \((sk_C, pk_C)\) and \((sk_S, pk_S)\)
  (Assume they have exchanged X.509 public-key certificates)
- Secrecy of messages
  - freshly encrypted under \(pk_S\) and \(pk_C\)
- Request and Response Authentication
- Request-Response Correlation
  - signature of request counter-signed in response
Example 3: Request Message
(symbolic)

<Envelope>
  <Header>
  <Security>
  ts1 = <Timestamp Id='Timestamp'>
       <Created>Now1</Created>
       <Expires>PlusOneMinute</Expires>
  <BinarySecurityToken EncodingType='Base64Binary' Value_Type='X509v3'
       Id='X509Token-client.com'>
      X509(Root,C,sha1RSA,pk_C)</BinarySecurityToken>
  <EncryptedKey Id='Encrkey'>
   <EncryptionMethod Algorithm='rsa-1_5' />
   <KeyInfo><SecurityTokenReference>...</KeyInfo>
  <CipherData>
   <CipherValue>RSA-Encrypt{pk_s}[key]</CipherValue>
  <ReferenceList>
   <DataReference URI='guid6' /></ReferenceList>
  <Signature>...</Signature>
  </Body Id='Body'>
  <EncryptedData Id='guid6' Type='Content'>
   <EncryptionMethod Algorithm='aes128-cbc' />
  <CipherData>
   <CipherValue>AES-Encrypt{key}[req = <Symbol>"MSFT"/></CipherValue>
  </CipherData>
</Body>
</Security>
</Header>
</Envelope>
Attacks on WS-Security Protocols
Attacks on SOAP security

- Web services vulnerable to same sorts of attacks as websites
  - Buffer overruns, denial of service, SQL injection, etc
- New concerns: flexible, XML-based protocols
  - Web services developers can design and deploy their own application-specific security protocols
  - XML message format open to rewriting attacks
    - Much like classic active attackers (Needham-Schroeder’78)
    - Attacker can redirect, replay, modify, impersonate
    - New: message processing is driven by a flexible, semi-structured message format
- This flexibility is bad news for security
  - We found a range of problems in specs & code, thus motivating research on theory and tools
An attack that uses the XML format

A Signed SOAP Message Before...

<Message to bank’s web service says: “Transfer $1000 to Bob, signed Alice”

Bank can verify the signature has been computed using key derived from Alice’s secret password
and After an XML Rewriting Attack

Charlie has intercepted and rewritten this message

The indirect signature of the body, now hidden in **BogusHeader**, may still appear valid

Although Alice’s password has not been broken, the message now reads “Transfer $5000 to Charlie, signed Alice”
Why does the attack work?

- The SOAP message format is flexible, with optional headers
- A valid XML-Signature is not necessarily a secure WS-Security message signature
  - More checks are needed in the WS-Security implementation

- Implementing standards is tricky
  - An implementation must be willing to accept messages it will never send, for interoperability
  - It must implement a range of algorithms, one of which is dynamically chosen based on the incoming message
  - It must carefully correlate checks in different modules
Unsigned Message Timestamps

Alter and replay envelopes to confuse participants

Sent: Monday
From: Alice
To: Bank
Action: "Pay Charlie $20"
(signed by Alice)

Sent: Tuesday
From: Alice
To: Bank
Action: "Buy Charlie’s book"
(signed by Alice)

Sent: Wednesday
From: Alice
To: Bookshop
Action: "Buy Charlie’s book"
(signed by Alice)

Alice’s laptop

Alice’s bookshop (Web Service)

Someone on the net (Charlie?)
Encrypt or Sign First?

Should the client sign before encrypting or encrypt before signing? Both are allowed by the specifications. Both can be incorrect depending on the rest of the protocol.

Take credit for someone else’s data

From: Alice
To: Bookshop
“Publish this paper” (encrypted for bookshop) (signed by Alice)

From: Charlie
To: Bookshop
“Publish this paper” (encrypted for bookshop) (signed by Charlie)

Alice’s laptop

Someone on the net (Charlie?)

Alice’s bookshop (Web Service)
A Password Decryption Attack

Alice’s laptop

Someone on the net (Charlie?)

Alice’s bookshop (Web Service)
- **A semantics for web services authentication**, K Bhargavan, C Fournet, AD Gordon, Theoretical Computer Science, 2005. (POPL’04)
- **Verifying policy-based security for web services**, K Bhargavan, C Fournet, AD Gordon, TOPLAS, 2008. (CCS’04)
Simple one-message authentication protocol:

\[ A \rightarrow B : \langle b, \text{hmac}(b)_{\text{pwd}} \rangle \]
Simple one-message authentication protocol:

$$A \rightarrow B : \langle b, \text{hmac}(b)_{\text{pwd}} \rangle$$
Plan: Proving password-based signature in applied pi

- We define $\mathcal{F}$ to code XML messages with crypto
- We define a term rewriting system $\rightarrow_{\mathcal{R}}$
- We write processes for $A$ and $B$
- We specify the authentication goal using security events
- We prove the goals using invariants
Representing XML messages

- $\mathcal{F} = \{\text{pair/2, proj}_1, \text{proj}_2, \text{hmac/2, hash/1, \ldots}\}$
- Add constructors for string encodings: base64, utf8
- Add XML constructors: elem/3, att/2
  - XML tags and string constants encoded as 0-ary constructors
  - e.g. $<X \text{ id="1"}>t</X>$ is written as
    $\text{elem}(X(), \text{att}()(), \text{one}()), t)$
- Add lists: cons/2, nil/0
  - can appear both in attributes and element bodies
  - e.g. $<X a=s1 \ b=s2>e1 \ e2</X>$ is written as
    $\text{elem}(X(), \text{cons}()(), \text{cons}(()(), (a, s1), (a, s2), \text{nil}()), \text{cons}(e1, \text{cons}(e2, \text{nil}())))$
Add inverses for string encodings: ibase64, iutf8
Add XML destructors: attributes/2, body/2, attval/2
  
  take both expected tag and element/attribute as parameter
  e.g. attributes(\texttt{X()}, \texttt{elem(X(), as, b)})
  e.g. body(\texttt{X()}, \texttt{elem(X(), as, b)})
  e.g. attval(id(), att(id(), one()))

Add list destructors: hd/1, tl/1
  e.g. hd(\texttt{cons(a, b)}), tl(\texttt{cons(a, b)})

No destructors for hash or hmac
  reflects symbolic crypto assumption on one-way functions
Term rewriting system

- Rewrite rules for destructors:

  \[
  \begin{align*}
  \text{proj}_1(\text{pair}(x, y)) & \rightarrow x \\
  \text{proj}_2(\text{pair}(x, y)) & \rightarrow y \\
  \text{ibase64}(\text{base64}(x)) & \rightarrow x \\
  \text{iutf8}(\text{utf8}(x)) & \rightarrow x \\
  \text{attributes}(x, \text{elem}(x, y, z)) & \rightarrow y \\
  \text{body}(x, \text{elem}(x, y, z)) & \rightarrow z \\
  \text{attval}(x, \text{att}(x, y)) & \rightarrow y \\
  \text{hd}(\text{cons}(x, y)) & \rightarrow a \\
  \text{tl}(\text{cons}(x, y)) & \rightarrow y
  \end{align*}
  \]

- Assume all constructors and destructors are public $\mathcal{F}_{\text{pub}} = \mathcal{F}$
Processes for password-based signature

\[ A \rightarrow B : \quad T \triangleq \langle \text{Envelope} \rangle \]

\[ S I \triangleq \langle \text{SignedInfo} \rangle \]

\[ A(u, p, n, t, b) = \textbf{let} \; k = \text{hash}((p, \langle n, t \rangle)) \textbf{ in} \]

\[ \textbf{let} \; si = \text{elem}(\text{SignedInfo}(), \cdots \]

\[ \text{elem}(\text{DigestValue}(), \text{nil}, \]

\[ \text{cons}(\text{base64}(\text{hash}(b)), \text{nil}))) \textbf{ in} \]

\[ \textbf{let} \; sv = \text{hmac}(\text{utf8}(si))_k \textbf{ in} \]

\[ \textbf{let} \; ut = \text{elem}(\text{UsernameToken}(), \text{nil}, \]

\[ \text{cons}(\text{elem}(\text{Username}(), \text{nil}, \text{cons}(u, \text{nil}))), \]

\[ \text{cons}(\text{elem}(\text{Nonce}(), \text{nil}, \text{cons}(\text{base64}(n), \text{nil}))), \]

\[ \text{cons}(\text{elem}(\text{Created}(), \text{nil}, \text{cons}(t, \text{nil})), \text{nil}))) \textbf{ in} \]

\[ \textbf{let} \; sig = \text{elem}(\text{Signature}(), \text{nil}, \]

\[ \text{cons}(si, \text{cons}(\text{elem}(\text{SignatureValue}, \text{nil}, \]

\[ \text{cons}(\text{base64}(sv), \text{nil})), \ldots)))) \textbf{ in} \]

\[ \textbf{out}(net, \text{elem}(\text{Envelope}(), \text{nil}, \]

\[ \text{cons}(\text{elem}(\text{Header}(), \text{nil}, \text{cons}(ut, \text{cons}(sig, \ldots))), \]

\[ \text{cons}(b, \text{nil}))).0 \]
Processes for password-based signature

\[ A \rightarrow B : \]

\[ B(u, p) = \text{in}(net, e). \]

let \( hdrs = \text{body}(\text{Header}(\), \text{hd}(\text{body}(\text{Envelope}(\), e))) \) in

let \( b = \text{tl}(\text{body}(\text{Envelope}(\), e)) \) in

let \( si = \text{hd}(\text{body}(\text{Signature}(\), \text{hd}(hdrs))) \) in

let \( sv = \text{ibase64}(\text{body}(\text{SignatureValue}, \text{hd}(\text{tl}(\text{body}(\text{Signature}(\), \text{hd}(hdrs)))))) \) in

let \( n, t = \ldots \) in

let \( k = \text{hash}(\langle p, \langle n, t\rangle \rangle) \) in

if \( sv = \text{hmac}(\text{utf8}(si))_k \)
	hen \( \text{out}(app, b).0 \)

else 0
System, with (public) free names: net, app, u, n, t, b

\[
S = \text{new } p. (A(u, p, n, t, b) \parallel B(u, p))
\]

Authentication goal: If B sends \( x \) on the channel app then \( x \) must be the message \( b \) sent by A

Informally, only A could have sent the message since it is the only one that uses \( p \) to sign messages and nobody else knows \( p \)

Formally, we code the authentication goal as a correspondence assertion between security events [Woo, Lam '93]
Embedding Authentication Goals

- We extend the syntax of processes with events:
  \[ P, Q, R ::= \text{Processes} \]
  \[ \text{begin}(M) \quad \text{begin event} \]
  \[ \text{end}(M) \quad \text{end event} \]

- Events are inert, they have no reductions

- A begin event specifies a sender’s intention; an end event specifies a receiver’s decision.

\[
A(u, p, n, t, b) = \text{begin}(\text{Send}(u, b)) \parallel \text{let } k = \ldots
\]

\[
B(u, p) = \text{in}(\text{net}, e).
\]

\[
\ldots
\]

\[
\text{if } sv = \text{hmac(utf8(si))}_k
\]

\[
\text{then } (\text{end}(\text{Send}(u, b)) \parallel \text{out}(\text{app}, b).0)
\]

\[
\text{else } 0
\]
Embedding Authentication Goals

- **Authentication goal**: If $S \xrightarrow{\tilde{I}} B$ and $B$ has $\text{end}(\text{Send}(u, x))$ as a top-level process for some $u, x$, then $B$ also has $\text{begin}(\text{Send}(u, x))$ as a top-level process.

$$\forall \tilde{I}, B \text{ such that } S \xrightarrow{\tilde{I}} B :$$

$$B \equiv (\text{new } \bar{a}.\text{end}(M) \parallel A) \Rightarrow B \equiv (\text{new } \bar{a}.\text{begin}(M) \parallel A')$$

- This goal must be preserved as an invariant in all runs of $S$
Proof by Invariant Preservation

- **Invariant:** All extended processes obtained from $S$ can be deterministically reduced to a process of the form

  $$\text{new } p.\begin{cases} \text{begin} (\text{Send}(u, b)) \parallel \\
  \{\text{hmac}(\text{utf8}(si))_{hash(\langle p, nt \rangle)/sv} \parallel \\
  (B(u, p) + 0 + (\text{end}(\text{Send}(u, b)) \parallel \{b/x\})) \}
  \end{cases}$$

- Exercise: Show that this is indeed an invariant of $S$
  - Hint: There is a bug in the definition of $B$ which causes this invariant to fail, you need to fix the bug to make the invariant succeed
  - Hint: You will need to prove a lemma from the invariant that the only message of the form $\text{hmac}(\text{utf8}(X))_{hash(\langle p, NT \rangle)}$ that the attacker can deduce must have $X = si$, where $si$ is as computed by $A$.

- Exercise: Show that this invariant guarantees our authentication goal
What if $A$ and $B$ are replicated processes?

- The invariant must account for $N$ active sessions of the protocol, each with its own message $b_i$.
- We can make the authentication goal more precise by including $n, t$ in the events.

What if there are many users and passwords accepted by $B$, some of which may be controlled by the attacker? We refine the authentication goal: if $\text{end}(\text{Send}(u, b))$ then either $\text{begin}(\text{Send}(u, b))$ or else $u$ is compromised.

What if $b$ is encrypted? We add a secrecy goal for $b$. We enrich our invariant to account for the attacker's knowledge of $b$.

As we increase the complexity, establishing the invariant becomes more difficult and eventually infeasible.
What if $A$ and $B$ are replicated processes?
- The invariant must account for $N$ active sessions of the protocol, each with its own message $b_i$.
- We can make the authentication goal more precise by including $n, t$ in the events.

What if there are many users and passwords accepted by $B$, some of which may be controlled by the attacker?
- We refine the authentication goal: if end($Send(u, b)$) then either begin($Send(u, b)$) or else $u$ is compromised.

What if $b$ is encrypted?
- We add a secrecy goal for $b$.
- We enrich our invariant to account for the attacker's knowledge of $b$.

As we increase the complexity, establishing the invariant becomes more difficult and eventually infeasible.
What if $A$ and $B$ are replicated processes?
- The invariant must account for $N$ active sessions of the protocol, each with its own message $b$;
- We can make the authentication goal more precise by including $n, t$ in the events.

What if there are many users and passwords accepted by $B$, some of which may be controlled by the attacker?
- We refine the authentication goal: if $\text{end}(\text{Send}(u, b))$ then either $\text{begin}(\text{Send}(u, b))$ or else $u$ is compromised

What if $b$ is encrypted?
- We add a secrecy goal for $b$
- We enrich our invariant to account for the attacker’s knowledge of $b$
Proofs for more advanced protocols

- What if $A$ and $B$ are replicated processes?
  - The invariant must account for $N$ active sessions of the protocol, each with its own message $b_i$.
  - We can make the authentication goal more precise by including $n$, $t$ in the events.

- What if there are many users and passwords accepted by $B$, some of which may be controlled by the attacker?
  - We refine the authentication goal: if $\text{end}(\text{Send}(u, b))$ then either $\text{begin}(\text{Send}(u, b))$ or else $u$ is compromised.

- What if $b$ is encrypted?
  - We add a secrecy goal for $b$
  - We enrich our invariant to account for the attacker’s knowledge of $b$

- As we increase the complexity, establishing the invariant becomes more difficult and eventually infeasible.
Verifying WS-Security Protocol Implementations
A Verified WS-Security Library

Client Code

Server Code

XML-Encryption

XML-DSIG

WS-Security

SOAP

WS-Addressing

User Code

Web Services Libraries

Core Libraries

Crypto

Net

Prins

Platform (CLR/Windows)

Cryptographic algorithms (RSA, AES)

Networking Protocols (TCP, HTTP)

Principal and Credentials (X.509 certificates, passwords)
Verifying WS-Security Code

1. Replace core libraries and platform with modules implementing a symbolic Dolev-Yao Abstraction

2. fs2pv

3. Verify

"Fails security goals, here's an attack!"

"Provably satisfies security goals"

"Proverif"
Experimental results

- We coded and verified a series of protocols and libraries
  - An implementation of Otway-Rees
  - Libraries for principals + realistic attacker models
  - Libraries for Web Services Security standards
  - A series of Web Services sample protocols

- We tested interoperability with other implementations of web services protocols (WSE, WCF)
  - We can use our command-line client
    + client application code in C#
    + an IIS/WSE web server
  - We can register an IIS/F# SOAP filter for our server
    + client application code in C# using WSE
## Experimental results

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LOCs</td>
</tr>
<tr>
<td>password-based MAC</td>
<td>38</td>
</tr>
<tr>
<td>password-based MAC variant</td>
<td>75</td>
</tr>
<tr>
<td>Otway-Rees</td>
<td>148</td>
</tr>
<tr>
<td>WS password signing</td>
<td>85</td>
</tr>
<tr>
<td>WS X.509 signing</td>
<td>85</td>
</tr>
<tr>
<td>WS password-based MAC</td>
<td>85</td>
</tr>
<tr>
<td>WS request-response</td>
<td>149</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Security Goals</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>queries</td>
<td>secrecy</td>
</tr>
<tr>
<td>password-based MAC</td>
<td>4</td>
<td>weak pwd</td>
</tr>
<tr>
<td>password-based MAC variant</td>
<td>5</td>
<td>pwd</td>
</tr>
<tr>
<td>Otway-Rees</td>
<td>16</td>
<td>key</td>
</tr>
<tr>
<td>WS password signing</td>
<td>5</td>
<td>no</td>
</tr>
<tr>
<td>WS X.509 signing</td>
<td>5</td>
<td>no</td>
</tr>
<tr>
<td>WS password-based MAC</td>
<td>3</td>
<td>weak pwd</td>
</tr>
<tr>
<td>WS request-response</td>
<td>15</td>
<td>no</td>
</tr>
</tbody>
</table>
Protocol Narration for Managed Card

Initially | C has: cardId, PK(kp), PK(kRP) | IP has: kp, PK(kRP), Card(cardId, claimsuy, pwdip, Jp, kcardId) | RP has: kRP, PK(kip)

C: Request(RP, Mreq)
C: Select InfoCard(C, RP, pwdip, Jp, claim-tygP)
C: generate fresh k1, η1, η2, λk
C → IP:
  let Mek = RSAEnc(PK(kip), k1) in
  let ksig = PSHA1(k1, η1) in
  let kenc = PSHA1(k1, η2) in
  let Mrq = RST(cardId, claim-tygP, RP, λk) in
  let Muser = (U, pwdip) in
  let Mmac = HMACSHA1(ksig, (Mrq, Muser)) in
  Request Token(Mek, η1, η2,
               AESEnc(kenc, Mmac), AESEnc(kenc, Muser),
               AESEnc(kenc, Mrq))

IP: Issue Token(U, cardId, claimsuy, RP, display-tok)
IP: generate fresh η3, η4, ηse, k1
IP → C:
  let ksig = PSHA1(k1, η3) in
  let kenc = PSHA1(k2, η4) in
  let Mkrokkey = RSAEnc(PK(kRP), PSHA1(ηse, ηse)) in
  let ppdip = H1(kcardId, RP) in
  let Mrrok = Assertion(IP, Mkrokkey, claimsuy, RP, ppdip) in
  let Mtoksig = RSAEnc(k1, Mtoksig) in
  let Mek = RSAEnc(PK(kRP), k1) in
  let Mencok = (Mek, AESEnc(k1, SAML(Mtok, Mtoksig))) in
  let Mrsr = RST(Mencok, ηse) in
  let Mmac = HMACSHA1(ksig, Mrsr) in
  Token Response(η3, η4, AESEnc(kenc, Mmac), AESEnc(kenc, Mrsr))

U: ApproveToken(display-tok)
C: generate fresh k2, η5, η6, η7
C → RP:
  let Mek = RSAEnc(PK(kRP), k2) in
  let ksig = PSHA1(k2, η5) in
  let kenc = PSHA1(k2, η6) in
  let kproof = PSHA1(ηse, ηse) in
  let Mmac = HMACSHA1(ksig, Mreq) in
  let kendorse = PSHA1(kproof, η7) in
  let Mproof = HMACSHA1(kendorse, Mmac) in
  Service Request(Mek, η5, η6, η7, Mencok,
                 AESEnc(kenc, Mmac), AESEnc(kenc, Mproof),
                 AESEnc(kenc, Mreq))

RP: Accept Request(IP, claimsuy, Mreq, Mresp)
RP: generate fresh η8, η9
RP → C:
  let ksig = PSHA1(k2, η8) in
  let kenc = PSHA1(k2, η9) in
  let Mmac = HMACSHA1(ksig, Mresp) in
  Service Response(η8, AESEnc(kenc, Mmax), AESEnc(kenc, Mresp))
C: Response(Mresp)

C receives an application request
User selects card and provides password
Fresh session key, two nonces, and client entropy for token key
Encrypt session key for IP
Derive message signing key
Derive message encryption key
Token request message body
User authentication token
Message signature
Token Request, with encrypted signatures, token and body
IP issues token for U to use at RP
Fresh nonces, server entropy, token encryption key
Derive message signing key
Derive message encryption key
Compute token key from entropies, encrypt for RP
Compute PPID using card master key, RP’s identity
SAML assertion with token key, claims, and PPID
SAML assertion signed by issuer
Token encryption key, encrypted for RP
Encrypted issued token
Token response message body
Message Signature
Token Response, with encrypted signature and body
User approves token
Fresh session key, three nonces
Encrypt session key for RP
Derive message signing key
Derive message encryption key
Compute token key from entropies
Message signature
Derive a signing key from the issued token key
Endorsing signature proving possession of token key
Service Request, with issued token, encrypted signatures and body
RP accepts request and authorizes a response
Fresh nonces
Derive message signing key
Derive message encryption key
Message signature
Service Response, with encrypted signatures and body
C accepts response and sends it to application

Protocol Implemented by MSN Live Labs
Towards Automated Symbolic Security Proofs
Assume that the functions in $\mathcal{F}$ are divided into private and public functions: $\mathcal{F} = \mathcal{F}_{\text{pub}} \cup \mathcal{F}_{\text{priv}}$

The attacker can use public function symbols and fresh names to deduce terms from a frame

$\phi \vdash M$: $M$ can be deduced from $\phi$

- $\text{new } \overline{a}.\{\overline{M}/\overline{x}\} \vdash M_i$
- $\text{new } \overline{a}.\{\overline{M}/\overline{x}\} \vdash b$, if $b \in \mathcal{N} \setminus \overline{a}$
- $\text{new } \overline{a}.\{\overline{M}/\overline{x}\} \vdash f(M_1, \ldots, M_n)$
  if $\phi \vdash M_i$ for each $i \in [1..n]$
- $\text{new } \overline{a}.\{\overline{M}/\overline{x}\} \vdash N$
  if $\text{new } \overline{a}.\{\overline{M}/\overline{x}\} \vdash N$ and $M =_{\mathcal{R}} N$
- Exercise: Show how to deduce $a$ from $\text{new } a, k.\{\text{senc}(a, k)/x_1, k/x_2\}$
Decidability of attacker deduction

- Can we write an algorithm to decide whether $\phi \vdash M$?
- Exercise: Prove that $\phi \not\vdash p$ when

$$\phi = \text{new } p.\{n/y, t/z, si/w, \text{hmac(utf8(si))}_{\text{hash}}(\langle p, \langle n, t \rangle \rangle)/s\}$$
Decidability of attacker deduction

- Can we write an algorithm to decide whether $\phi \vdash M$?

- Exercise: Prove that $\phi \not\vdash p$ when

$$\phi = \textbf{new } p.\{n/ y, t/ z, si/ w, \text{hmac(utf8}(si))_{hash(\langle p, \langle n, t \rangle \rangle)}/s\}$$

- easy if $\phi$ did not mention $p$
Decidability of attacker deduction

- Can we write an algorithm to decide whether $\phi \vdash M$?
- Exercise: Prove that $\phi \not\vdash p$ when

$$\phi = \text{new } p.\{n/y, t/z, si/w, \text{hmac}(\text{utf8}(si))_{\text{hash}}(\langle p, \langle n, t \rangle \rangle)/s\}$$

- easy if $\phi$ did not mention $p$
- easy if we could show that the attacker can deduce only a fixed number of terms that contain $p$
Decidability of attacker deduction

- Can we write an algorithm to decide whether $\phi \vdash M$?

- Exercise: Prove that $\phi \not\vdash p$ when

$$\phi = \text{new } p.\{n/y, t/z, si/w, \text{hmac(utf8(si))}_{\text{hash}(<p,<n,t>>)}/s\}$$

- easy if $\phi$ did not mention $p$
- easy if we could show that the attacker can deduce only a fixed number of terms that contain $p$

- How would you automate the proof?
A decidability result (and algorithm) for attacker deduction
  - Deduction is PTIME complete

A decidability result (and algorithm scheme) for protocol security for bounded number of sessions
  - Bounded protocol security is NP complete

An undecidability result for unbounded number of sessions
  - Proof techniques that require user input or allow non-termination
Inference systems

- An inference rule is of the form

\[ M_1, \ldots, M_n \vdash M \]

- An inference system is a set of rules

- We rewrite the attacker’s deduction rules eliminating destructors:

\[
\begin{align*}
  x, y & \vdash \langle x, y \rangle & x, y & \vdash \text{hash}(x) \\
  x, y & \vdash \text{aenc}(x)_y & x, y & \vdash \text{senc}(x)_y \\
  x, y & \vdash \text{sig}\{x\}_y & x, y & \vdash \text{hmac}(x)_y \\
  \langle x, y \rangle & \vdash x & \langle x, y \rangle & \vdash y \\
  \text{aenc}(x)_y, \text{sk}(y) & \vdash x & \text{senc}(x)_y, y & \vdash x
\end{align*}
\]
Assume an inference system \( I \) and a set of terms \( T \).

A proof \( \Pi \) of \( T \vdash M \) in \( I \) is a tree such that:
- The root is labeled \( M \).
- Every leaf is a term in \( T \).
- Every node has a result (label) \( N \) and hypotheses \( N_1, \ldots, N_n \), such that \( N_1, \ldots, N_n \vdash N \) is an instance of an inference rule in \( I \).

Example: A proof that \( \langle \text{senc}(s)_\langle a, b \rangle, a \rangle, \text{senc}(b)_a \vdash s \) is:

\[
\begin{array}{c}
\langle \text{senc}(s), \langle a, b \rangle, a \rangle \\
\text{senc}(b, a) \\
\hline
\text{senc}(s, \langle a, b \rangle)
\end{array}
\]
Locality

- Inference system $I$ is *local* if whenever $T \vdash M$, there exists a proof $\Pi$ such that every conclusion in $\Pi$ is a subterm of a term in $T \cup \{M\}$.
  - Hence, in searching for a proof, one never need look outside the subterms of the known $T$ or the target $M$.
  - There is only a polynomial number of such subterms.

- *One-step deducibility*: can we test whether $T \vdash M$ in one inference step?

- Given locality and one-step deducibility, we obtain decidability in PTIME.
Proving Locality

Exercise: Prove that our attacker deduction system is local

\[
\begin{align*}
\langle x, y \rangle & \vdash x, y \vdash \langle x, y \rangle \\
x, y & \vdash \text{hash}(x) \\
x, y & \vdash \text{aenc}(x)_y \\
x, y & \vdash \text{senc}(x)_y \\
x, y & \vdash \text{sig}\{x\}_y \\
x, y & \vdash \text{hmac}(x)_y \\
\langle x, y \rangle & \vdash x \\
\langle x, y \rangle & \vdash y \\
\text{aenc}(x)_y, \text{sk}(y) & \vdash x \\
\text{senc}(x)_y, y & \vdash x
\end{align*}
\]

- Hint: only consider minimal proofs
- Hint: we will need a stronger invariant