Decentralized Jini Security

Pekka Nikander & Pasi Eronen
Helsinki University of Technology

Presentation outline

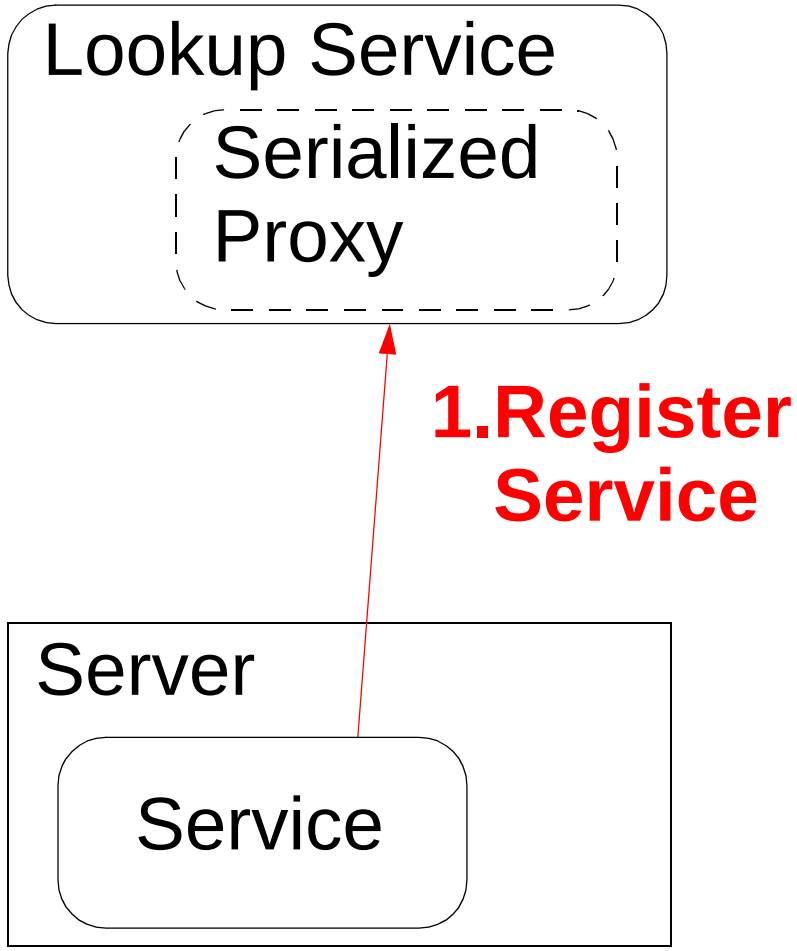
- Background and Goals
- Brief introduction to Jini
- Security in the current Jini Architecture
- Introduction to Trust Management
- The Proposed Architecture
- Conclusions

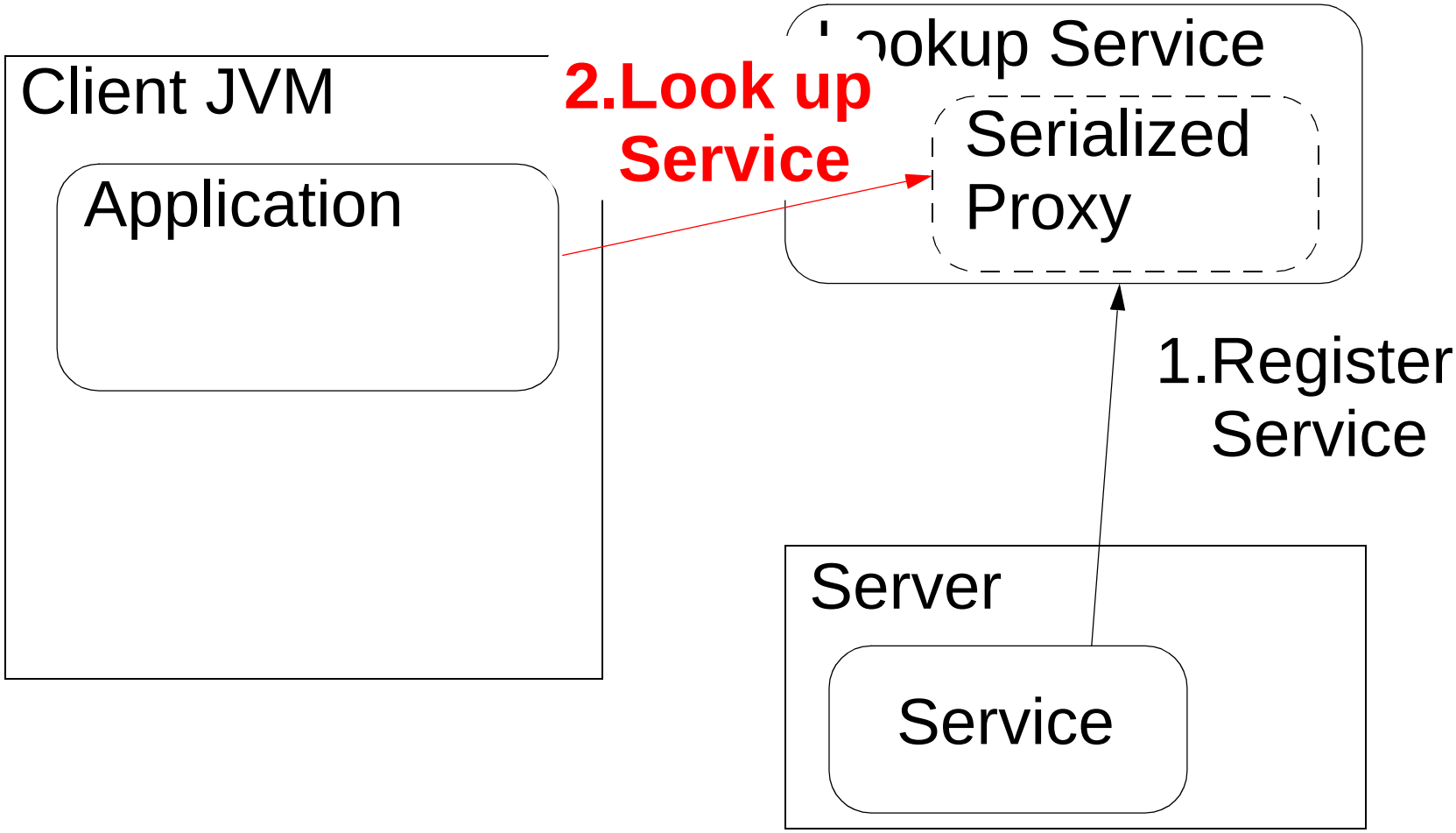
Background and Goals

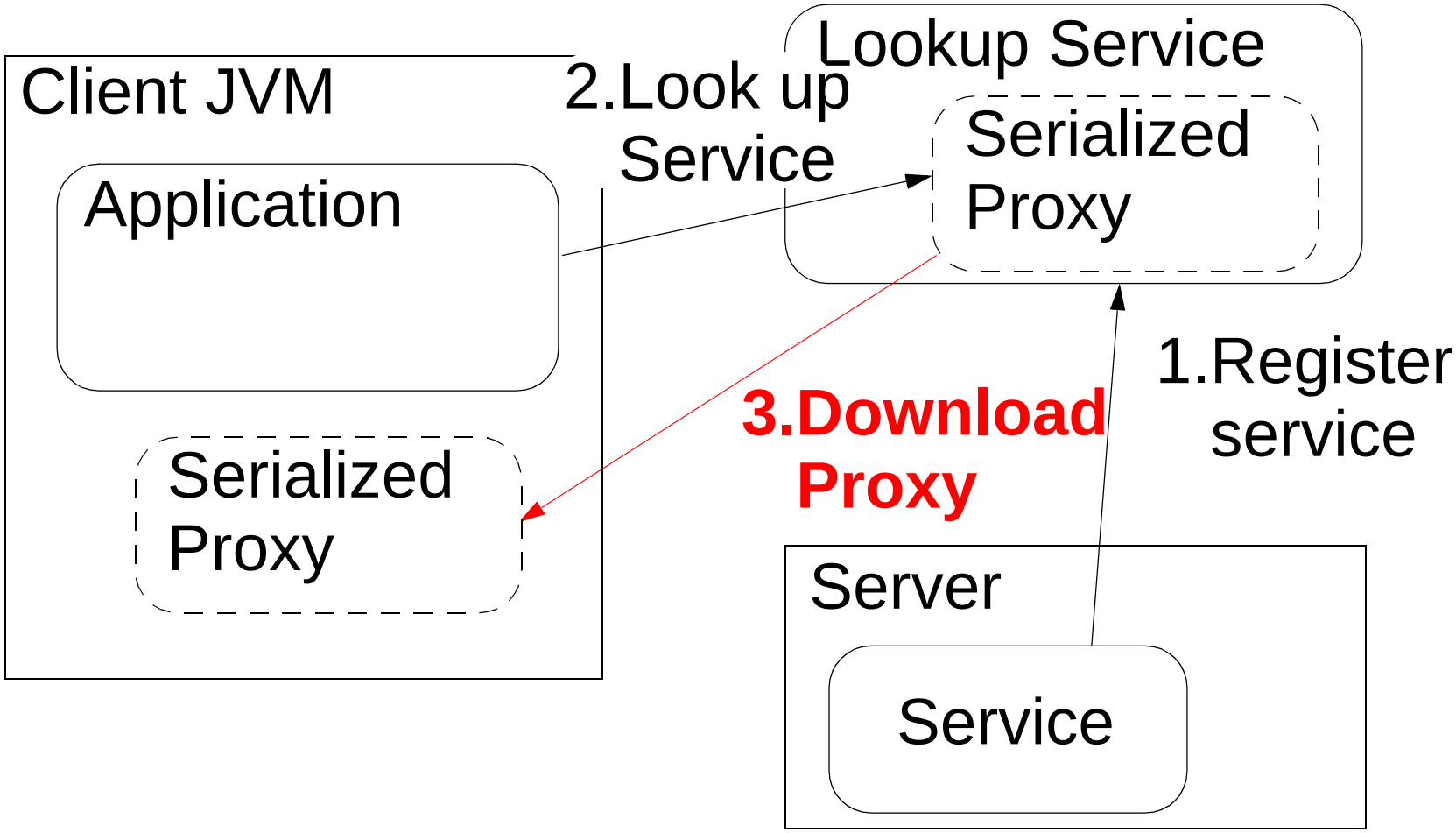
- Research on security in ad hoc networks
- Our prior work: Combining Java 2 Security & SPKI
- Goals of this work
  - A security architecture for Jini
  - No centralized components — fully decentralized
  - Integration into the Java 2 Security model
  - Protocol independency

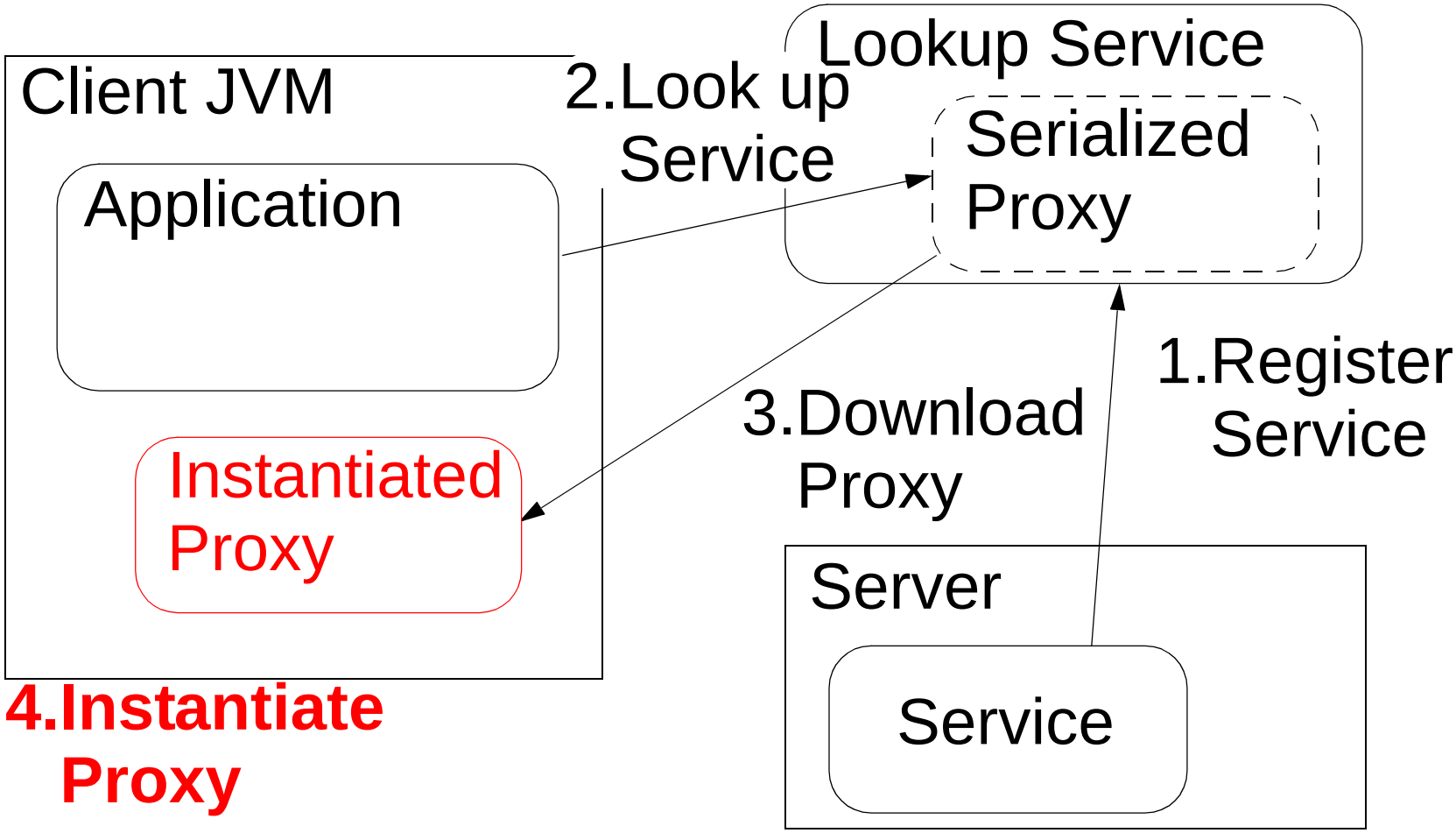
Brief introduction to Jini

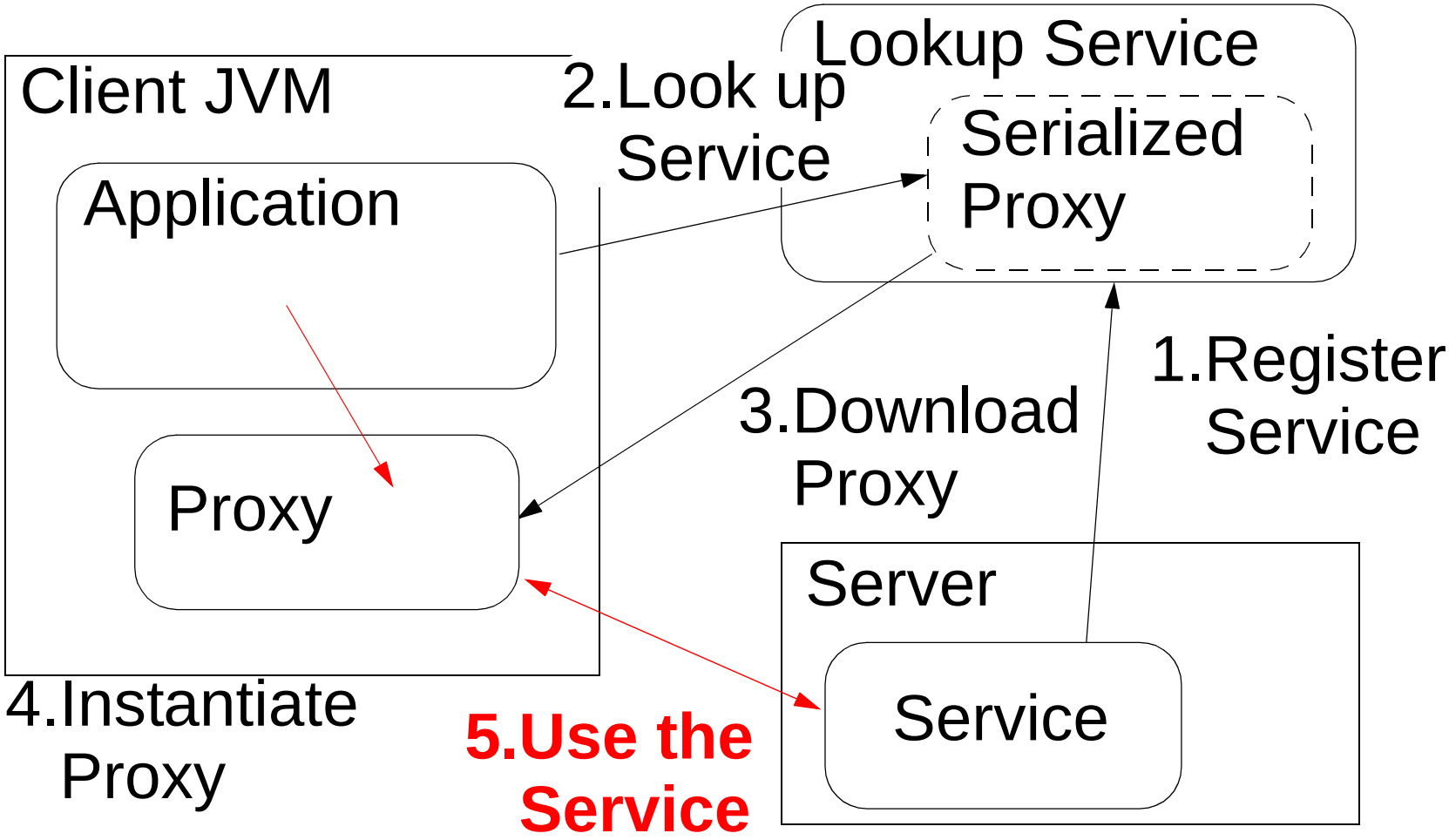
- A decentralized, ad hoc network service architecture
  - Requires no pre-established infrastructure
  - Allows applications to work under partial failures
- Services are build upon leases, events, transactions
  - Services register themselves to a Lookup Service
  - Any node may provide a Lookup Service
- Protocol independence using proxy objects


Lookup Service
Serialized
Proxy
1.Register
Service
Server
Service


Client JVM
Application
2.Look up Service
ookup Service
Serialized Proxy
1.Register Service
Server
Service


Client JVM
Application
Serialized
Proxy
2.Look up
Service
Lookup Service
Serialized
Proxy
1.Register
service
3.Download
Proxy
Server
Service


Lookup Service
Client JVM
2.Look up Service
Serialized Proxy
Application
1.Register Service
3.Download Proxy
Instantiated Proxy
Server
Service
4.Instantiate Proxy


Lookup Service
Client JVM
2.Look up Service
Serialized Proxy
Application
1.Register Service
3.Download Proxy
Proxy
Server
Service
4.Instantiate Proxy
5.Use the Service

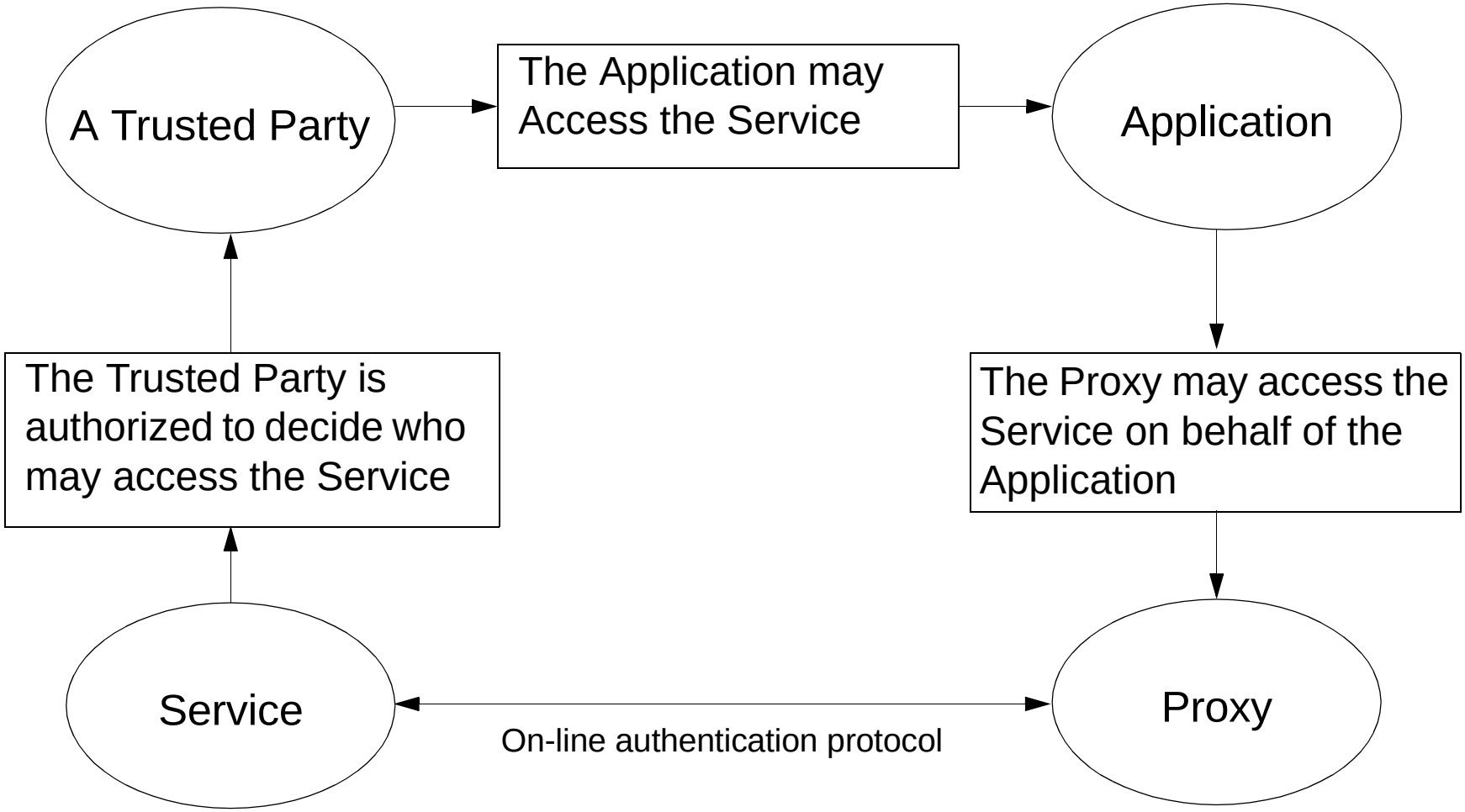
Security in the current Jini Architecture

- The current Jini architecture has no security features beyond those of Standard Java 2 SE
- Other solutions need centralized components
  - Additionally, they often have very restricting trust assumptions, unsuitable for ad hoc environments
- Maybe some day RMI security API will help
  - Not likely in the near future, though

Introduction to Trust Management

- All principals (servers, applications, proxies) are identified by public keys
  - These keys can be anonymous or temporary
  - Key lifetime depends on its purpose
- Authority can be delegated via *Certificate Chains*
- There are a number of existing Trust Manatement systems: PolicyMaker, KeyNote 2, SPKI

Certificate Chains



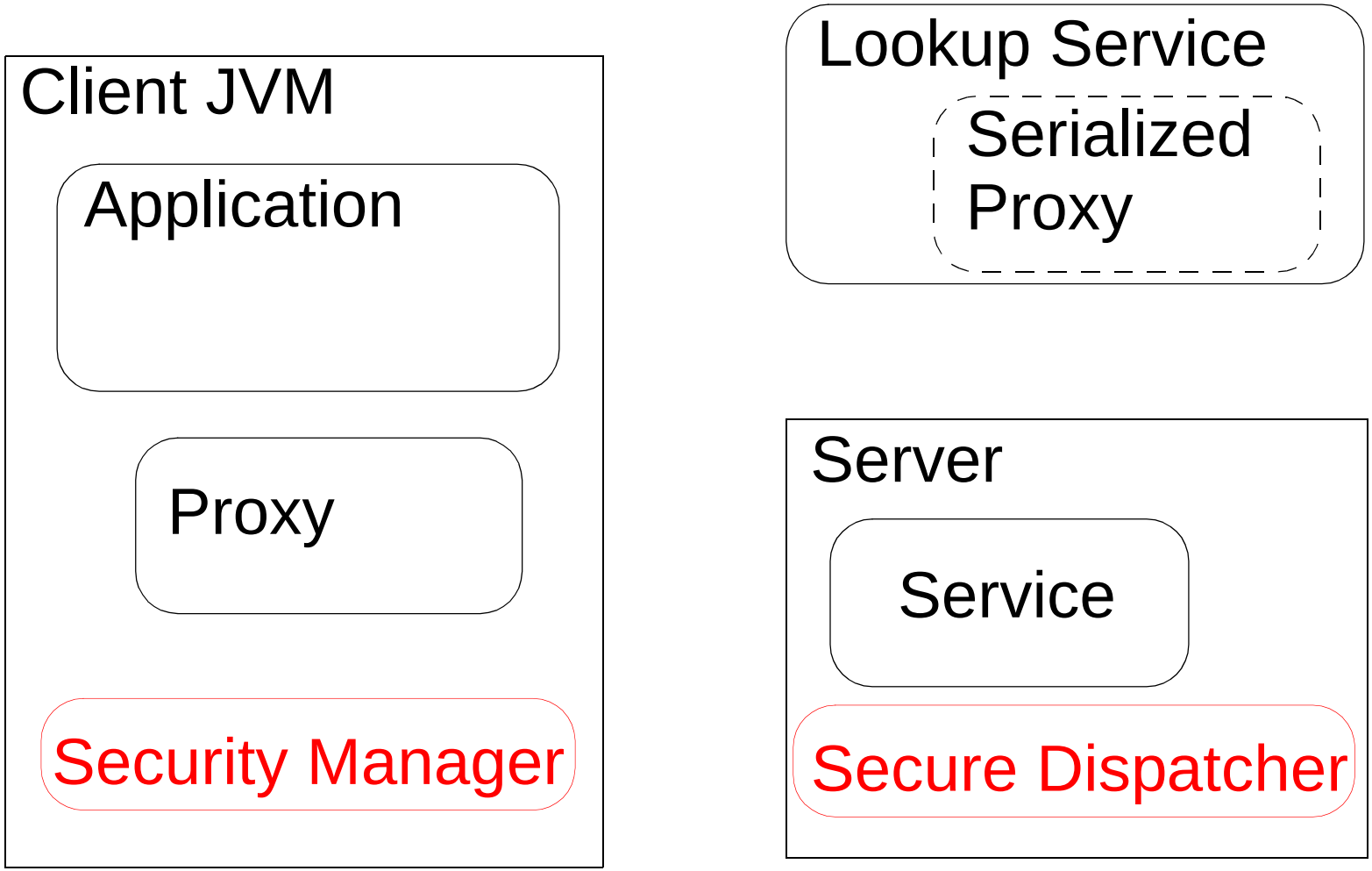
Presentation outline

...

- The Proposed Architecture
  - New functions
  - New steps
  - Benefits
  - Limitations & Future Work
- Conclusions

The Proposed Architecture

- No centralized security servers or CAs
  - Fully peer-to-peer ad hoc structure
- No unnecessary “identities” or “names”
  - All parties are identified by public keys
- Clients JVMs can run partially trusted applications
  - Java 2 Security Architecture + decentralization
- Avoid modifications to JDK and Jini


Client JVM
Application
Proxy
Security Manager
Lookup Service
Serialized
Proxy
Server
Service
Secure Dispatcher

New functions

- Sign proxies / proxy verify signatures
- Generate a temporary key pair for a proxy
- Delegate authority to a proxy’s key
- Sign arbitrary data with a proxy’s key
- Get the public key of a service
- Verify certificate chains

New steps

0. The service signs proxy before registering it

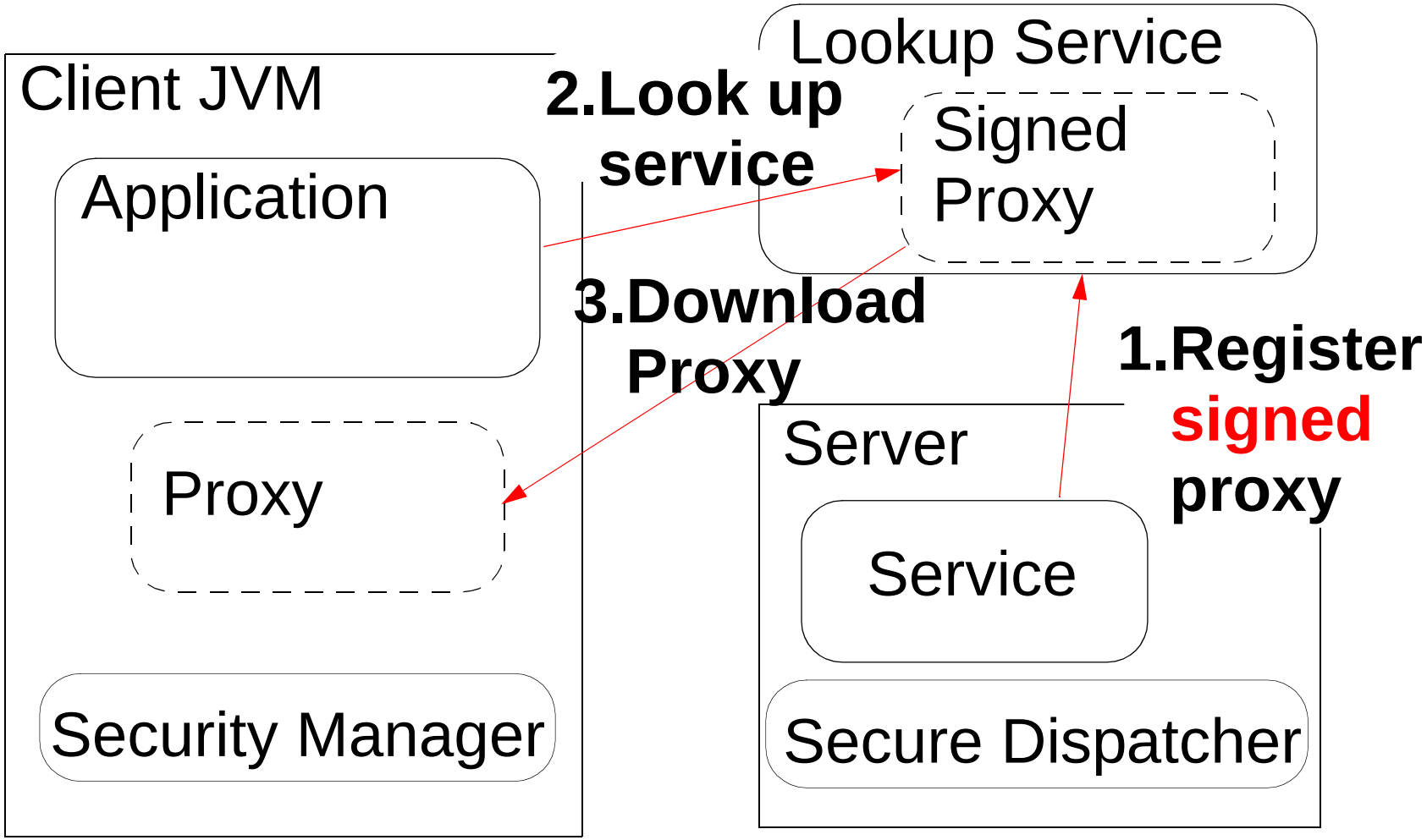
3’. The security manager authenticates the proxy

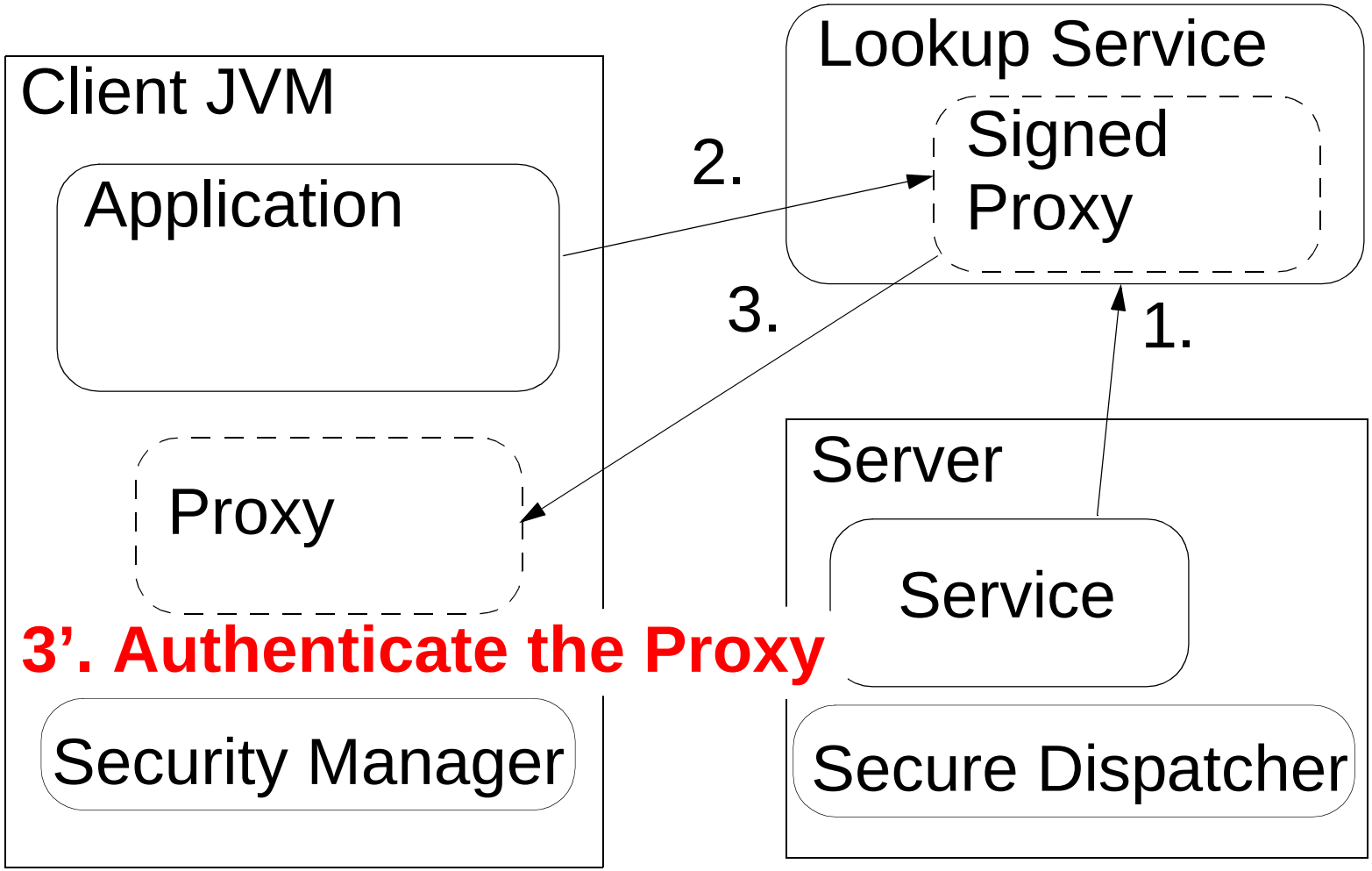
4’. The application accepts & authorizes the proxy

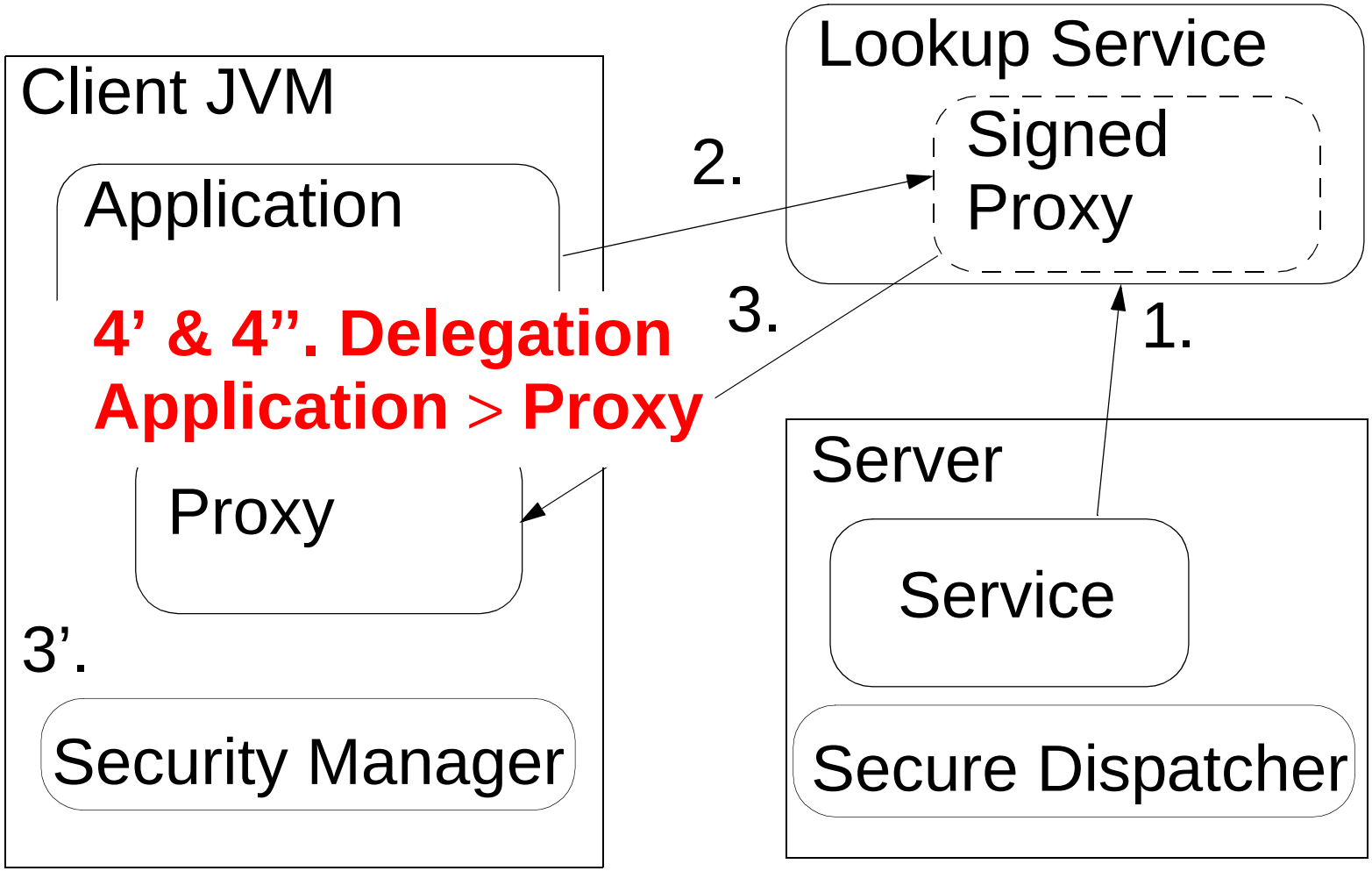
4”. The proxy requests for delegation

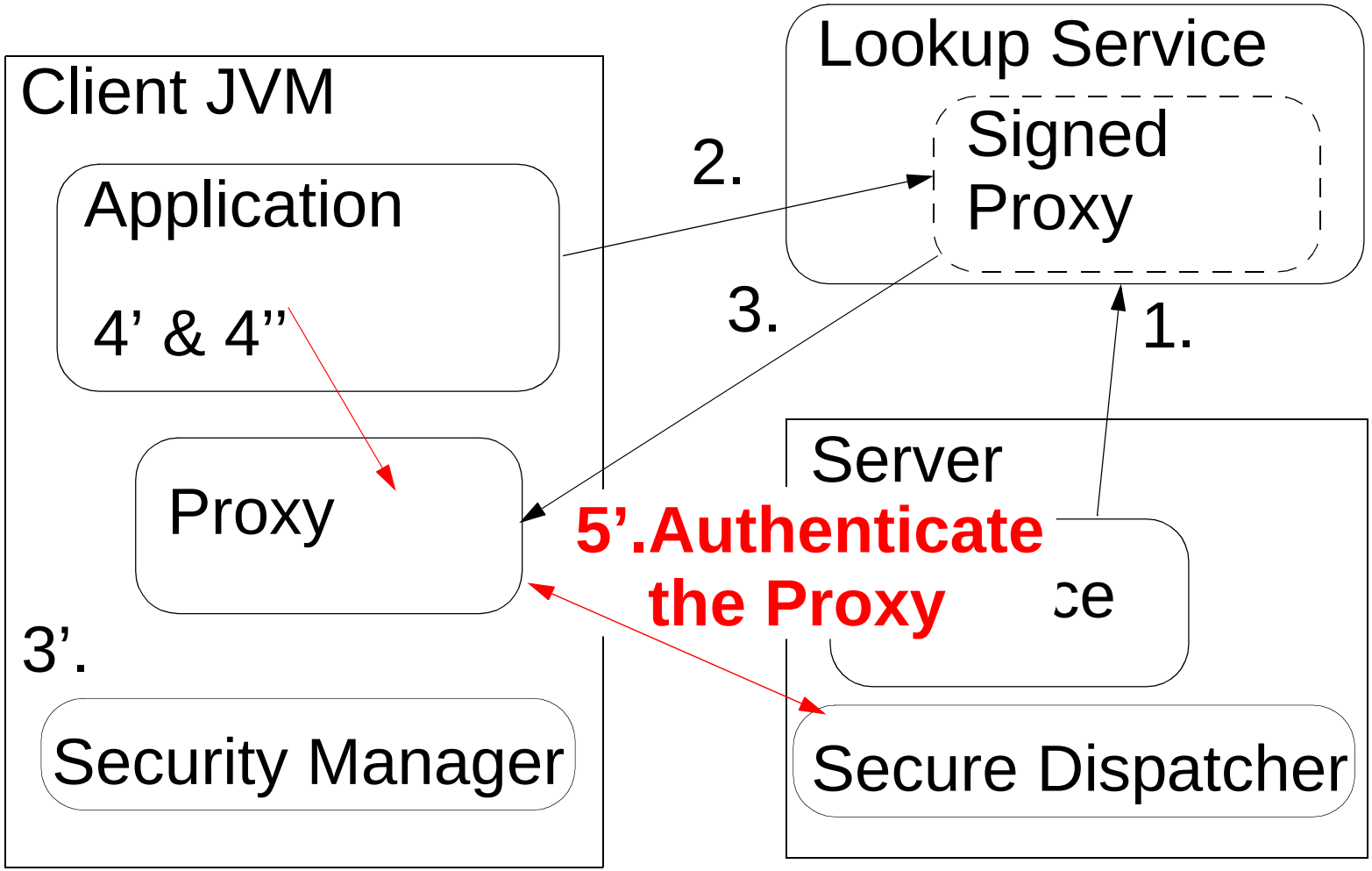
5’. The proxy authenticates itself to the dispatcher

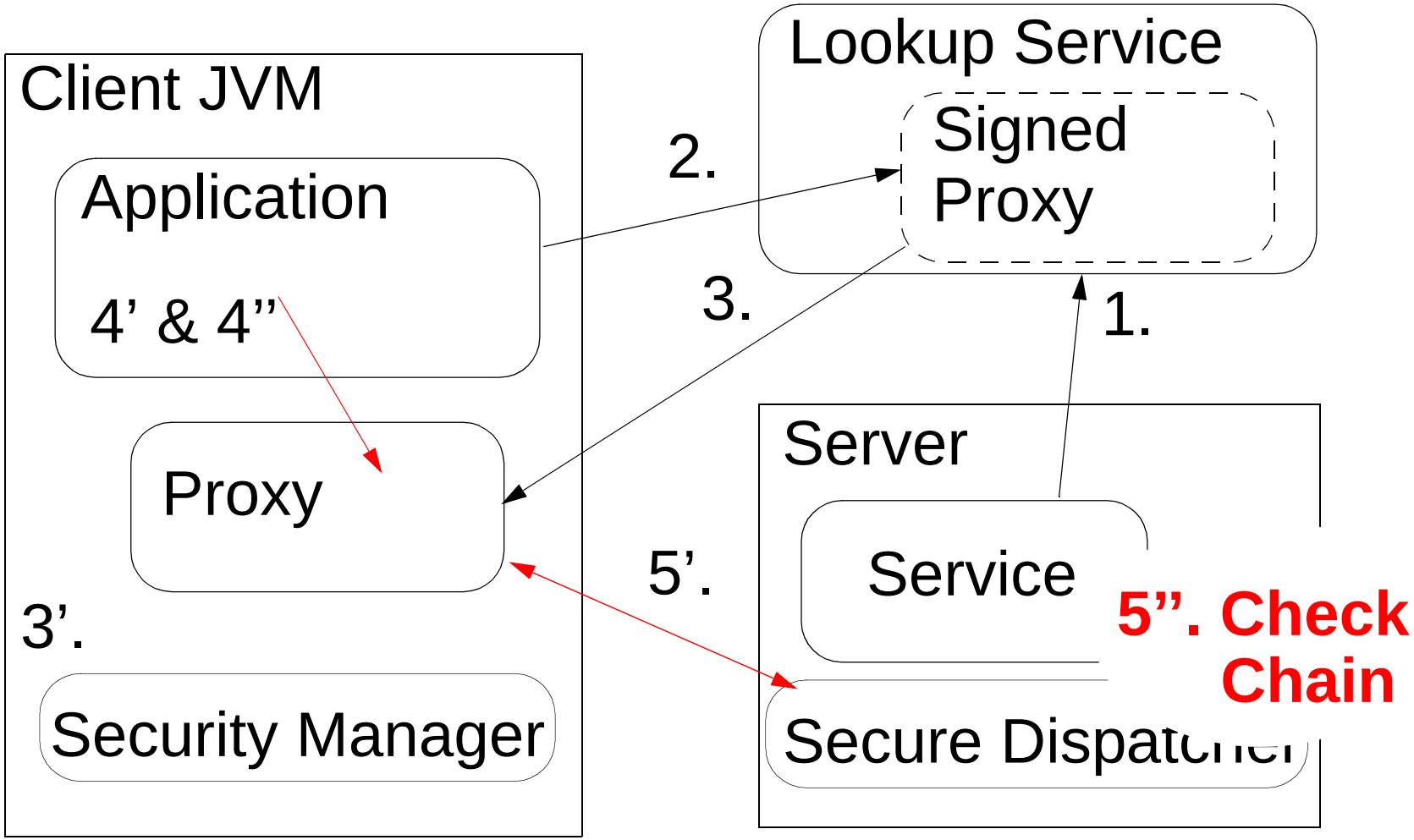
5”. The dispatcher checks certificate chains

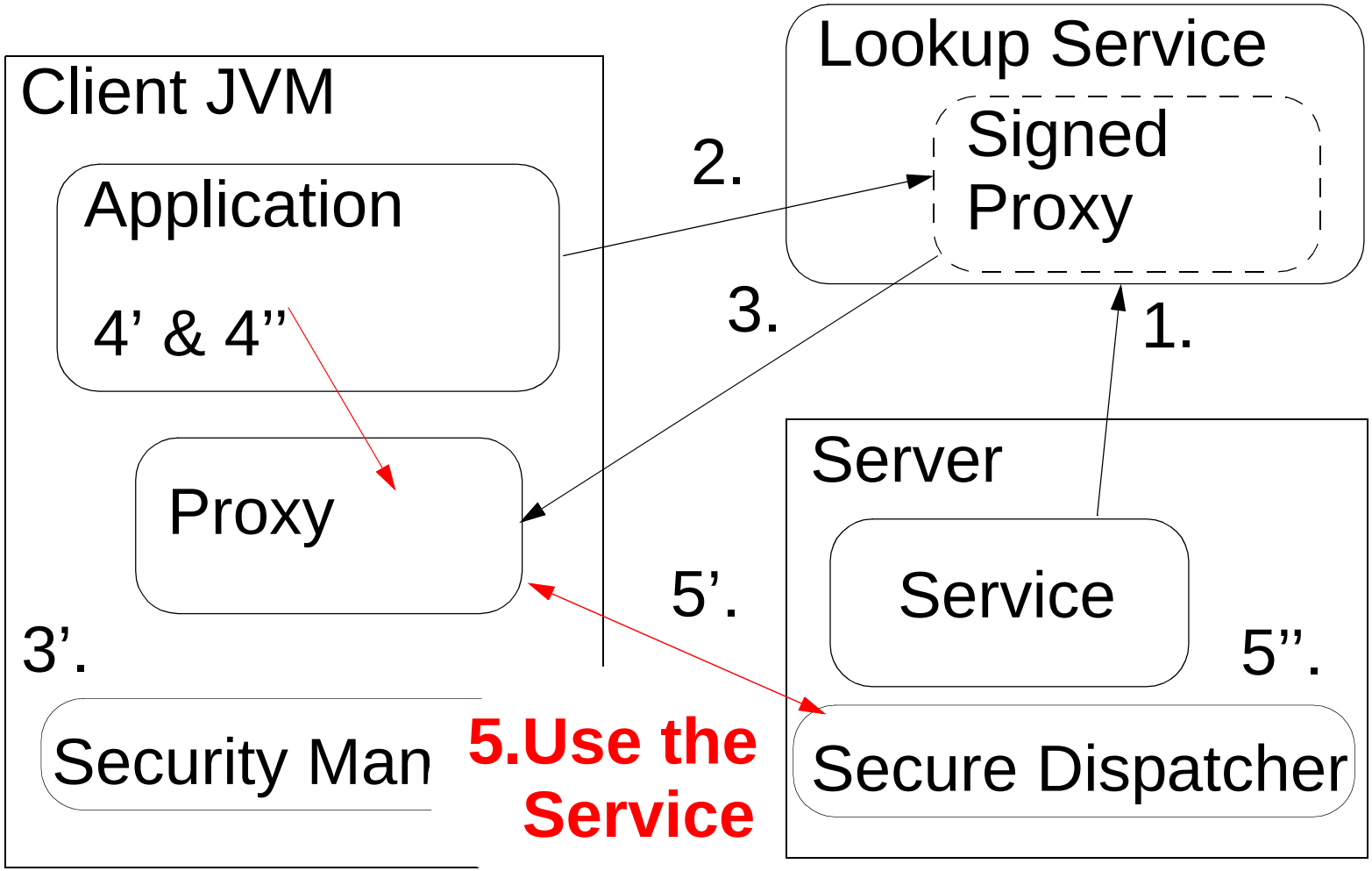

Client JVM
Application
Proxy
Security Manager
2.Look up service
3.Download Proxy
Lookup Service
Signed Proxy
1.Register signed proxy
Server
Service
Secure Dispatcher


Client JVM
Application
Proxy
Security Manager
Lookup Service
Signed
Proxy
2.
3.
1.
Server
Service
Secure Dispatcher
3'. Authenticate the Proxy


Client JVM
Application
4’ & 4”. Delegation
Application > Proxy
Proxy
3’.
Security Manager
2.
3.
Lookup Service
Signed
Proxy
1.
Server
Service
Secure Dispatcher


Client JVM
Application
4' & 4''
Proxy
3'.
Security Manager
2.
3.
Lookup Service
Signed
Proxy
1.
Server
5'.Authenticate
the Proxy
Secure Dispatcher


Client JVM
Application
4' & 4''
Proxy
3'.
Security Manager
Lookup Service
Signed
Proxy
2.
3.
1.
Server
Service
5'.
5''. Check
Chain
Secure Dispatcher


Client JVM
Application
4' & 4''
Proxy
3'.
Security Man
Lookup Service
Signed
Proxy
2.
3.
1.
Server
Service
5'.
5''.
Secure Dispatcher
5.Use the
Service

Benefits

- Minimal implicit trust assumptions;
  can be used in different environments
- Allows application specific “authentication UI”
- Preserves most of the protocol independence
- Integrates quite well with Java 2
  security architecture
- No modifications to JDK or Jini

Limitations & Future Work

- Only public keys, not interoperable with existing authentication systems (like JAAS)
- The proof-of-concept implementation requires some mixing of application and security code
  - This deficiency could be removed by adding new Jini libraries and integrating to JAAS

Conclusions

- Centralized CAs are not the only solution to problems with downloaded code
- Trust management systems provide flexible solutions to distributed security problems
  - Identifying principals with public keys, and associating permissions directly with the keys
- More details in the form of Pasi Eronen's Master's Thesis available at http://www.iki.fi/pe/publications/

Questions and/or Comments?

- http://www.iki.fi/pe/publications/