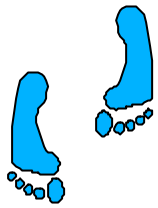
# *Secure Sessions*

*Dave Otway*

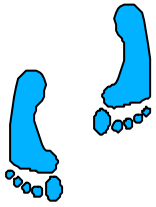
12 November 97

# *Progression*



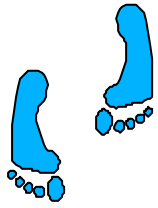
Kamae (MoD)

- Secure RMI (Java)
  - cipher and (2 level) digest sessions
  - active labels



E2S (Esprit)

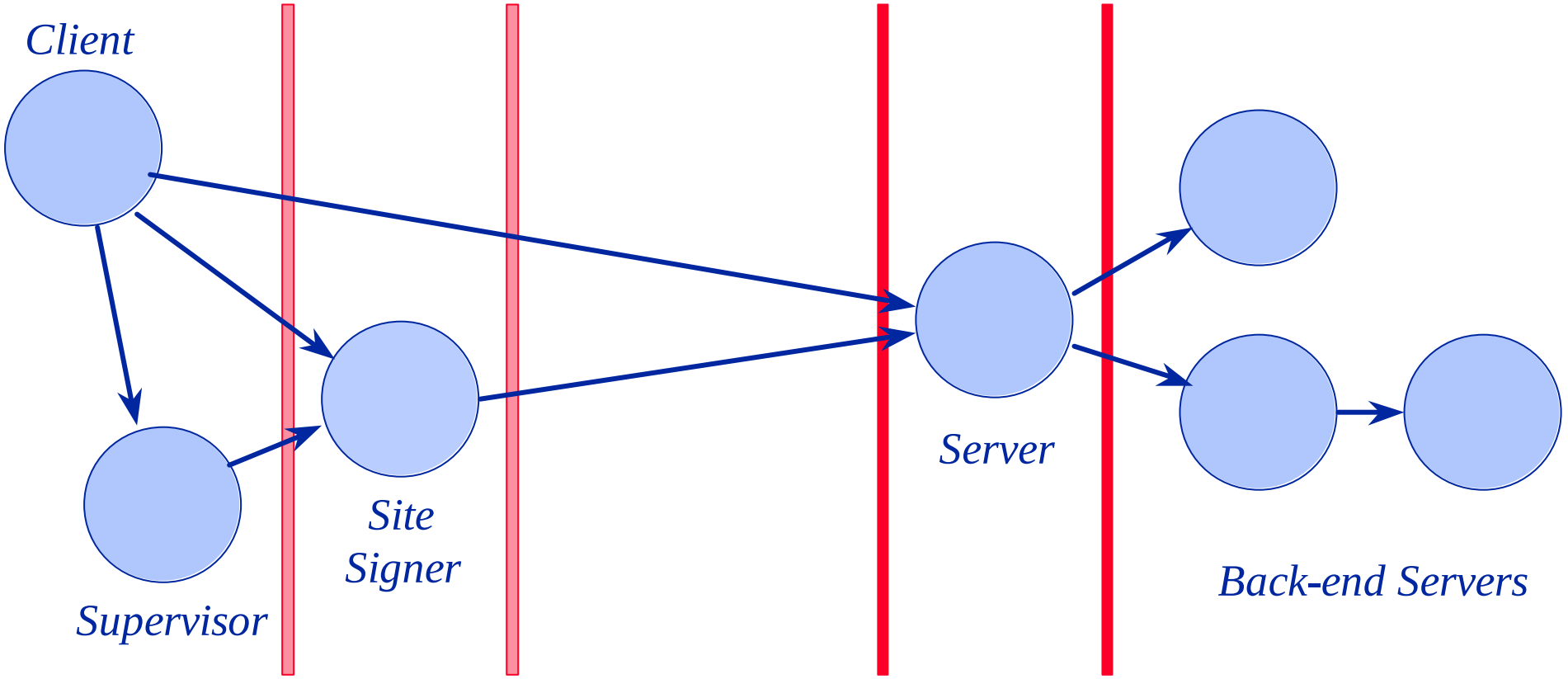
- retrofitted (in Java) onto OrbixWeb
  - with (2 level) digesting at application level
- then translated into C++ on Orbix



Flexinet (ANSA)

- fully recursive application level digests
  - automatically serialized or explicitly marshalled
- integrated with Flexinet cipher sessions

# Securing Distributed Objects




# *Assumptions and Principles*

- Assume necessary and sufficient infrastructure for:
  - strong object and process encapsulation
    - process encapsulation is OS dependent
    - object encapsulation is language dependent
  - application processes being responsible for their own protection
- Assume emerging smart card technology
- Maximize transparency of mechanism
- Apply same principles and protocols everywhere
  - including security service objects

# *Secure RMI Objectives*

- Applications in total control
- Fully transparent engineering

in conflict

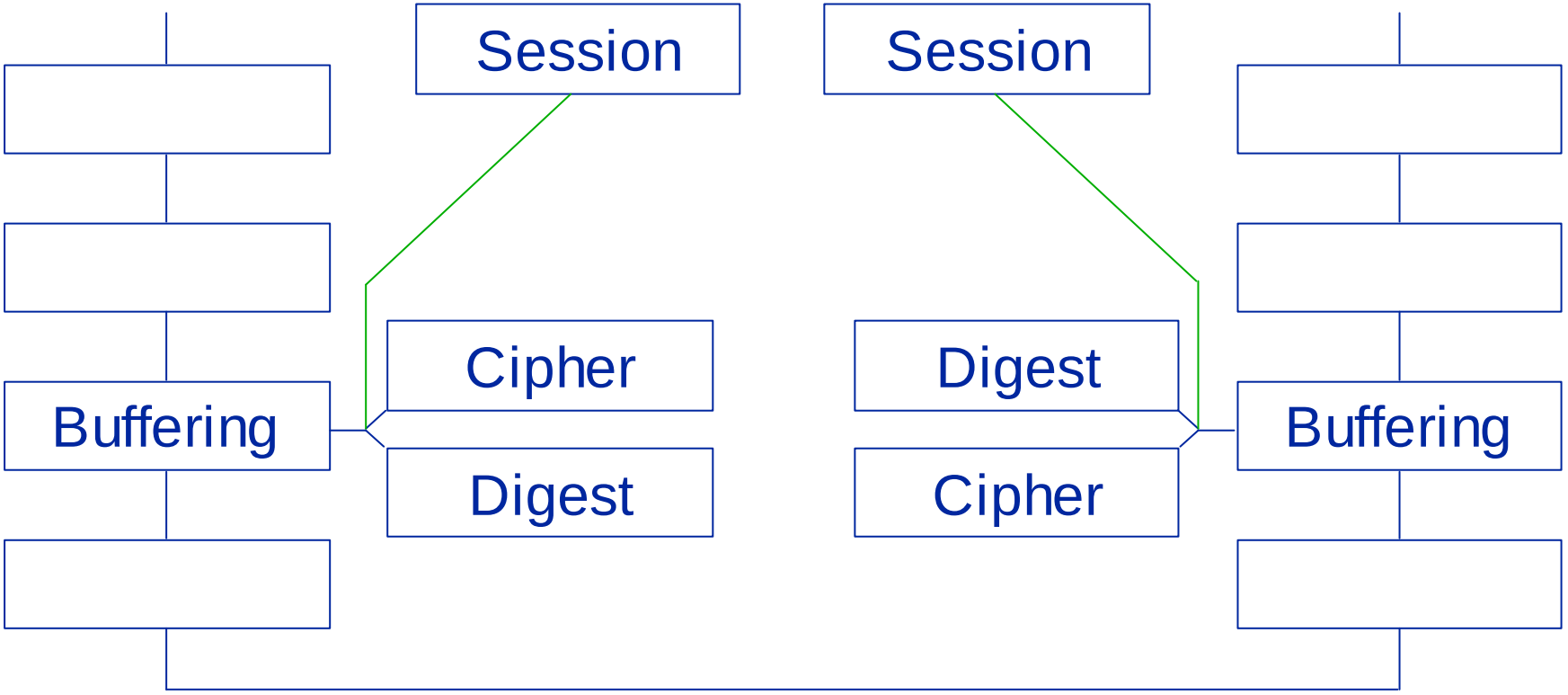
  - conflict handled by separating concerns

- Allow interworking with non-secured applications
- Allow local or remote control of policy

# *Minimal Protocol Mechanisms*

- Mechanisms necessary and sufficient for purpose
  - keyed digests for integrity and authentication
  - XORed keyed digests for key distribution
  - symmetric encryption for confidentiality
  - (asymmetric signed digests for non-repudiation)
- Hash functions
  - fixed length output
  - faster
  - less legal problems in commerce (COTS)

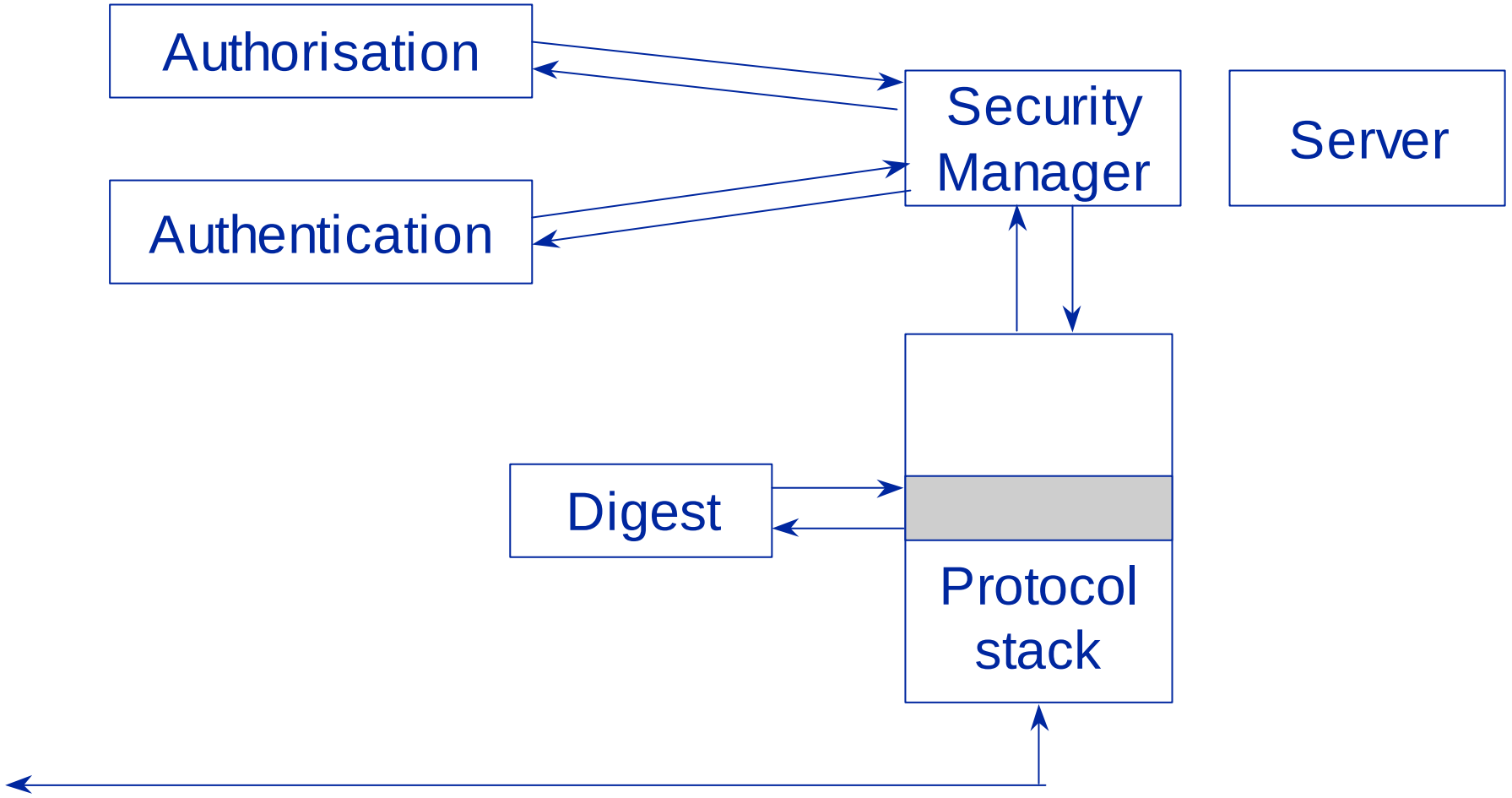
# *Secure RMI Structure*



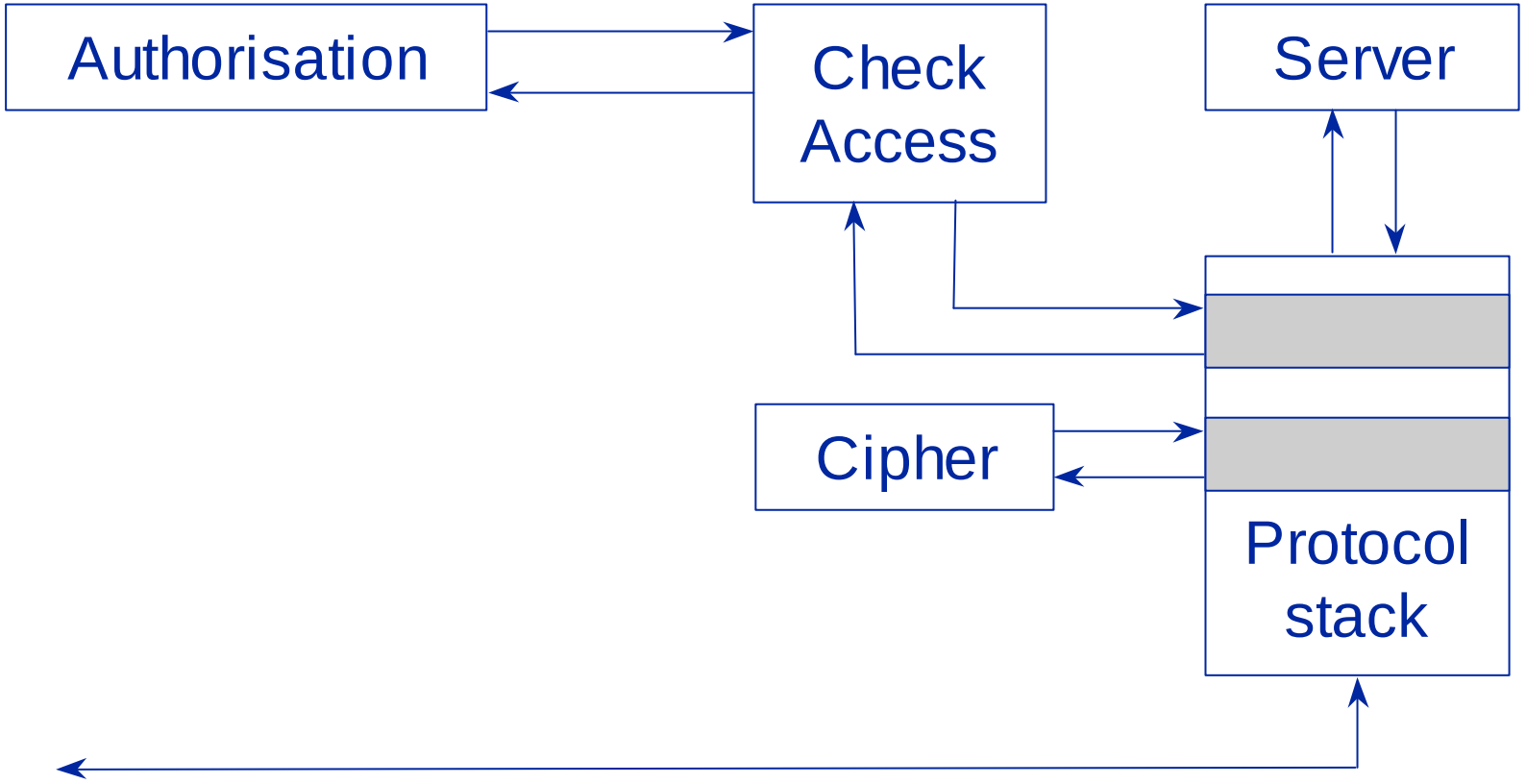

# *Secure RMI Engineering*

- An application security manager added to the server
- Client bound to security manager instead of server
  - use digest session to manager to open server
  - use cipher session to access server
- Server application can take control
  - or delegate to third party security servers
- No security parameters in server methods
  - check access called before each method

# *Opening a Server*



# *Per Method Authorisation*




# *Protocol Properties*

- Generalisation to client-server chains
  - recursive protocol
  - session keys returned to each pair of participants
- Overlay of authentication and delegation
- Symmetric call and return engineering
- Simplifies federation of administrative domains
  - its just another indirection

# *Protocol Description*

a -> b : Da = [a, b, Na, --]Ka

b -> c : Db = [b, c, Nb, Da]Kb

c -> k : [c, k, Nc, Db]Kc

k -> c : [c, b, Nc, Nb, Kcb⊕<c,b,Nc,Nb>Kc, Eb]Kcb

c -> b : Eb = [b, c, Nb, Nc, Kbc⊕<b,c,Nb,Nc>Kb,

[b, a, Nb, Na, Kba⊕<b,a,Nb,Na>Kb,Ea]Kba]Kbc

b -> a : Ea = [a, b, Na, Nb, Kab⊕<a,b,Na,Nb>Ka, --]Kab

# *Lessons Learned from Kamae*

- it is feasable to transparently security a remote interface
- but its hard to nest and unnest digests as byte arrays
  - and its very hard to add and extract applications arguments
- it is possible to almost hide the distinction
  - between digest and cipher references
  - but not between them and clear references
- the application managers have to be aware of digest and cipher sessions

# *Conclusion*

Do digesting and session management at the application level

(in management classes alongside the client and server classes)

# *Migration to OrbixWeb*

OrbixWeb has 3 protocol hooks
(But they don't all work with IIOP)

- Transformers
  - only access to byte stream (encryption)
  - only other info is remote host name and direction
- Per-Object Filters
  - only on server side (per method authorization ?)
- Per-Process Filters (digesting)
  - 8 versions
    pre or post marshalled
    client or server
    request or reply

# *Approach*

- encrypt in the transformers
  - cipher sessions indexed by remote host name
  - if session found then en/de-cipher
    else pass in clear (for authentication digests)
  - add random IV to each cipher message

- authenticate, authorize and distibute session keys in application security managers
  - explicit marshalling of digests into byte arrays
  - explicit session setup

# *Invocation Path*

client outRequestPremarshal

client transformer outgoing

server transformer incoming

server inRequestPreMarshal

server application

server outReplyPreMarshal

server transfomer outgoing

client transformer incoming

client inReplyPreMarshal

# *Problems*

1) an attacker can invoke a server in clear

2) if a client's host machines are behind an IP address translating firewall

   - a second client's transformer would transmit a digest request in clear
   - but server's transformer would decipher it

3) server needs to call back-end servers in clear

# *Lessons Learned from E2S*

- implement sessions for all remote calls
  - add session IDs to all messages
  - plaintext, ciphertext and digested
  - can then have different session key for each user
- provide digest sessions as well as cipher sessions
  - for ensuring integrity when encryption is not allowed
  - or cooperation with the encryption mechanism is difficult
- provide flexible application level digests which:
  - can be automatically serialized or explicitly marshalled
  - can be nested to any depth
  - can enclose application data in both directions

# *Flexinet*

- sessions are fully engineered into the protocol stack
- CipherSession & DigestSession are subclasses of Session
  - no other security engineering in the protocol stack
- fully nested application level protocol for authentication, authorization and key distribution using keyed digests
  - using classes which can be serialized or explicitly marshalled
  - and can carry application data in both directions
- explicit binding of CipherSession & DigestSession
  - interfaces bound to a CipherSession or DigestSession cannot be bound to any other type of session