# Cryptographic Protection of Migratory Software

Volker Roth
CRCG Omaha, NE
Fraunhofer IGD, Germany
vroth@igd.fhg.de

6 Aug, 2003

@ *NebraskaCERT,* Omaha, August 2003

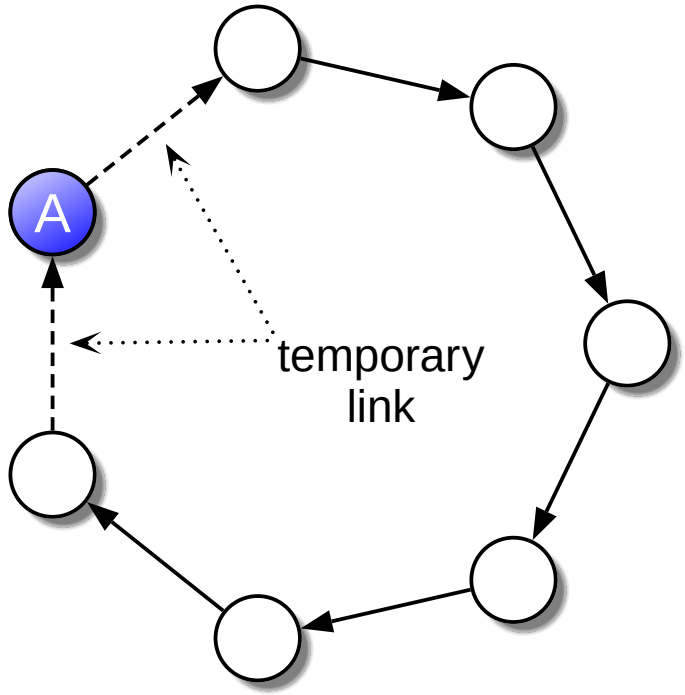
# Migratory Software Agents

Mobile agent starts (and ends) at initial (blue) node $A$

Agent migrates and operates autonomously

Upon agent's return, the owner verifies returned data

Problem: agents do not have a Trusted Computing Base



# (Some) Security Objectives

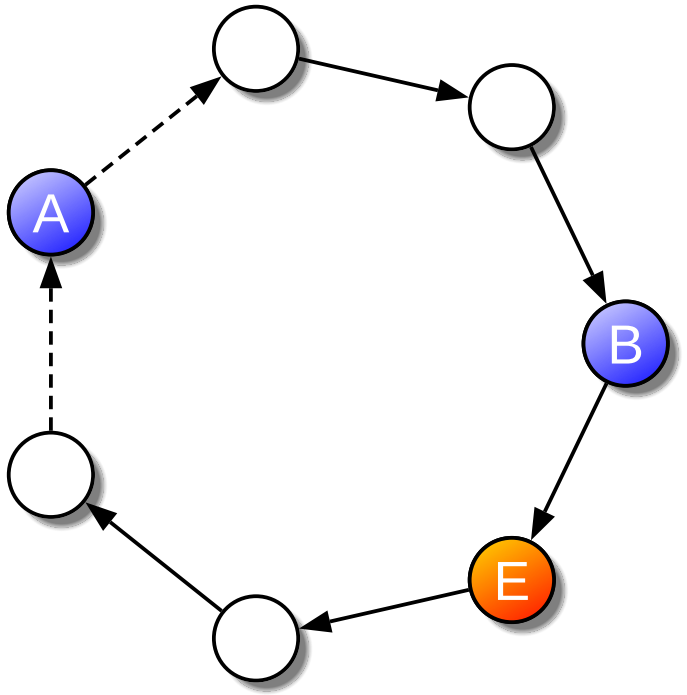
Asset: data of the agent

Adversary: malicious host

**Confidentiality:** some data is revealed *e.g.,* only on node $B$

**Integrity:** $E$ cannot tamper with data the agent collected prior to reaching node $E$

**Authenticity:** $E$ cannot spoof data from other nodes



# Adversary's Objectives

1. The adversary gains knowledge to which she should not have access.

2. The adversary exercises control over the (partial) computation results of a (free-roaming) agent.

3. Upon the agent's return, its results must appear unsuspicious and authentic to the agent's owner.

$\Rightarrow$ use cryptographic protocols against adversary

# What Can Go Wrong?

Designing good cryptographic protocols is hard!

- Signing of encrypted data
- Lack of explicitness (naming, typing, assumptions)

$\Rightarrow$ Signing or encryption oracles

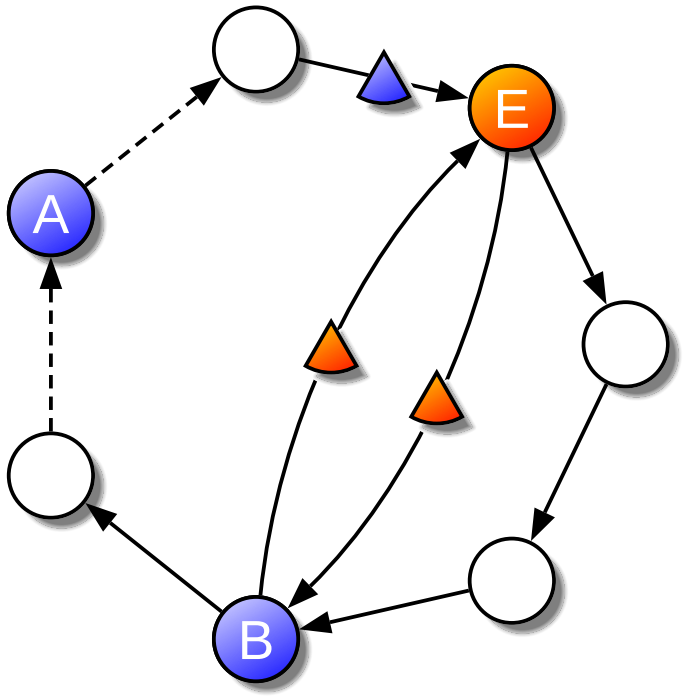
  - Missing identities essential to the meaning of a message
  - No sufficient distinction of different protocol runs

# Decrypting Oracles

Adversary copies ciphertext into her agent; sends it to trusted node

Trusted node innocently decrypts ciphertext

Adversary's agent carries plain text back

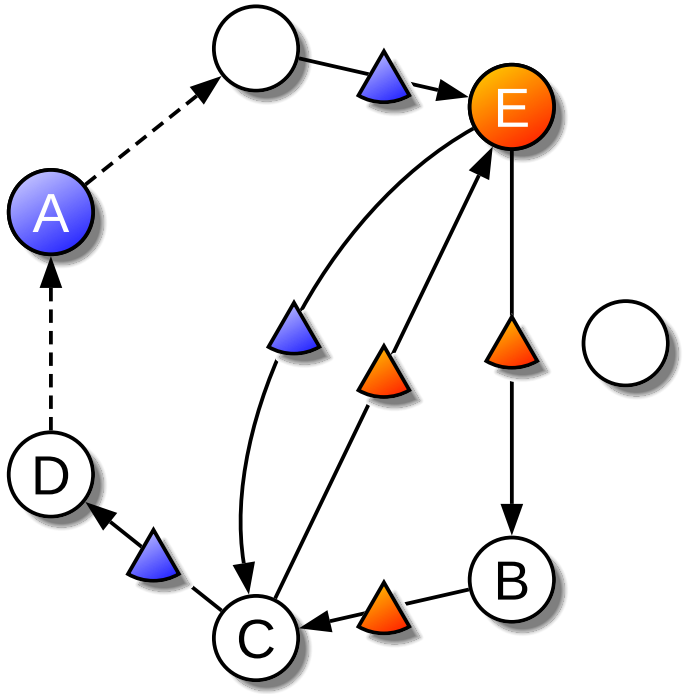


# Signing Oracles

Adversary collects signatures with her own agents

Pastes collected signatures into Alice's agent

Adversary releases Alice's agent



# Flaws in Early SSL

Client $A$ intends to prove its identity to server $B$:

| | |
|---|---|
| Message 1 | $A \rightarrow B : \{K_{ab}\}_{K_b}$ |
| Message 2 | $B \rightarrow A : \{N_b\}_{K_{ab}}$ |
| Message 3 | $A \rightarrow B : \{\mathrm{Cert}_A, \{\boldsymbol{N_b}\}_{\boldsymbol{K_a^{-1}}}\}_{K_{ab}}$ |

Client $A$ can be abused as signing oracle.

due to Martín Abadi, Roger Needham

# Flaws in Early SSL

Server $E$ impersonates client $A$.

| | |
|---|---|
| Message 1 | $A \rightarrow E : \{K_{ae}\}_{K_e}$ |
| Message 1' | $E \rightarrow B : \{K_{ae}\}_{K_b}$ |
| Message 2' | $B \rightarrow E : \{N_b\}_{K_{ae}}$ |
| Message 2 | $E \rightarrow A : \{N_b\}_{K_{ae}}$ |
| Message 3 | $A \rightarrow E : \{\mathrm{Cert}_A, \{N_b\}_{K_a^{-1}}\}_{K_{ae}}$ |
| Message 3' | $E \rightarrow B : \{\mathrm{Cert}_A, \{N_b\}_{K_a^{-1}}\}_{K_{ae}}$ |

# Fixing Early SSL

Identities are $A, B$; the protocol run identifi er is $K_{ab}, N_b$. Message 3 should look like this:

Message 3 $$A \rightarrow B : \{\mathrm{Cert}_A, \{A, B, N_b\}_{K_a^{-1}}\}_{K_{ab}}$$

"If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message."

# Protocol MA vs CS: Differences

- Multiple untrusted parties are involved in the protocol
- Some parties are not known *a priori*
- Agent owner participates only at beginning and end of protocol
- Active messages

# Example: Breach of Confidentiality

The *targeted state* proposed by Karnik *et al.*, 1999

$$\text{Initialization} \quad : \quad \{\{m\}_{K_b} \cdots\}_{K_a^{-1}}$$

$$\text{Migration} \quad : \quad A \xrightarrow{*} B : \Pi_a,\ \{\{m\}_{K_b}\}_{K_a^{-1}}$$

Bob decrypts $\{m\}_{K_b}$ and $\Pi_a$ operates on $m$

(The authors fi xed this protocol meanwhile in a more recent publication)

# Attack

1. Attack with Eve's agent:

$$
\begin{aligned}
A \overset{*}{\rightarrow} E &: \Pi_a, \{\boldsymbol{\{m\}_{K_b}}\}_{K_a^{-1}} \\
E \rightarrow B &: \Pi_e, \{\boldsymbol{\{m\}_{K_b}}\}_{K_e^{-1}} \\
B &: \Pi_e, \{\{m\}_{K_b}\}_{K_b^{-1}} = \boldsymbol{m}
\end{aligned}
$$

2. Eve's agent returns:

$$
B \rightarrow E : \Pi_e, \{\{m\}_{K_b}\}_{K_e^{-1}}, \boldsymbol{m}
$$

# Summary

- Hosts assume that the signer of the encrypted data knows (is allowed to access) the plaintext.
- Protocol data is not associated with a protocol run or entity.
- Consequently, hosts let themselves be abused as oracles.

# Example and Attack: Breach of Authenticity

Chained digital signatures with forward privacy (P2)

$$\mathcal{M}_n = \overbrace{\{\{m_n\}_{K_{i_n}^{-1}}, r_n\}_{K_a}}^{\text{oracle}}, C_n$$

$$C_n = h(\mathcal{M}_{n-1}, r_n, \underbrace{i_{n+1}}_{\text{secret}})$$

$$i_n \rightarrow i_{n+1} \quad : \quad \Pi_a, \{\mathcal{M}_0, \ldots, \mathcal{M}_n\}$$

# Summary

- The protocol does not specify how an agent's owner is determined.
- Eve can collect offers that appear valid to other entities.
- Host let themselves be abused as oracles for signed offers.
- Hosts cannot validate intermediate states (e.g., host $i_{n+1}$ cannot verify that $\mathcal{M}_n$ contains an offer from $i_n$).

How must a protocol be designed?

What is a suitable protocol run identifi er?

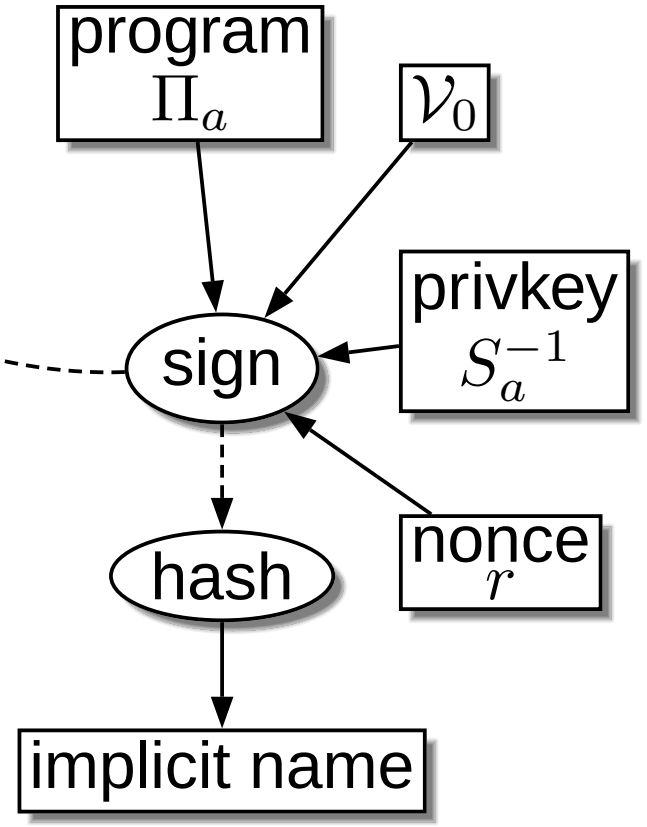
# Protocol Run Identifier

"I, the owner $A$, delegate to agent instance $r$ with program $\Pi_a$ and initial configuration $\mathcal{V}_0$ the right to speak on my behalf."

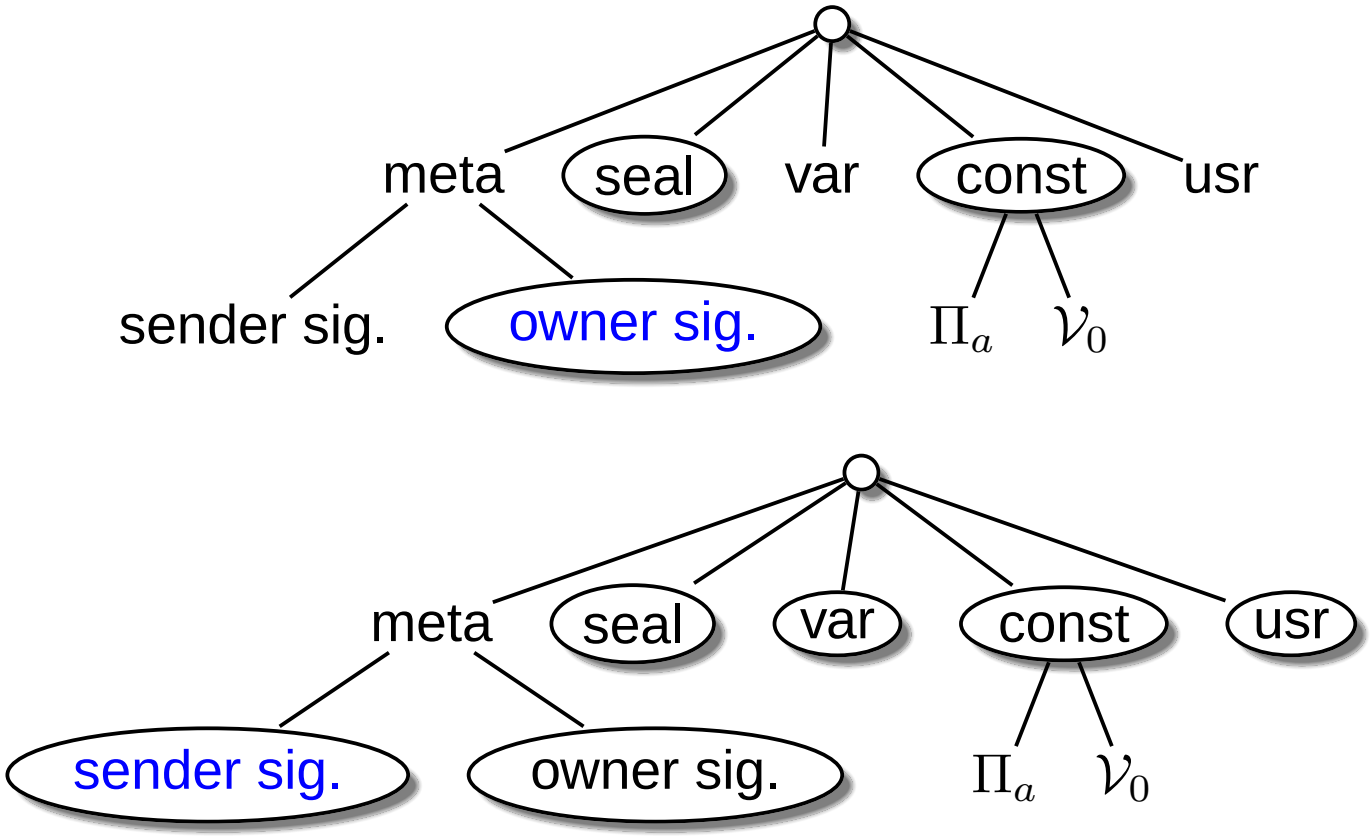
Agent instance $\Leftrightarrow$ protocol run

Implicit name is unique identifier of protocol run

Signing code is not enough, agent *instance* is not unique

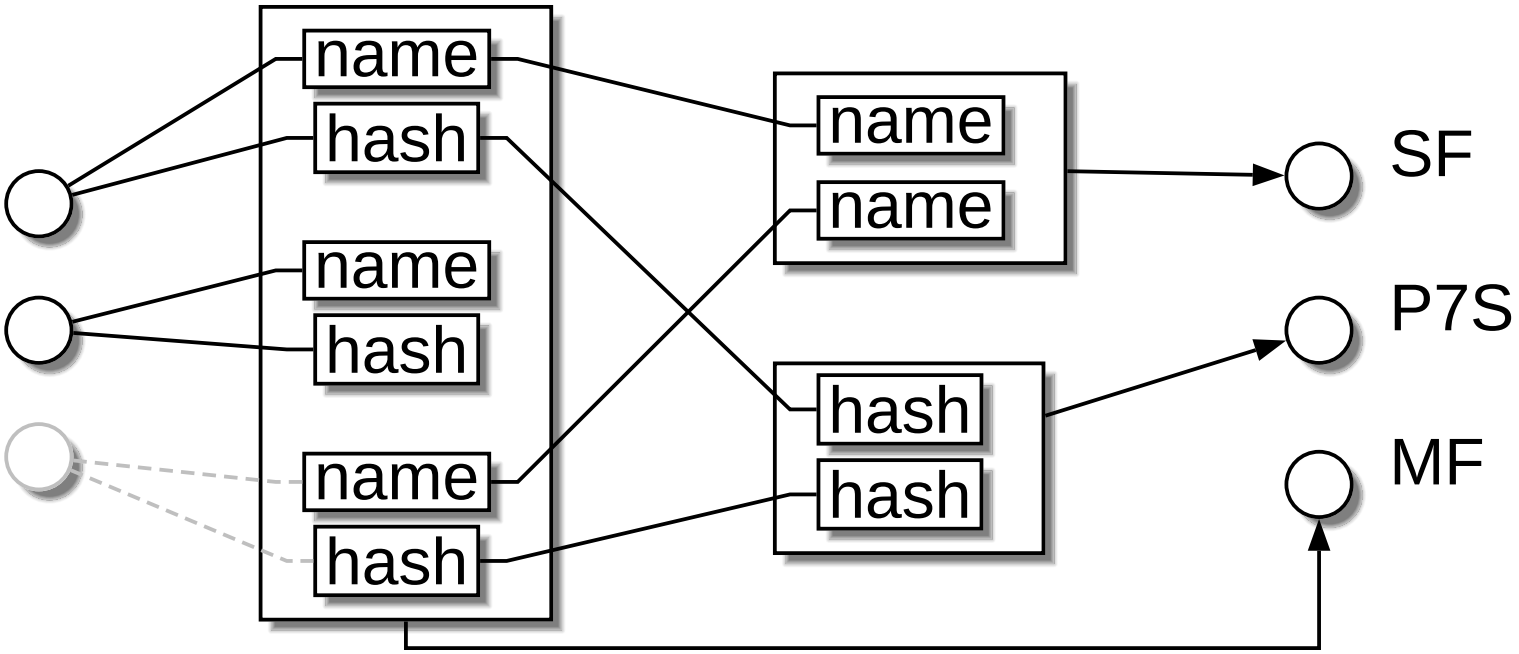


# Signing Agents



# Efficient Signing

Partial signatures, missing fi les, differential Manifests



# Agent File Structure

| *path/name* | | *mark* |
|---|---|---|
| `META-INF/` | **`manifest.mf`** | × |
| | **`owner.sf`** | × |
| | **`owner.p7s`** | ○ |
| | **`sender.p7s`** | × |
| | `prac.bin` | × |
| | *i*`.dmf` | × |
| `SEAL-INF/` | `owner.cert` | ● |
| | `install.cfg` | ● |
| | *i*`.p7m` | ● |
| | *i*`.ear` | ○ |
| `VAR-INF/` | *i*`.ear` | ○ |
| | *i*`.p7m` | ○ |
| | *i*`.p7s` | ○ |
| *name*`.class` | | ● |

| Extension | Formatting |
|---|---|
| `mf` | ManifestFile‡ |
| `sf` | SignatureFile‡ |
| `dmf` | ManifestSections‡ |
| `p7s` | PKCS#7 SignedData† |
| `p7m` | PKCS#7 EnvelopedData† |
| `ear` | raw encrypted ZIP fi le |
| `cfg` | InstallFile‡ |
| `bin` | binary data |
| `cert` | X.509v3 Certifi cate |

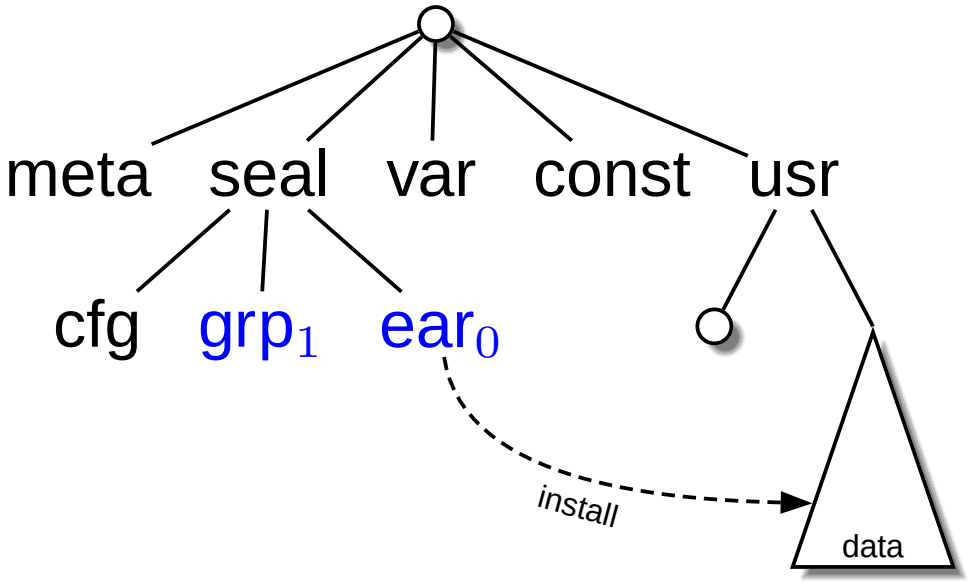
# Revealing Plain Texts

Agent arrives

Decrypt it transparently

Install plain text

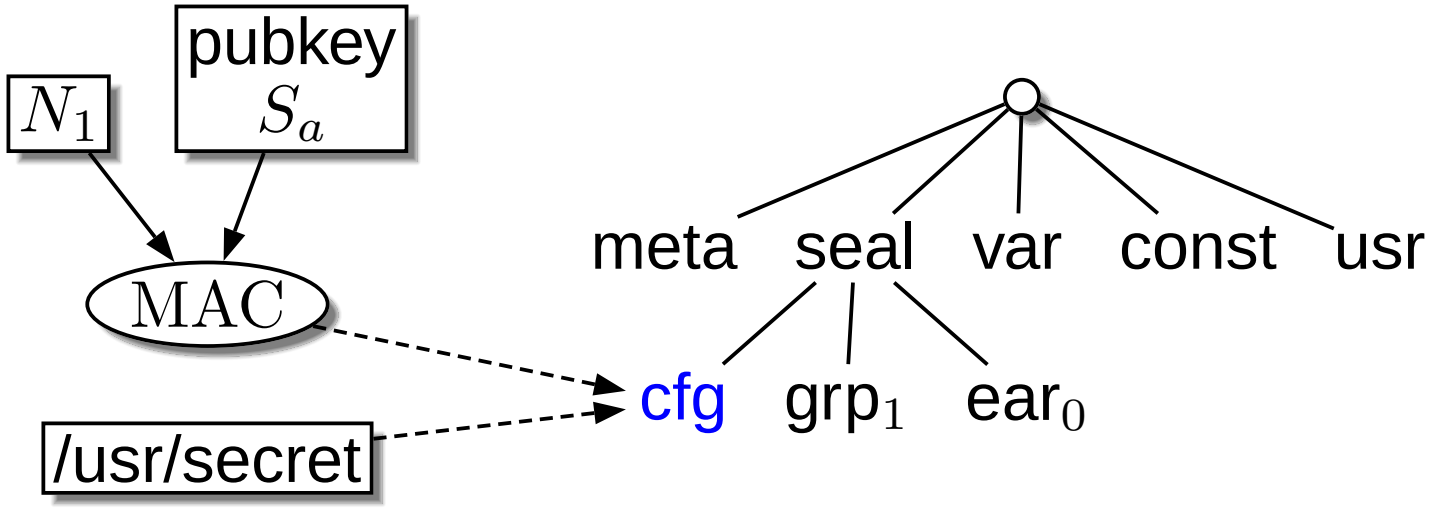
Run agent




$\text{ear}_0 = \{\text{zip}(\text{data})\}_{N_1} \quad \text{grp}_1 = \{(B, \{N_1\}_{K_b}), (C, \{N_1\}_{K_c})\}$

(This data is signed by the agent's owner.)

# Avoiding Decryption Oracles




$\mathrm{MAC}$ computation requires knowledge of $N_1$
Owner signature must match public key input in $\mathrm{MAC}$

Proves that agent owner knows $N_1$ (cfg is signed by owner)
Does not require special syntax for $\text{ear}_0$, saves signature ops

# Packaging Partial Results

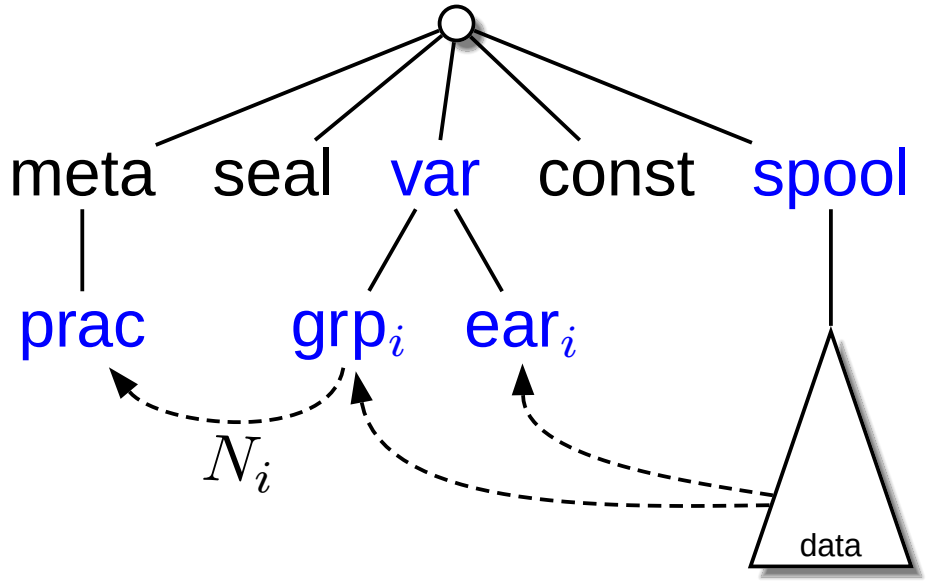
Agent spools results

Encrypt it transparently

Update PRAC

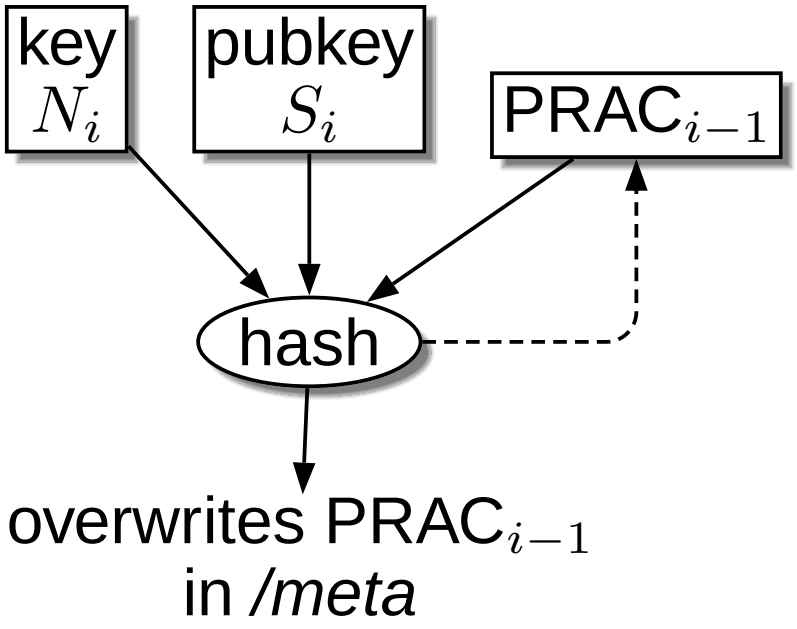
Delete plain text

Sign & transport agent




Partial results are encrypted for agent's owner,
encryption certificate in seal folder (signed by owner)

# Updating the PRAC



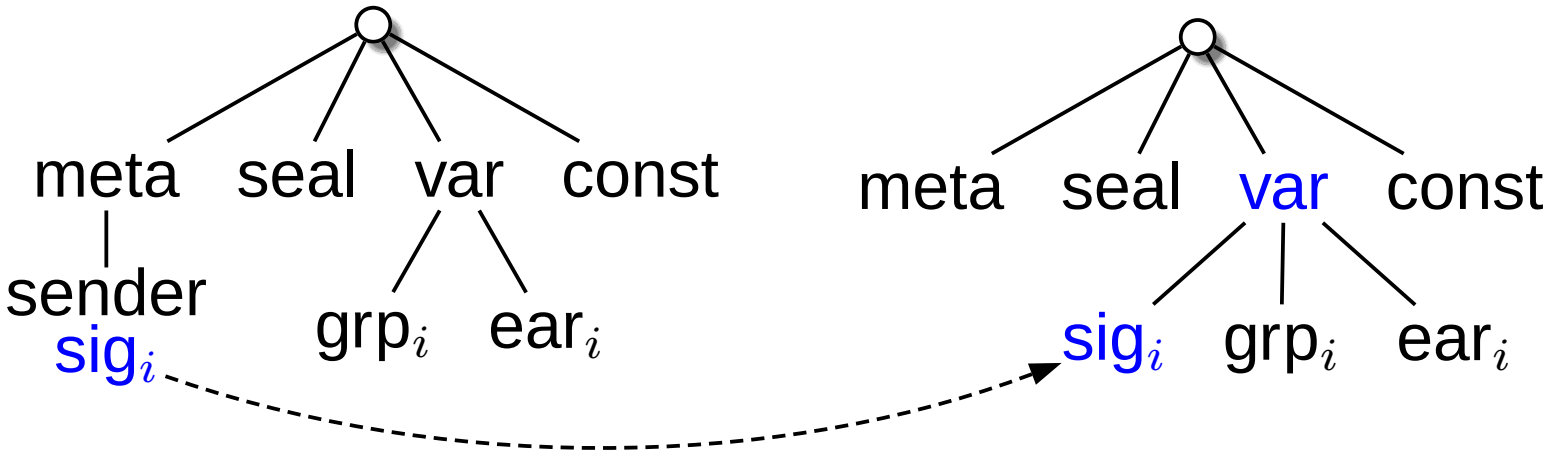

Sender $i$ signs entire agent incl. partial results

Input of $N_i$ and $S_i$ into PRAC$_i$ proves that $i$ knew $N_i$

Initial PRAC$_a$ is chosen randomly, kept for verifi cation

Counters attack: host $i$ *e.g.,* strips signature of host $i-1$, signs agent, and claims to be originator of partial result $i-1$

# Saving Previous Signatures




Agent hops from host $i$ to host $i+1$

Host $i+1$ detects partial result $i$

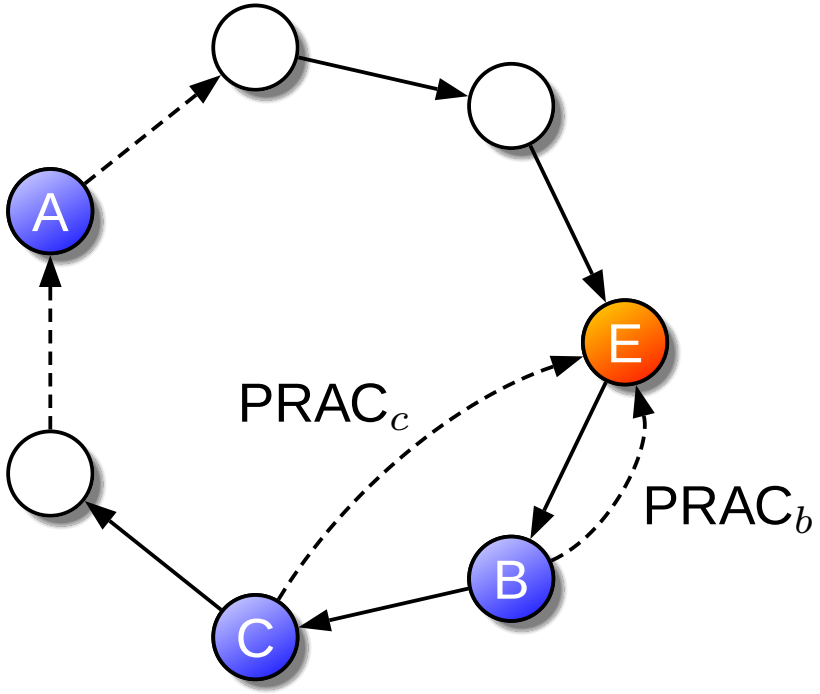
Saves signature of previous host $i$ in the agent

# Effect of Signatures & PRAC

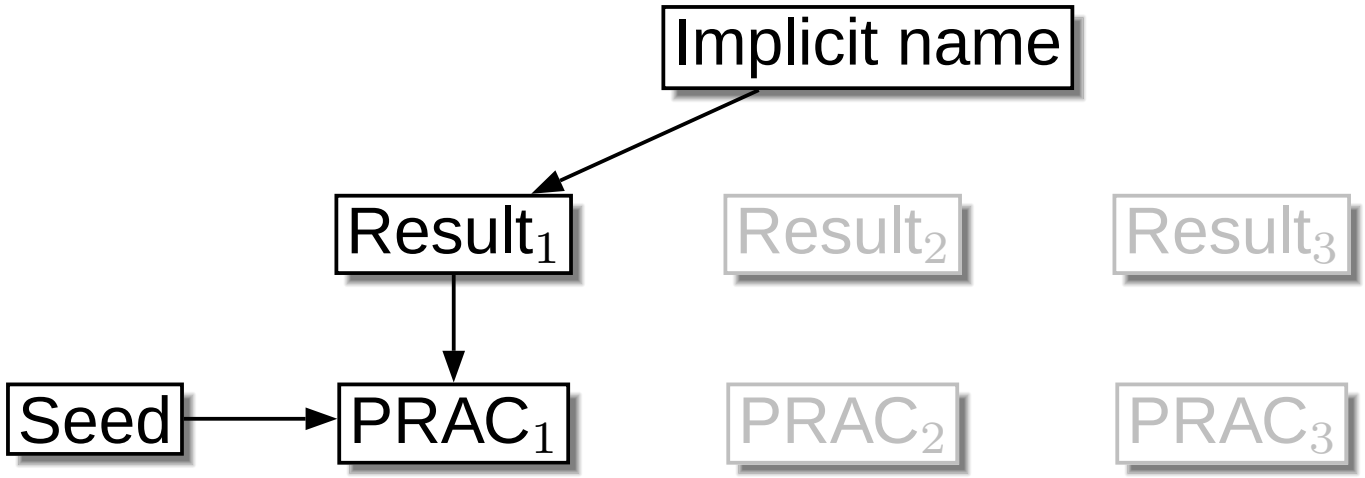
If agent hops from $B$ back to $E$ then $E$ can delete partial result of host $B$

If agent hops from $C$ back to $E$ then $E$ does not know PRAC$_b$

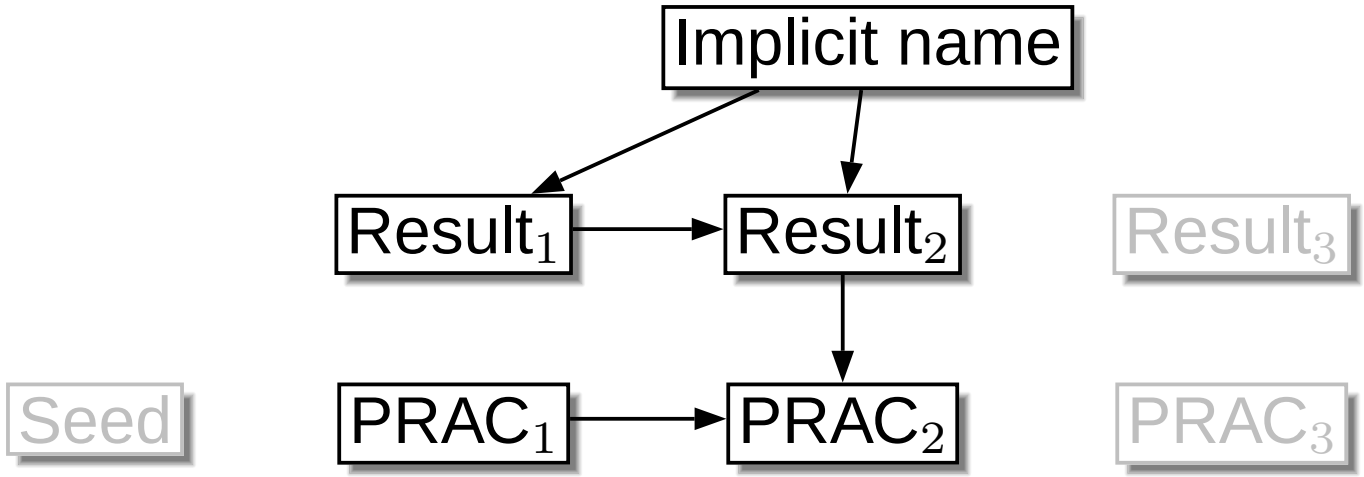
$\Rightarrow$ $E$ cannot delete partial result of $C$ without deleting partial result of $B$ as well



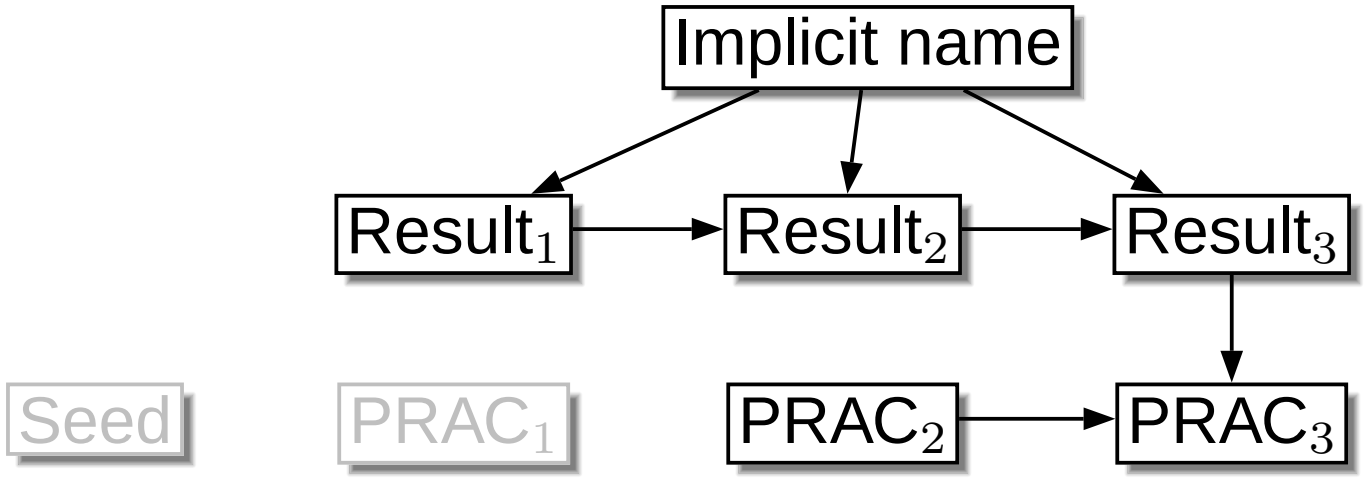
# Effect of Signatures & PRAC



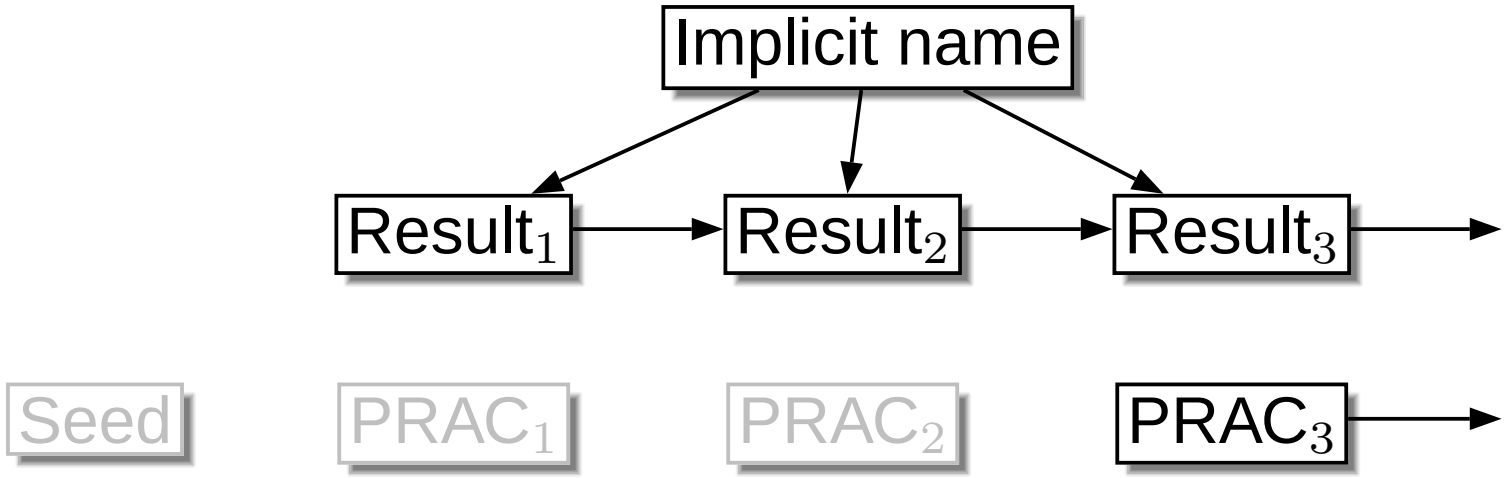
# Effect of Signatures & PRAC



# Effect of Signatures & PRAC



# Effect of Signatures & PRAC

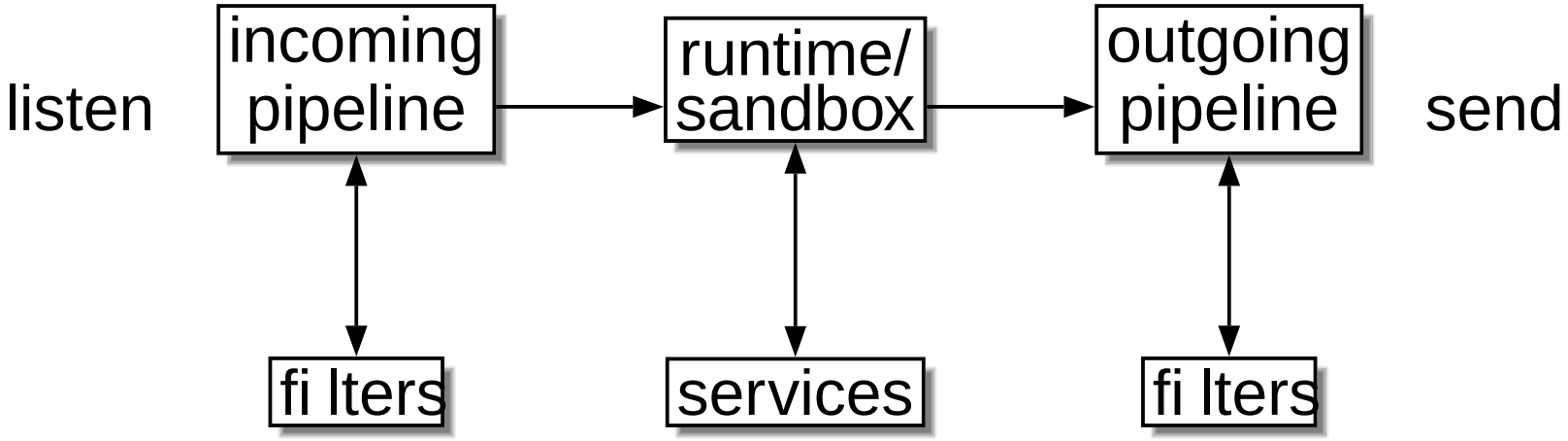


# Verification

Subsequent to her agent's return, Alice

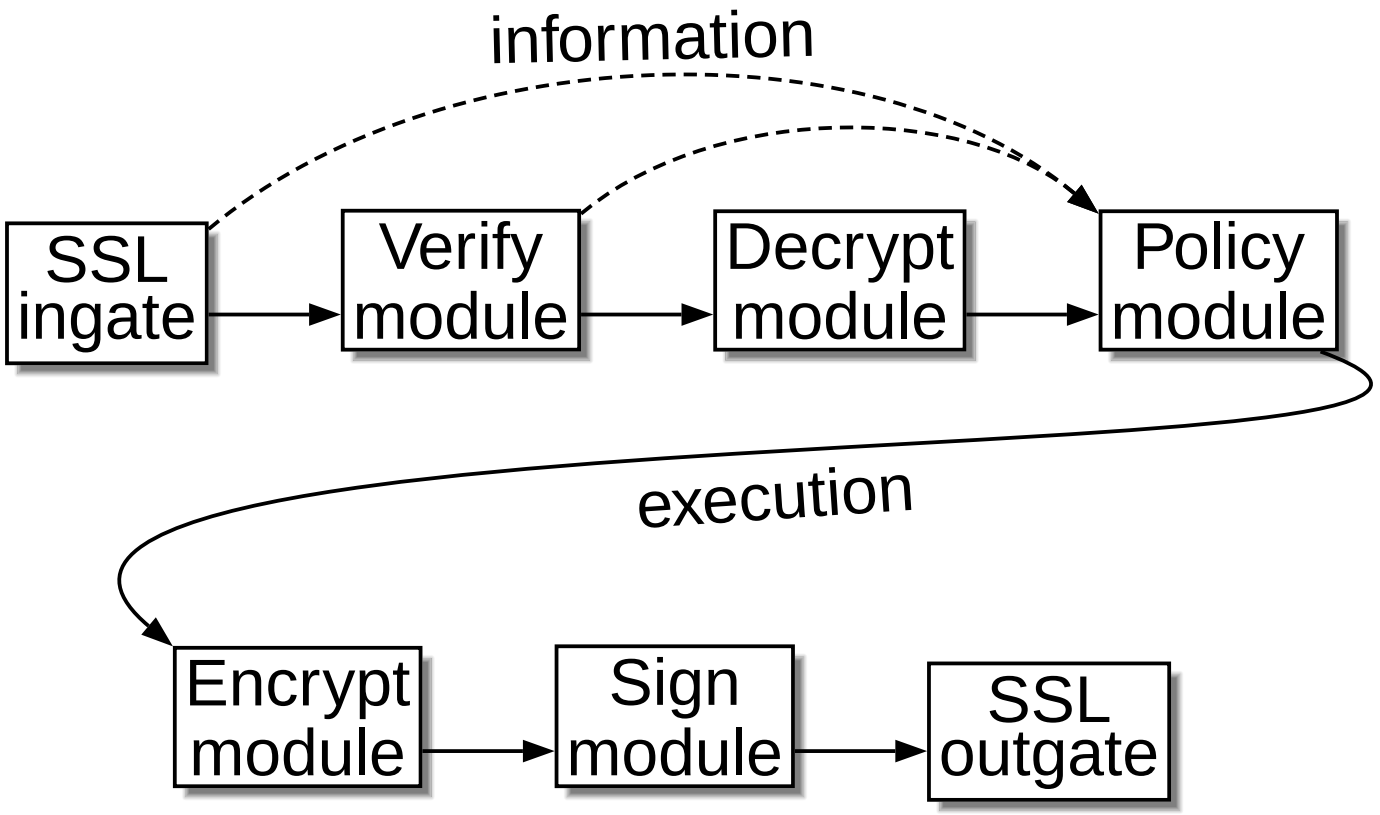
- First pass verifi es signatures backwards.
  - Restore manifest, verify actual signature.
  - Verify MF entries of archive, seal, and previous signature.
- Second pass verifi es PRAC forwards.
  - First pass gives sequence of signers.
  - Extract secrets from seals.
  - Iterate PRAC computation from start value.
  - Compare result.

# Implementation

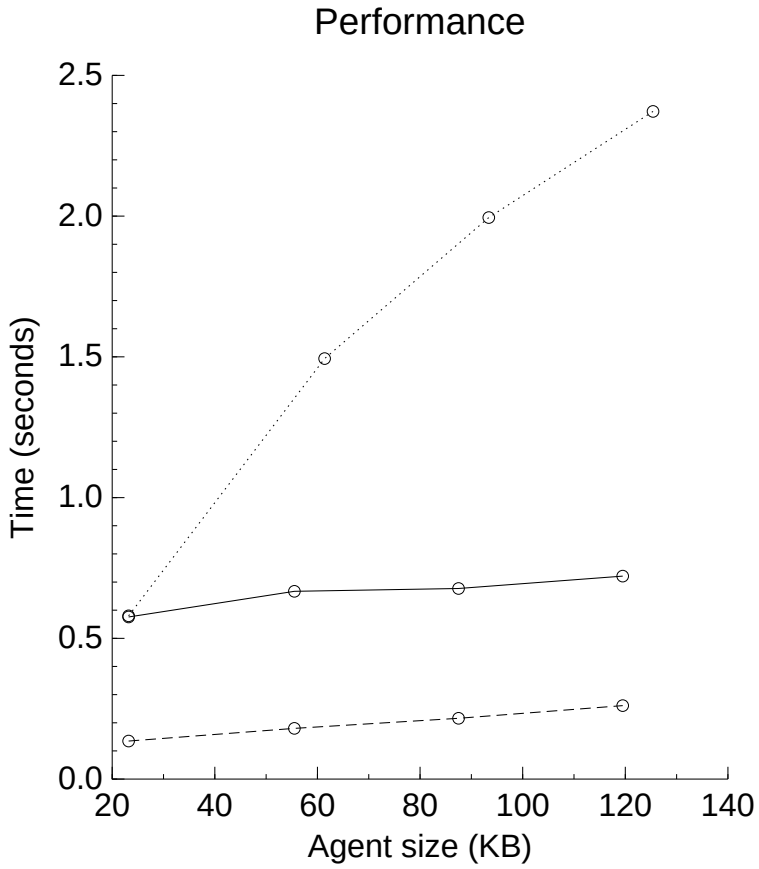



Cryptographic processing of agent is transparent to agent and programmer

# Implementation Cont'd



# Performance (No Partial Results Encryption)




- 4 $\times$ Sun Ultra 5/10 Solaris 8
- JDK 1.3.0_01 HotSpot VM native threads, sunjit
- Signing: SHA1+MD5, DSA; Encryption: RSA, DESede
- Payloads: 0KB, 32KB, 64KB, 96KB
- 600 hops / experiment

# Summary

- Hierarchical fi lesystem structure with security semantics
- Flexible and transparent security services
- Effi cient use of public key operations
  (3 $\times$ verify, 1 $\times$ sign per hop, signatures are re-used)
- Reduced risk of signing & decryption oracles despite signing after encryption

# How To Write Mobile Code?

Adversary cannot use arbitrary program in attack

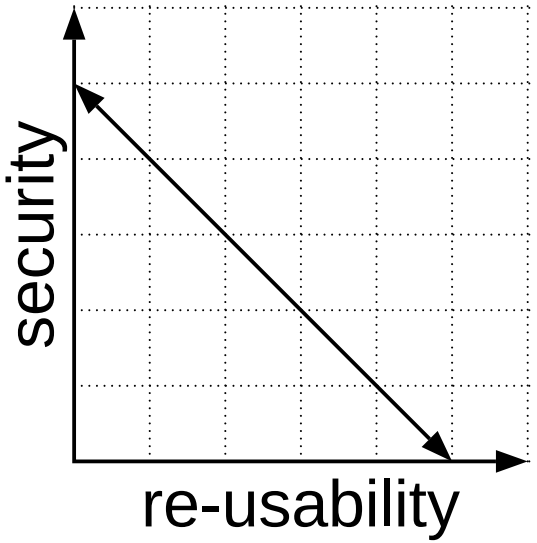
Adversary can manipulate mutable state of Alice's agent

The more reconfigurable, the greater the adversary's freedom

Reduce re-usability by hardwiring partial state (signed $\mathcal{V}_0$)

Still – is the agent program secure?

Software engineering of mobile code



Thank you! Questions?