Security Protections for Mobile Agents

Stephen R. Tate
Dept. of Computer Science and Engineering
University of North Texas

Talk describes joint work with Ke Xu and Vandana Gunupudi
Research supported by the National Science Foundation
Agent Model

class Agent {
    public void Register() {
        // Code...
    }
}

AGENTID=007 HOME=129.120.36.23

Originator

Delta Airlines

American Airlines

United Airlines

Hosts
Host Security

- Malicious agent attacks
  - Accessing non-approved resources
  - Denial of service

- Solution
  - Host has complete control over execution environment, so use a “sandbox” (approach taken by Java applets, AgentTCL, etc.)
Agent Security

- Possible attacks
  - Tampering with the agent state
    - Modifying bids from previous hosts
  - Modifying execution branches
    - Force into an “accept bid” branch
  - Examining agent logic to gain unfair advantage
    - Discover maximum acceptable bid
  - Repeated execution until favorable result
    - Restart with different bids until one is accepted
  - Denial of Service
    - Throws agent away
Results Overview

• **Integrity-only protection methods**

• **Comprehensive protection with minimal trust assumptions**

• **Provable security in the universally composable paradigm**
Data Integrity

• Easy solution: All hosts have public/private keys and certificate, and sign all data and a running hash.

• But: Can this be done either
  (a) without a PKI
  or (b) without using public key crypto
Previous Methods

- Idea 1: Create one-use secrets for a MAC that only the originator can recover.
  - Partial Result Authentication Code [Yee 1998]
    • new secret = hash(current secret)
  - Hash Chaining [Karjoth et al., 1999]
    • new secret = hash(current secret, data, next host)

- Problem: Old results can’t change
  - Set Authentication Code [Loureiro, 2001]
    • “Set Hash” of MAC values from hosts
    • Given host secret, host can insert and delete elements
    • But: Originator and each host must share a secret
Our Results

• New, more efficient method for establishing shared secrets for set authentication code
• Experimental study of all methods.
More Results – Times

- overalltimes Baseline
- overalltimes Hash Chaining
- overalltimes PRAC
- overalltimes Set Authentication Code
- overalltimes Modified Set Authentication Code

Time (seconds) vs. Number of host visits

Security Protections for Mobile Agents

Slide 9
Comprehensive Protection – Goals

- Agent state remains secret from hosts
- Possibilities for host data:
  - Limited host data privacy (LHDP)
    - Limited by the size of state
  - Verifiable host data privacy (VHDP)
    - Known/analyzable agent function
  - Complete host data privacy (CHDP)
    - Complete secrecy
- Only LHDP and VHDP possible wrt originator
- CHDP possible wrt other hosts
Prior Work on Agents with Secret Data

Our Improvements

- No third-party trust required
- Protection against collusion
- Fault tolerance – no single point of failure
- Standard multi-agent benefits (parallelism, etc.)
- Clear security model definition and protocol that is provably secure in the universally composable paradigm

Security Protections for Mobile Agents
Basic Tool: 2-party Secure Function Evaluation [Yao 1986]

• Inputs: Alice holds value w
  Bob holds value x
• Computation: Compute $f(w,x) \rightarrow (y,z)$
• Output: Alice gets y
  Bob gets z
• Security:
  – Alice learns no more about x than follows from y
  – Bob learns no more about w than follows from z
• Examples: Salary comparisons
Here’s an encrypted circuit to compute $f(w,x)$ along with the encrypted form of $w$. I’m also sending the key to decrypt your output $(z)$.

I need the encrypted form of my $x$, so let’s do an oblivious transfer so I can get this bit-by-bit.

Now I evaluate the encrypted circuit to get encrypted $(y,z)$.

Here’s the encrypted form of $y$.

I can decrypt my own output $z$. 
Here’s an encrypted circuit to compute f(w,x) along with the encrypted form of w. I’m also sending the key to decrypt your output (z).

I need the encrypted form of my x, so let’s do an oblivious transfer so I can get this bit-by-bit.

Now I evaluate the encrypted circuit to get encrypted (y,z).

Here’s the encrypted form of y.

I can decrypt my own output z.
Oblivious Transfer (OT) Step

• Option 1:
  • Host does traditional oblivious transfer protocol with originator
  • Disadvantage: violates the autonomous agent requirement that the originator may go off-line

• Option 2:
  • Use a “trusted third party” to decrypt the required inputs
  • Requirement: The originator and all parties must trust the 3rd party not to violate the basic oblivious transfer security requirements.
Illustration

Send to remote host

Circuit C and

E_T(id,1,0,k_{1,0})
E_T(id,1,1,k_{1,1})

k_{1,0}
k_{2,1}

Now host can evaluate encrypted circuit

E_T(id,1,0,k_{1,0})
E_T(id,2,1,k_{2,1})

Originator

Remote Host

Trusted Third Party (T)
Efficiency

- Conversion to boolean circuit is costly
- Conversion to encrypted circuit is costly
- Is it hopeless????
  - NO!
  - Only encrypt sensitive parts of agent function
  - Example: Encrypted circuit to track best bid secretly is about 8k bytes
Our New Solution

- Multi-agent protocol to effect oblivious transfer
- Security of host’s data is guaranteed (no trust in an external party required)
- Security of agent’s state is guaranteed (originator doesn’t need to trust an external party)

- One technicality: Agents must avoid being on the same host, or on a set of colluding hosts (although the threshold for collusion can be set arbitrarily).
- For presentation simplicity: assume two agents/hosts: \( H_a \) (agent host) and \( H_h \) (helper host)
Oblivious Threshold Decryption (OTD)
A new cryptographic primitive

- **Setting:**
  - $H_a$ holds two ciphertexts, encrypted with a common public key
  - $H_a$ and $H_h$ hold shares of the corresponding private key

- **Goal:**
  - $H_a$ learns the decrypted value of one of the ciphertexts (selected by $H_a$)
  - $H_h$ gets no information about which value $H_a$ received
Extensions

- Easily extended from one helper to multiple helper hosts
- Tool: Threshold cryptography.
  - Decryption function split into $n$ “shares” for $n$ agents
  - $t$ (threshold) or more agents can decrypt
  - $t-1$ or fewer cannot decrypt

- Since $t$ can be less than $n$, adds fault tolerance
- Since information safe from “collusions” of up to $t-1$ hosts, security is increased.
- Note: OTD interleaves OT and Threshold operations, but possible if using one-round protocols (which exist)
Universally Composable Security

• Proposed by Canetti [FOCS 2001]
• Problem to solve: Some protocols are secure when analyzed by themselves, but become insecure when combined with other protocols
  – Example: Zero-knowledge proofs
• UC model of computation includes parties, adversary, and environment
Universally Composable Security

- Adversary talks to Z, corrupts parties, controls communication
- F does all the computation

Security Goal: ∀ A ∃ S ∀ Z: IDEAL_{F,S,Z} ≈ REAL_{π,A,Z}
Important Properties of UC Security

• Composability (informal): If protocol $\rho$ is secure using ideal functionality $F$ as a sub-protocol/oracle, then $\rho$ using $\pi$ is secure.

• Malicious adversary compiler: If a protocol $\pi$ is secure against honest-but-curious adversaries, it can be “compiled” into a protocol $\sigma$ that is secure against malicious adversaries.
  – Down side: Inefficient construction and requires broadcasting all messages to all parties.
Difficulties with UC Agent Computation

- Concept of “traveling” integral to mobile agents, and treating as a function evaluation doesn’t capture this.
- Autonomous property of agents means originator should be able to go off-line and not participate further – but malicious adversary compiler requires all parties participate in all steps.
Our Solutions

• Carefully crafted “ideal functionality” reflects notion of agent moving
  – Ideal functionality “steps” agent computation
  – Agent purposely exposed to adversary between steps

• Properties of encrypted circuits used to handle malicious hosts wrt the originator’s interests, and “compiler” used for the rest
Ideal Functionality $F_{MA}$

1. Originator sends all initial agent states to $F_{MA}$
2. Hosts send private inputs to $F_{MA}$
3. $F_{MA}$ does the following:
   a) $F_{MA}$ collects values sent to it, and when it has both the agent state input to a host and that host’s private data, it computes the new state and output, sending the output back to the host.
   b) If $F_{MA}$ receives a request from both the originator and a host to reveal the agent state, send the input state for this agent back to the originator (note that otherwise all intermediate agent states never leave $F_{MA}$).
   c) When the final host is visited for any agent, send the final agent state back to the originator.
Main Results

Theorem 1: Given a UC realization of non-interactive threshold cryptography, for any threshold parameter \( m > n/2 \), our protocol securely realizes the mobile agent ideal functionality in which an honest-but-curious adversary can corrupt up to \( m-1 \) hosts.

Theorem 2: Given a UC realization of non-interactive threshold cryptography, for any threshold parameter \( m > n/2 \), our protocol in conjunction with our combined use of the malicious adversary compiler and our custom technique securely realizes the mobile agent ideal functionality in which a malicious adversary can corrupt up to \( m-1 \) hosts.
Conclusions and Open Problems

- Strong security properties achievable in a software-only setting, with various strengths of security and assurance trading off with practicality
- Security and robustness enhanced through the use of multiple agents
- Due to reliance on encrypted circuits, efficiency is an issue – is there an alternative to encrypted circuits?
- More efficient/direct building-blocks?