

Identity Management Basics

Part 1 of Identity Management with Progress OpenEdge

Peter Judge
OpenEdge Development
pjudge@progress.com



What Is Identity Management?

- Identity management is all about trust relationships
- It's about protecting your business data
- **You** make security decisions on behalf of your customers ... understand the maximum loss **they** might suffer

This Is Nothing New

- Forces aligned against you are more prevalent, and they have more, and more sophisticated weapons
- And you've given people a door and invitation via the internet
- So now the things you used to do are no longer adequate

What Is Identity Management?

It's about protecting your business data by

- Controlling and verifying who accesses your data **AUTHENTICATION**
- Controlling what they can do with your data Authorization
- Reviewing what they did with your data Auditing
- Maintaining information about your users Administration

Authorization and Auditing

■ Authorization

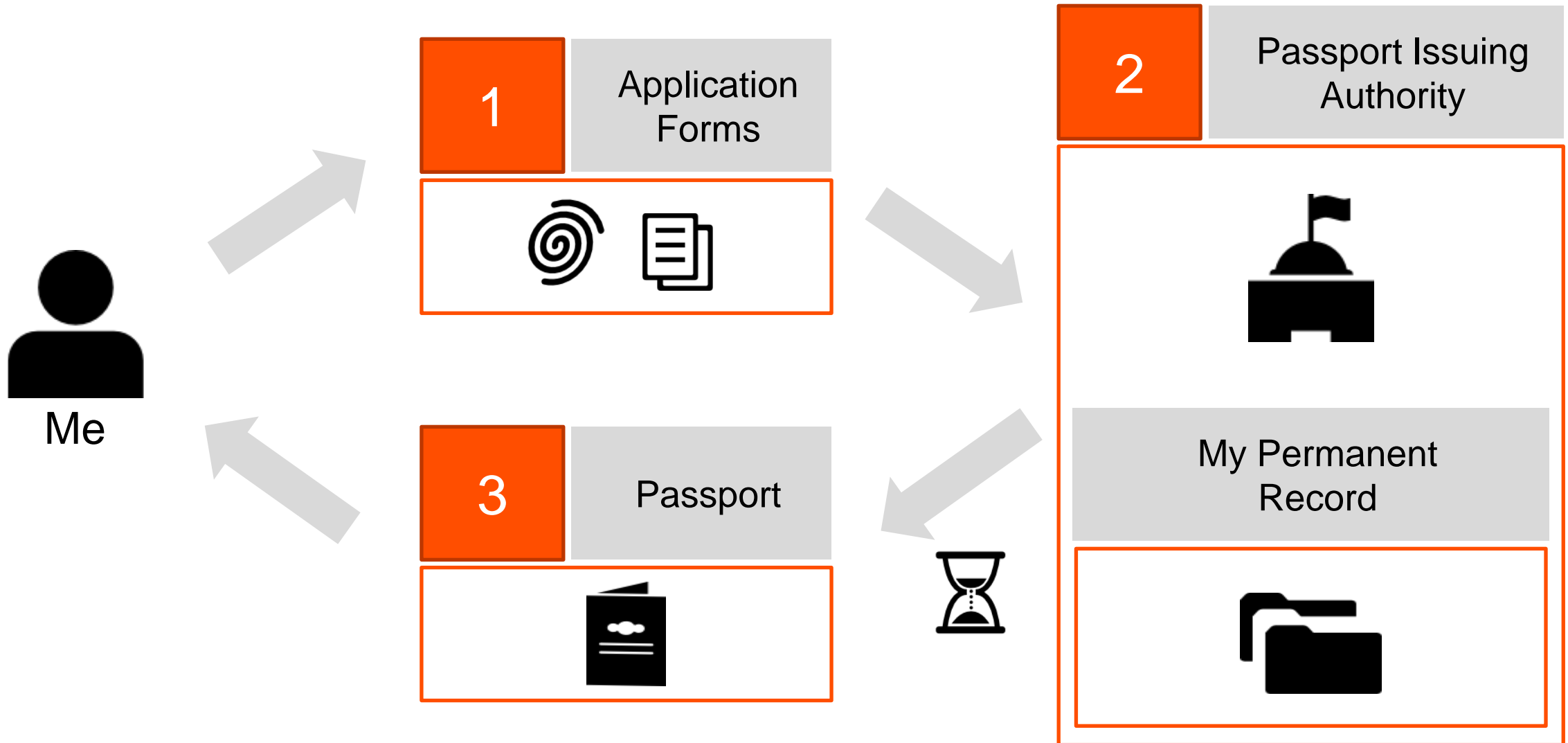
- What services can the user access?
- What data can the user see and/or modify?
 - Multi-tenancy
 - Record-level, field-level

■ Auditing

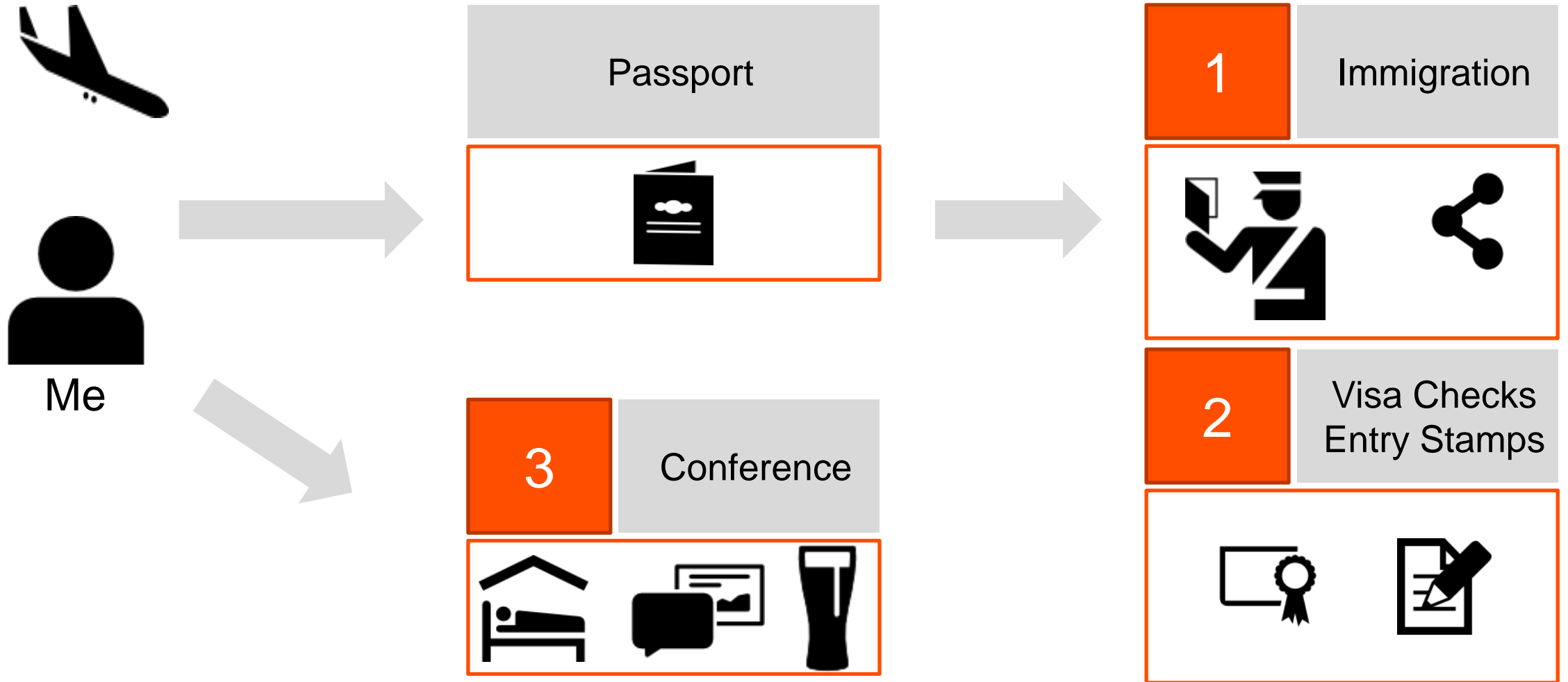
- Verifiable trace of a user's actions



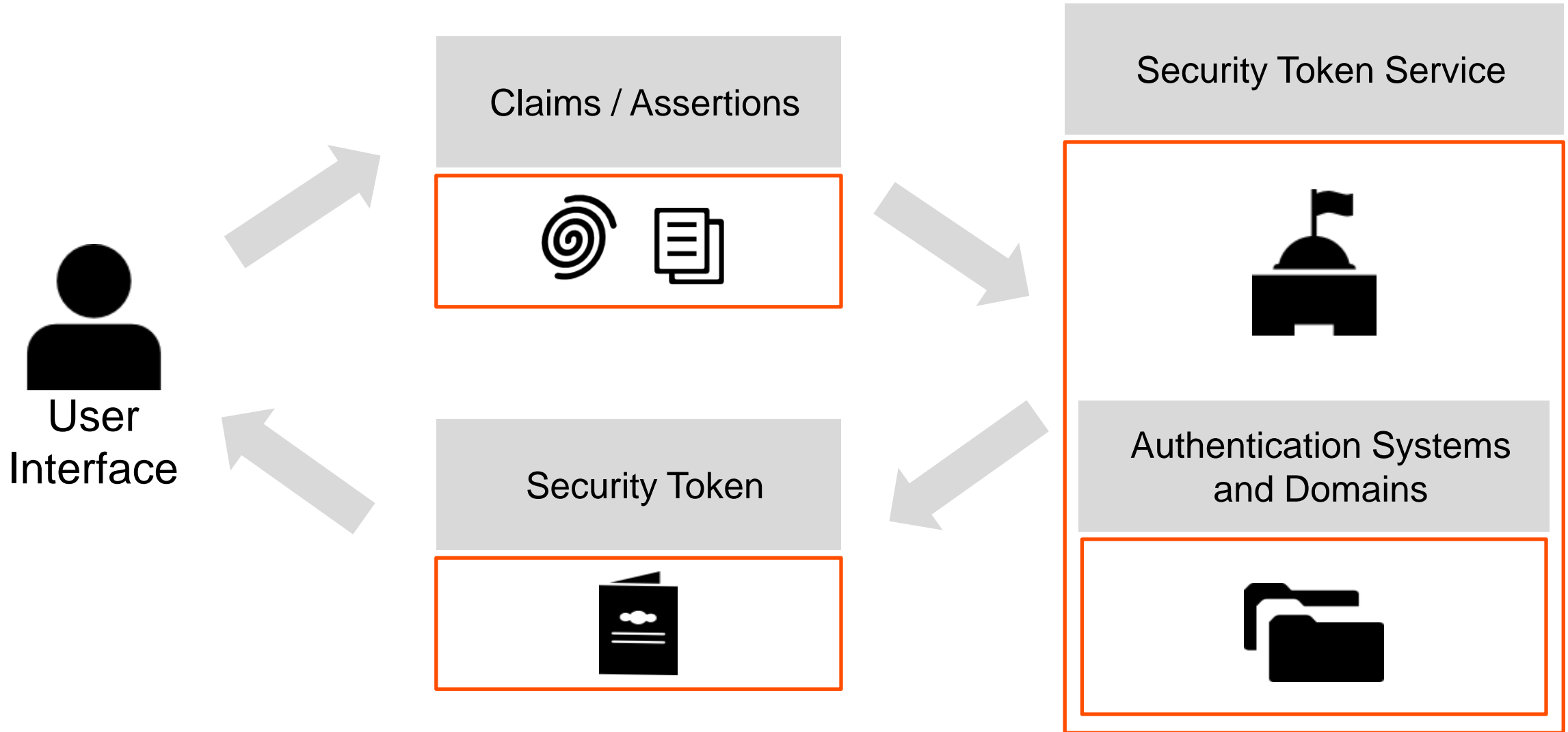
Getting a Passport



Using a Passport



Application Flow: Login



What Is a Security Token?

- A transportable block of data that can be used as proof of user identity by any systems or applications that have a trust relationship with the originator of the security token
 - Exists for same reason passports do: so that a gatekeeper doesn't have to ask you for everything every time you want to pass
- Enables Single Sign On (SSO)
 - Authenticate once and allow access many times across (ABL) processes
- Secure, time sensitive and data-integrity protected

The ABL CLIENT-PRINCIPAL

- CLIENT-PRINCIPAL = ABL security token
- Sets current identity in any connected db or AVM session
- AVM creates if not created explicitly
- Manage a user's login session

```
CREATE CLIENT-PRINCIPAL hCP.  
hCP:INITIALIZE(<args>)
```

```
SECURITY-POLICY:SET-CLIENT(hCP).  
SET-DB-CLIENT(<dbname>, hCP).
```

```
SETUSERID(<userid>, <pass>, <dbname>).  
cmd> $PROEXE -U <userid> -P <pass>
```


```
hCP = SECURITY-POLICY:GET-CLIENT().  
rCP = hCP:EXPORT-PRINCIPAL.  
hCP:LOGOUT().
```

OE 10.1A+

What Are Domains?

- A group of users with a common set of
 - Roles and responsibilities
 - Level of security
 - Data access privileges
- Configured in db meta-schema

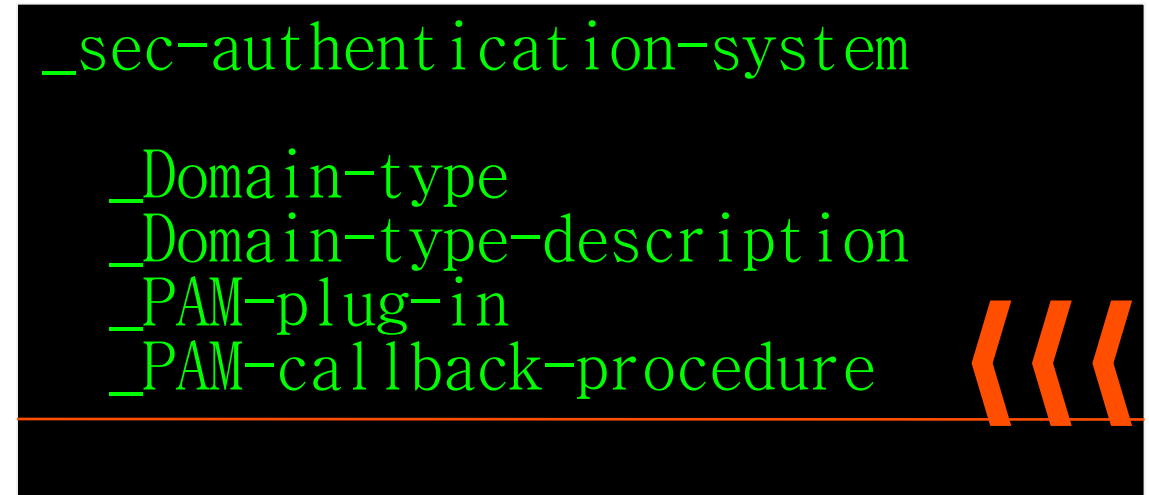
```
_sec-authentication-domain
  _Domain-name
  _Domain-type
  _Domain-description
  Domain-access-code
  _Domain-runtime-options
  _Tenant-name
  _Domain-enabled
```



Authentication Systems (aka Plug-ins)

- Validates requesting entity's claims
 - Full user login (i.e. user authentication), or
 - Single Sign-On (SSO)
- Specifies actual means of performing authentication
 - ABL callbacks available for user-defined systems
- Single authentication system can support multiple domains
 - One domain has one authentication system

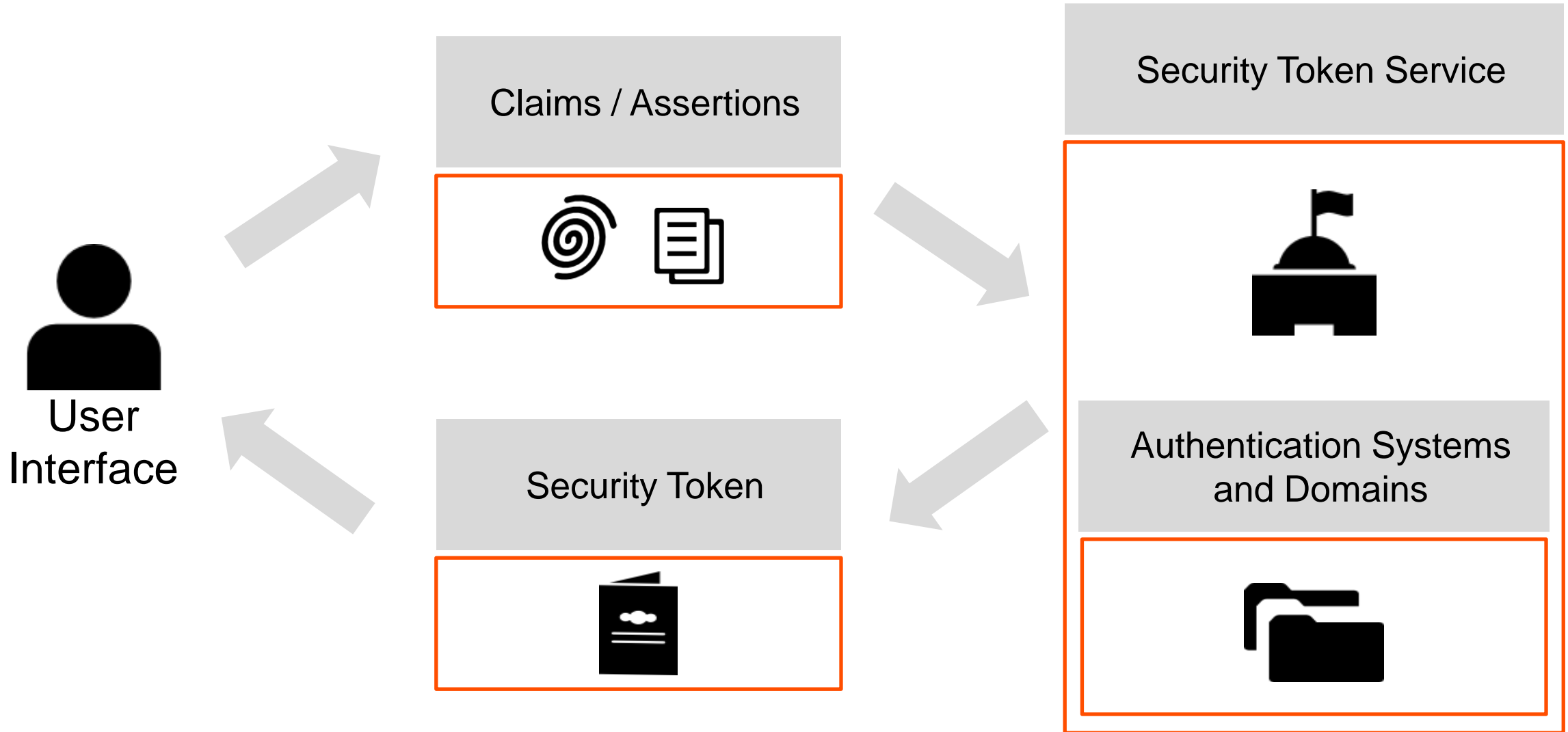
OE 11.1+



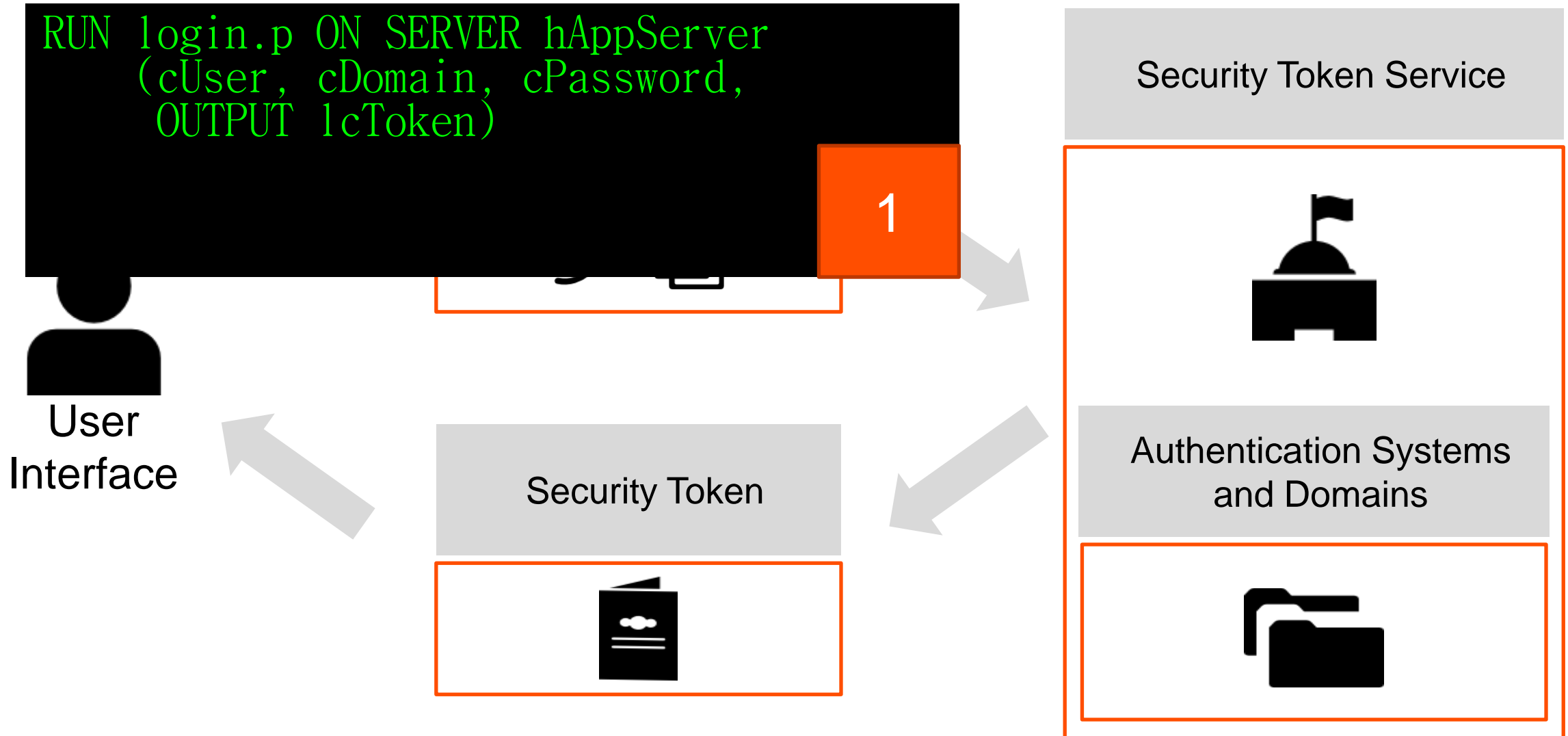
User Credentials Example Schema

```
ADD TABLE "ApplicationUser"  
  AREA "Data"  
  DESCRIPTION "The application's user table. Contains login names, passwords and  
mappings to login domains."  
  DUMP-NAME "applicationuser"  
  
ADD FIELD "LoginName" AS character  
/* Domain necessary for re-use */  
ADD FIELD "LoginDomain" AS character  
ADD FIELD "Password" AS character  
ADD FIELD "LastLoginDate" AS datetime-tz  
/* Last login IP address / host */  
ADD FIELD "LastLoginLocation" AS character  
  
ADD INDEX "Login" ON "ApplicationUser"  
  AREA "Indexes"  
  UNIQUE  
  INDEX-FIELD "LoginName" ASCENDING  
  INDEX-FIELD "LoginDomain" ASCENDING
```

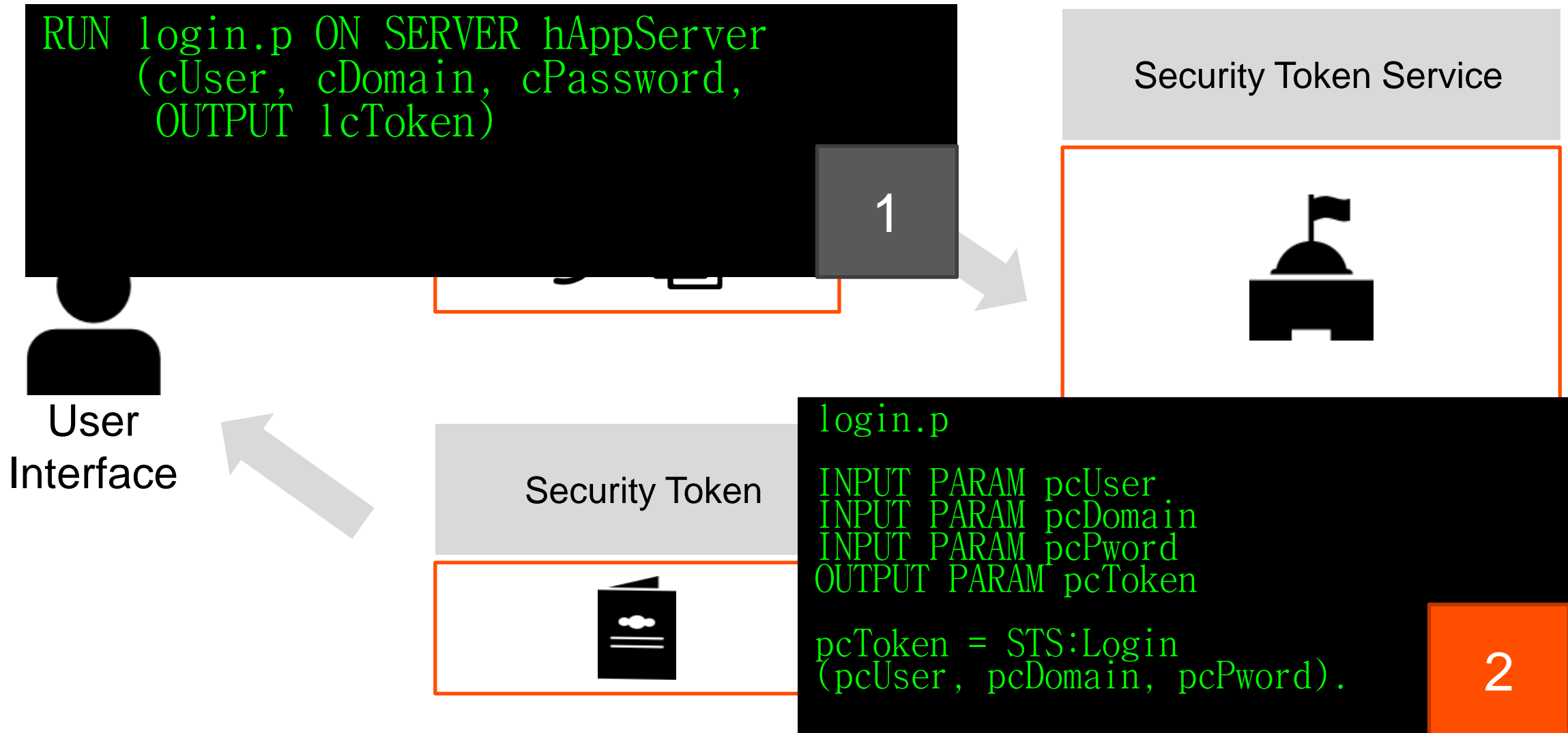
Application Flow: Login



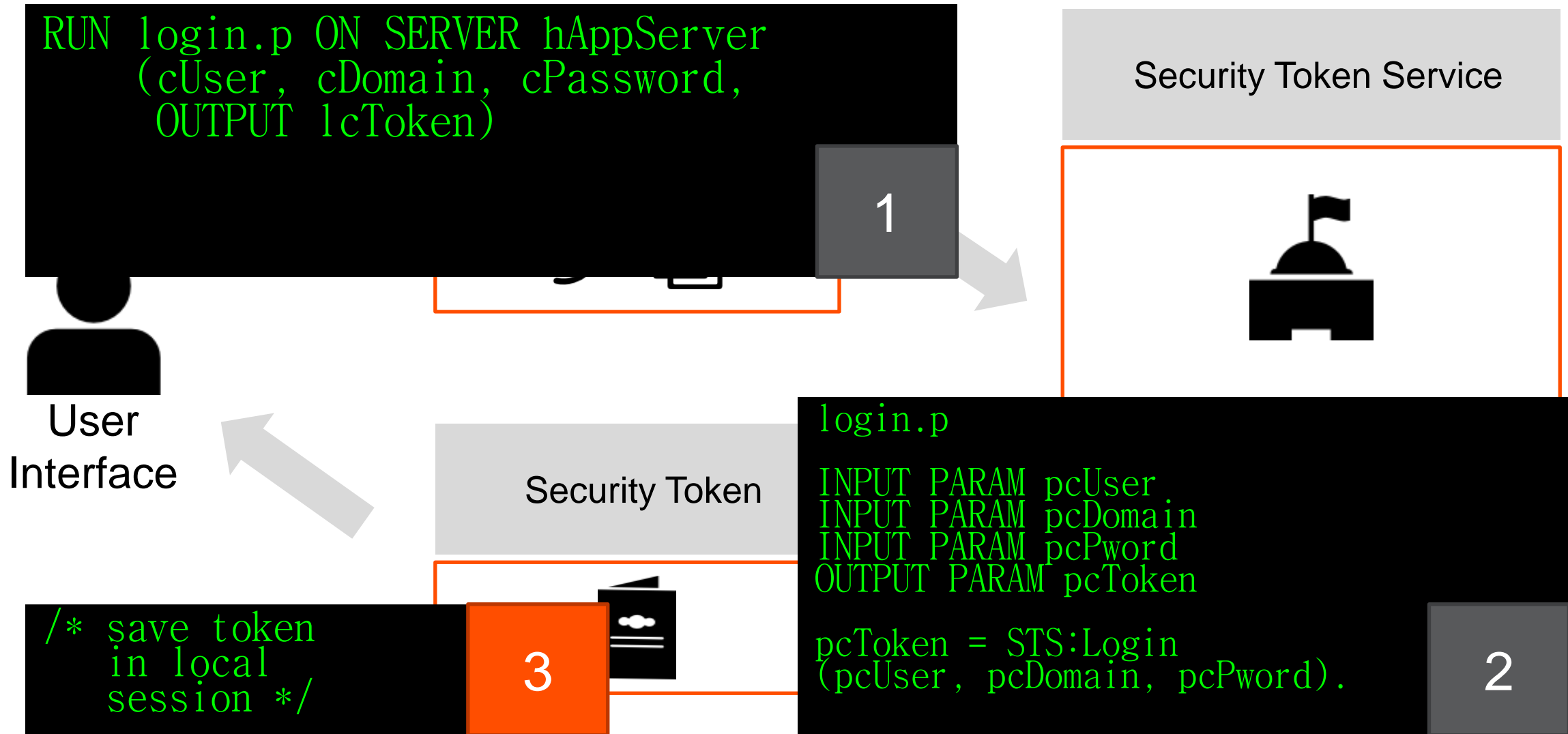
Application Flow: Login



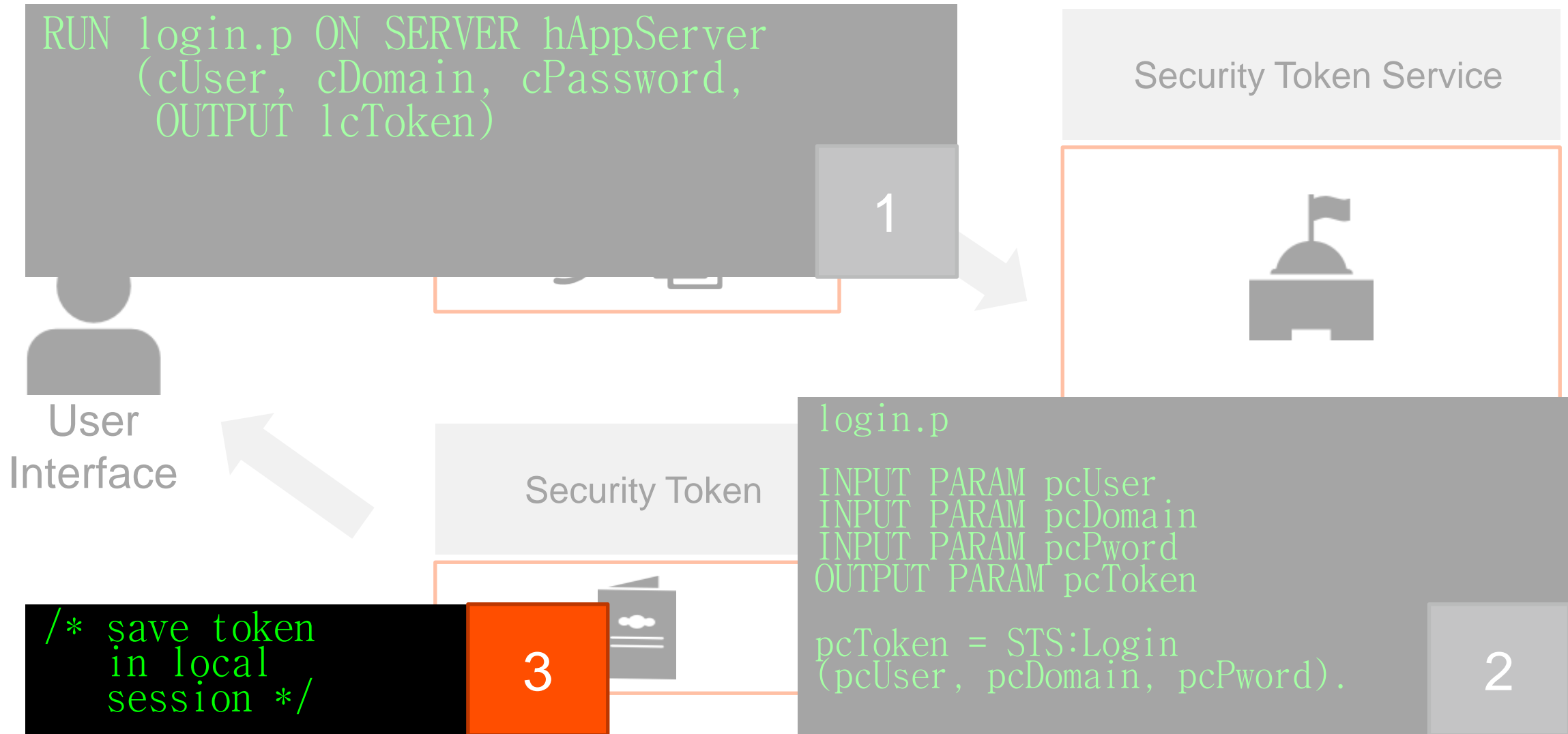
Application Flow: Login



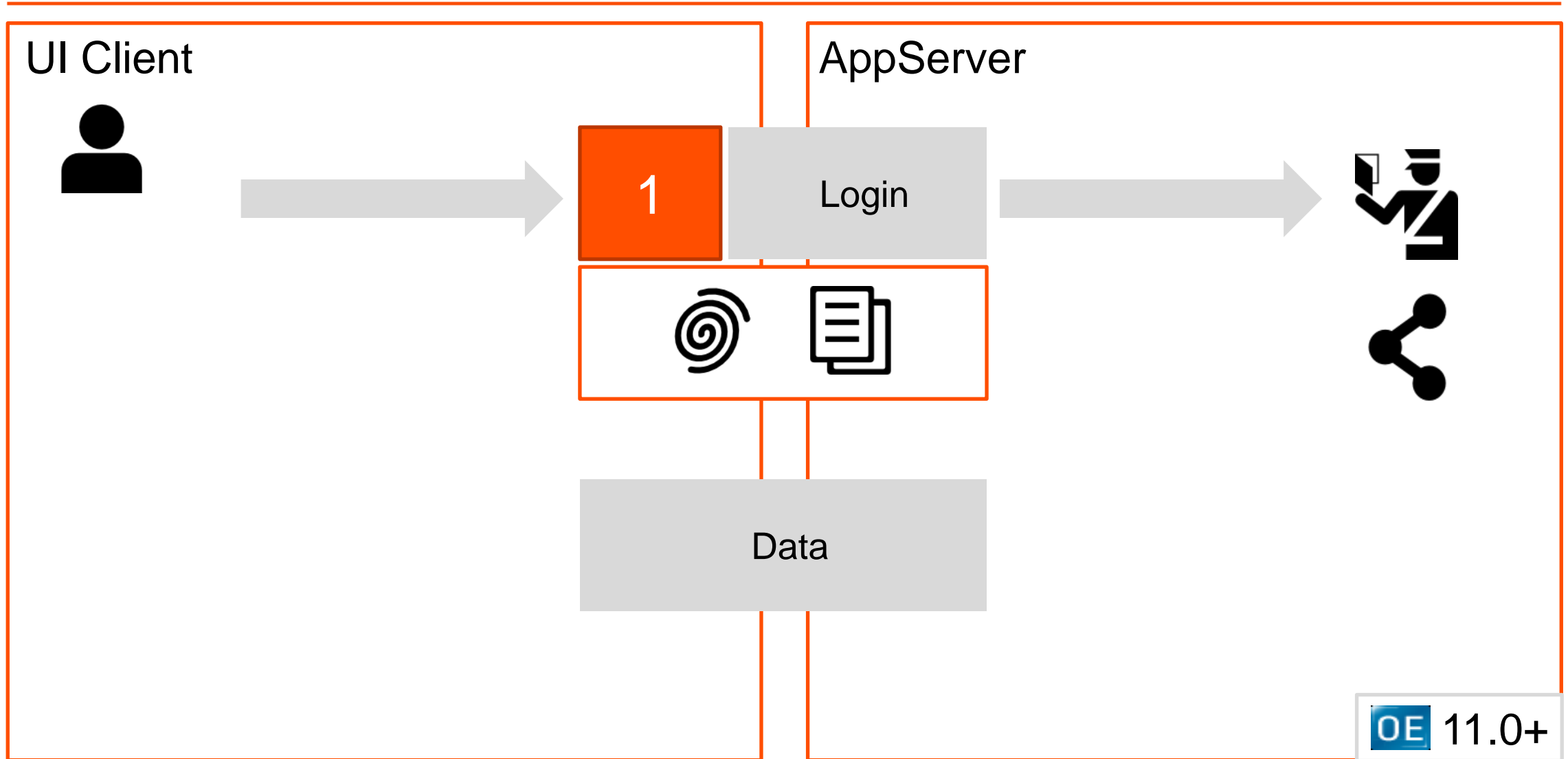
Application Flow: Login



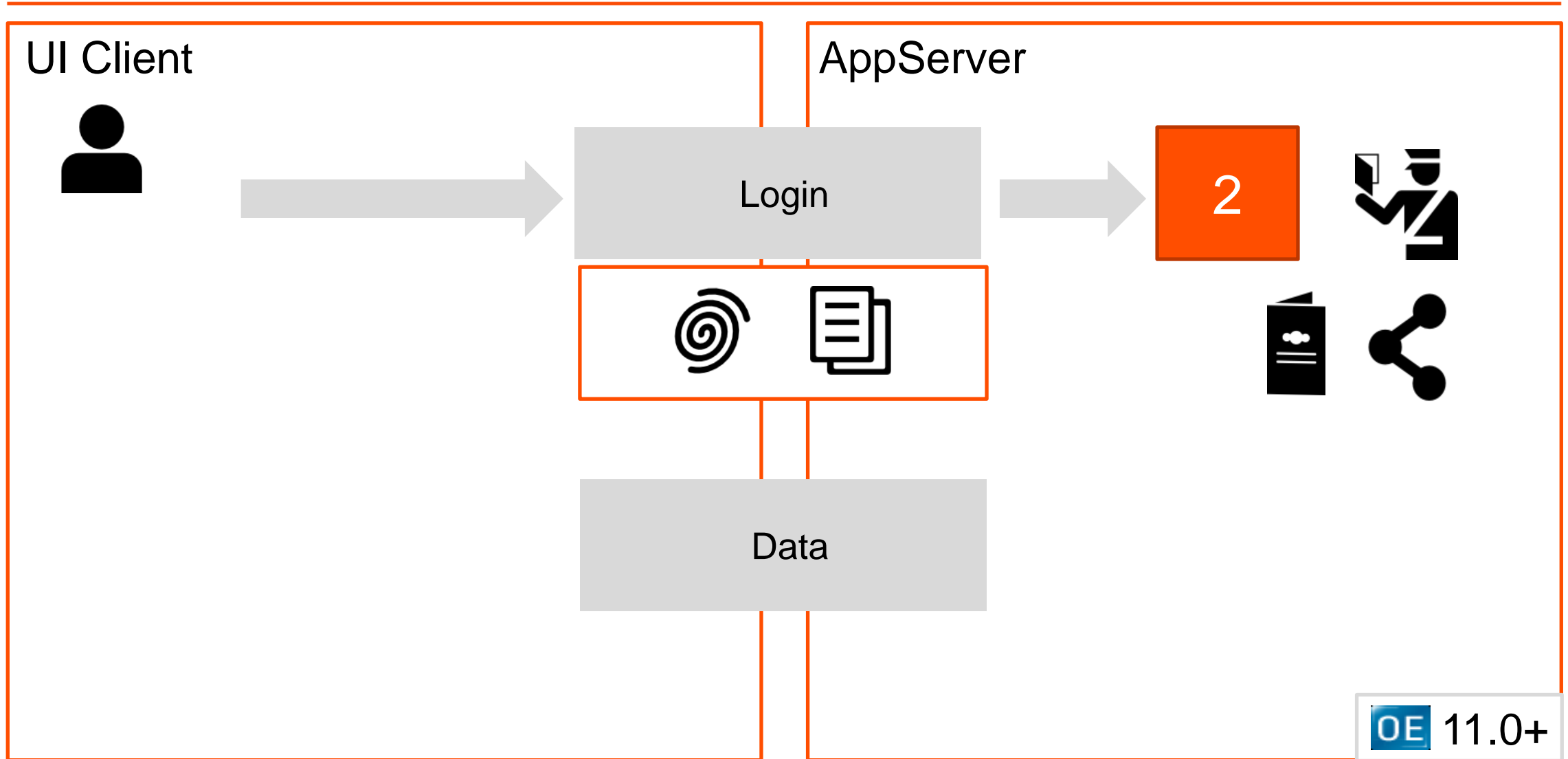
Application Flow: Login



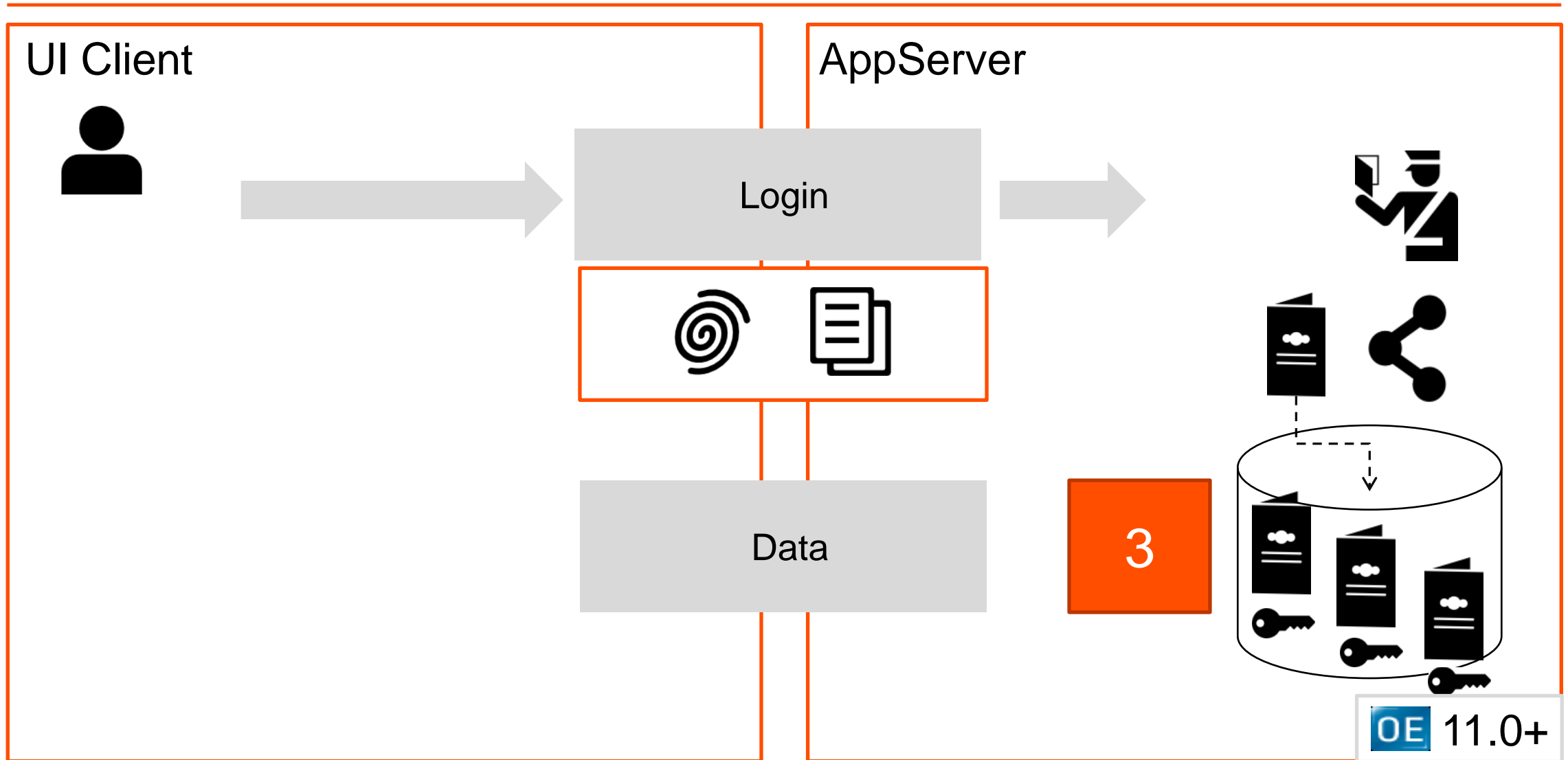
Managing Security Context: Server



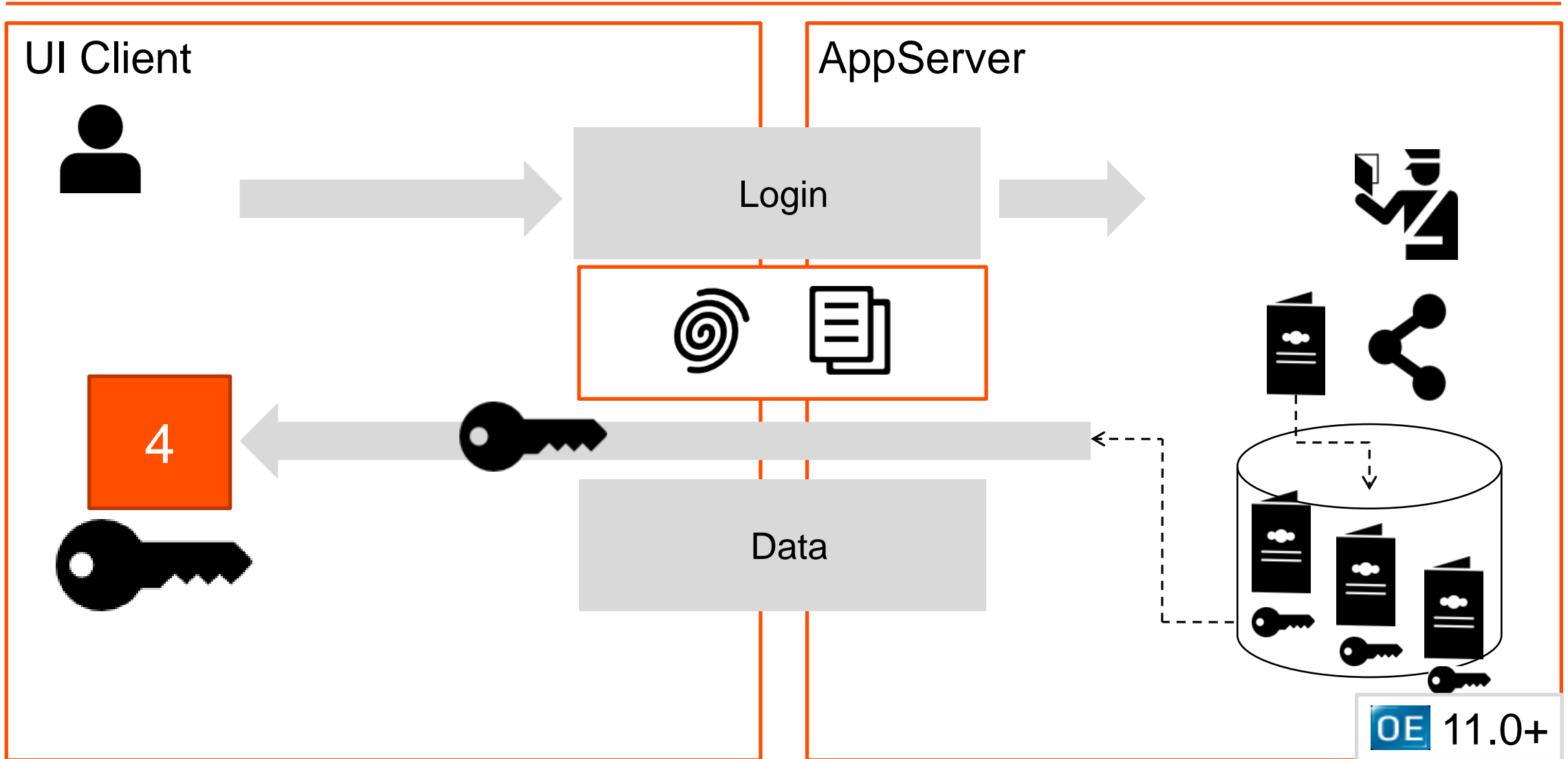
Managing Security Context: Server



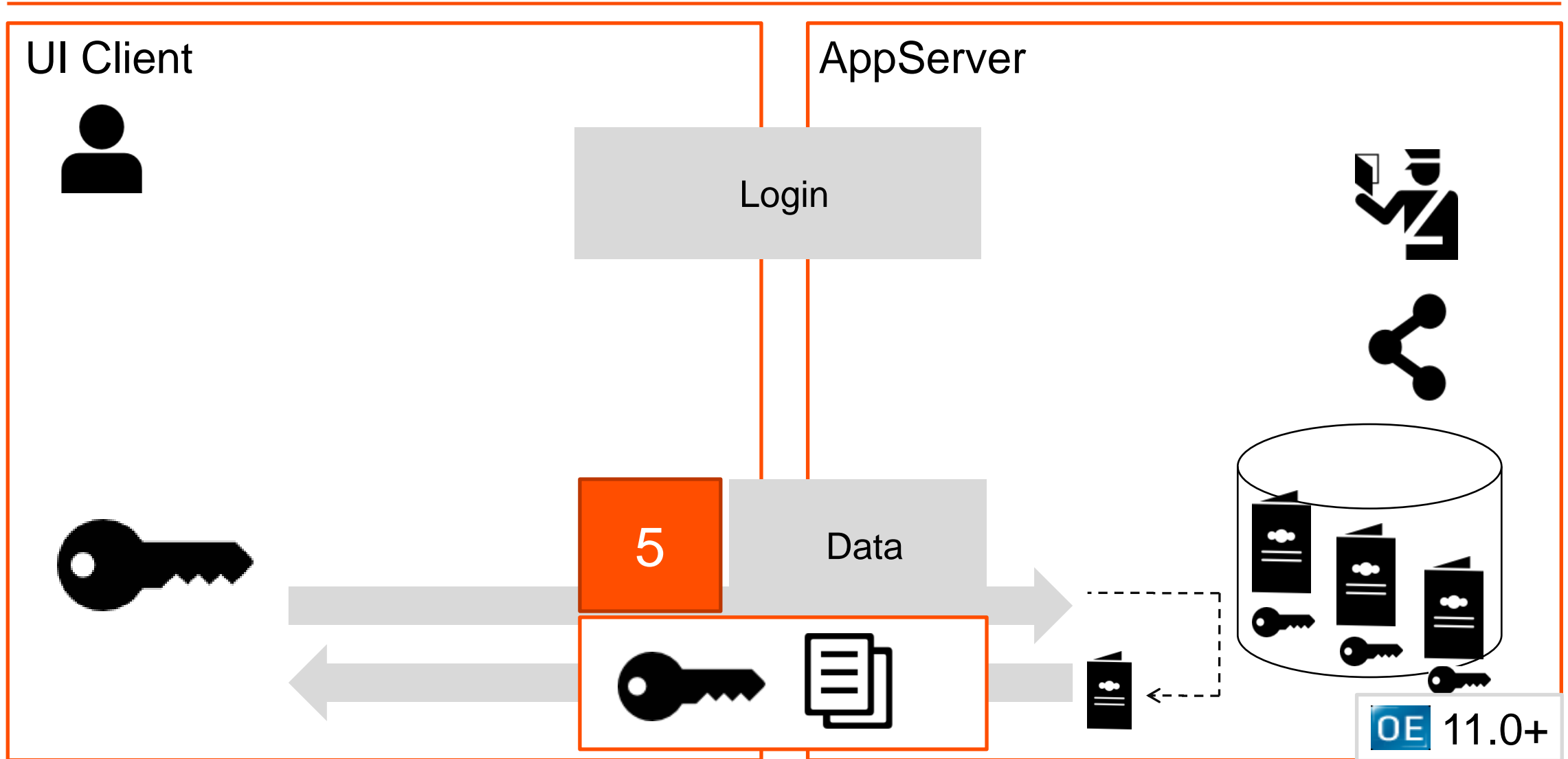
Managing Security Context: Server



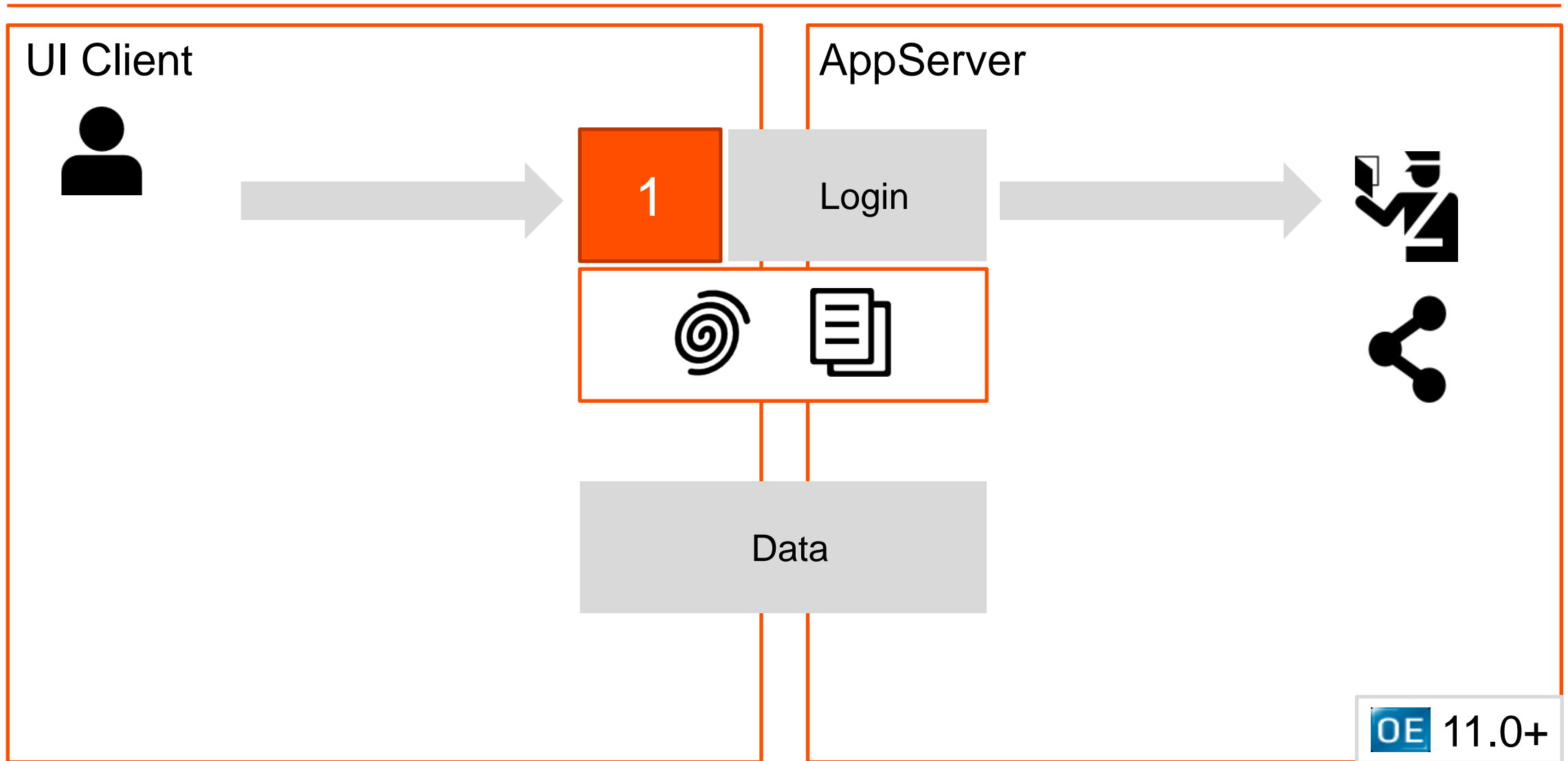
Managing Security Context: Server



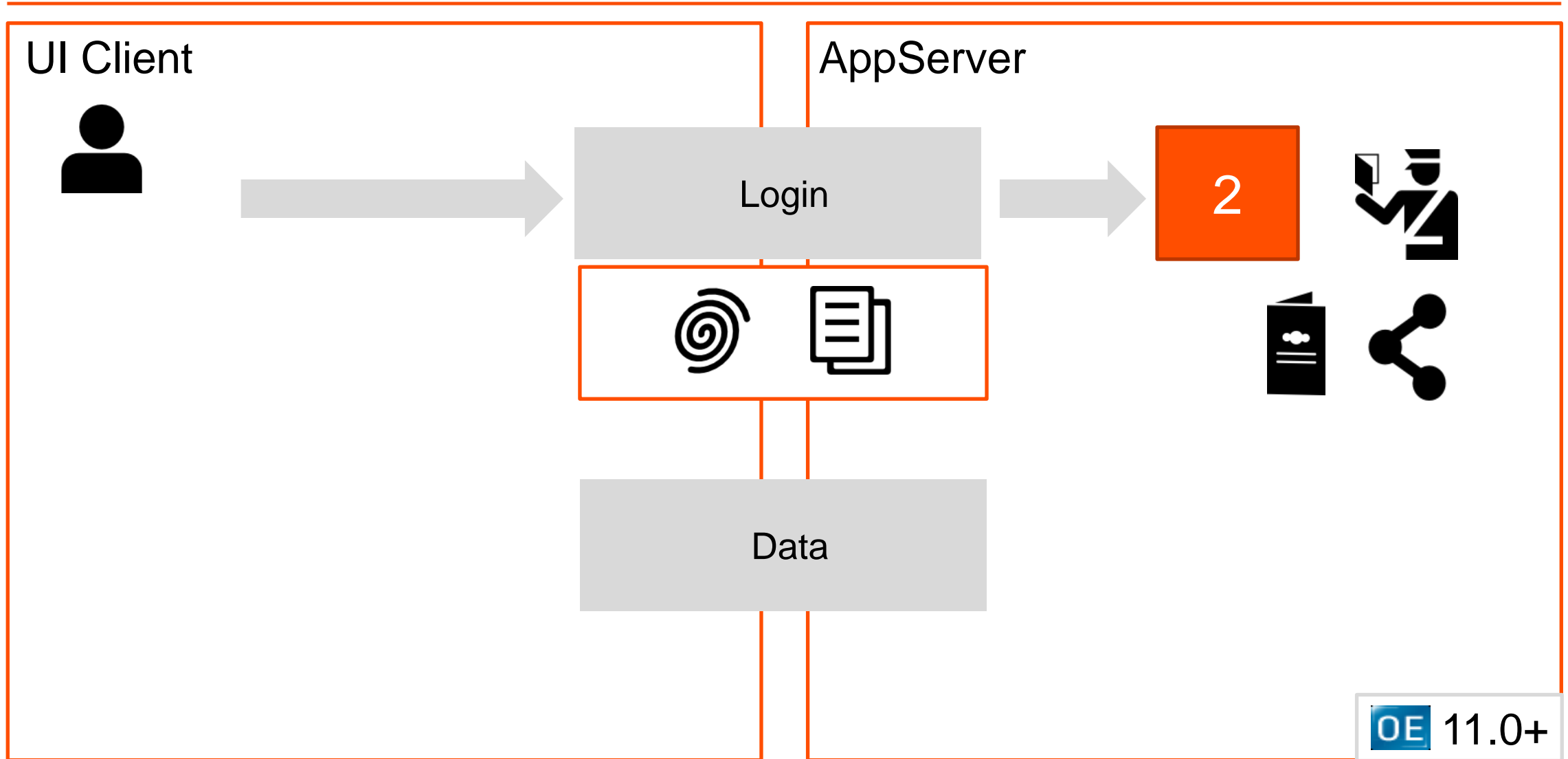
Managing Security Context: Server



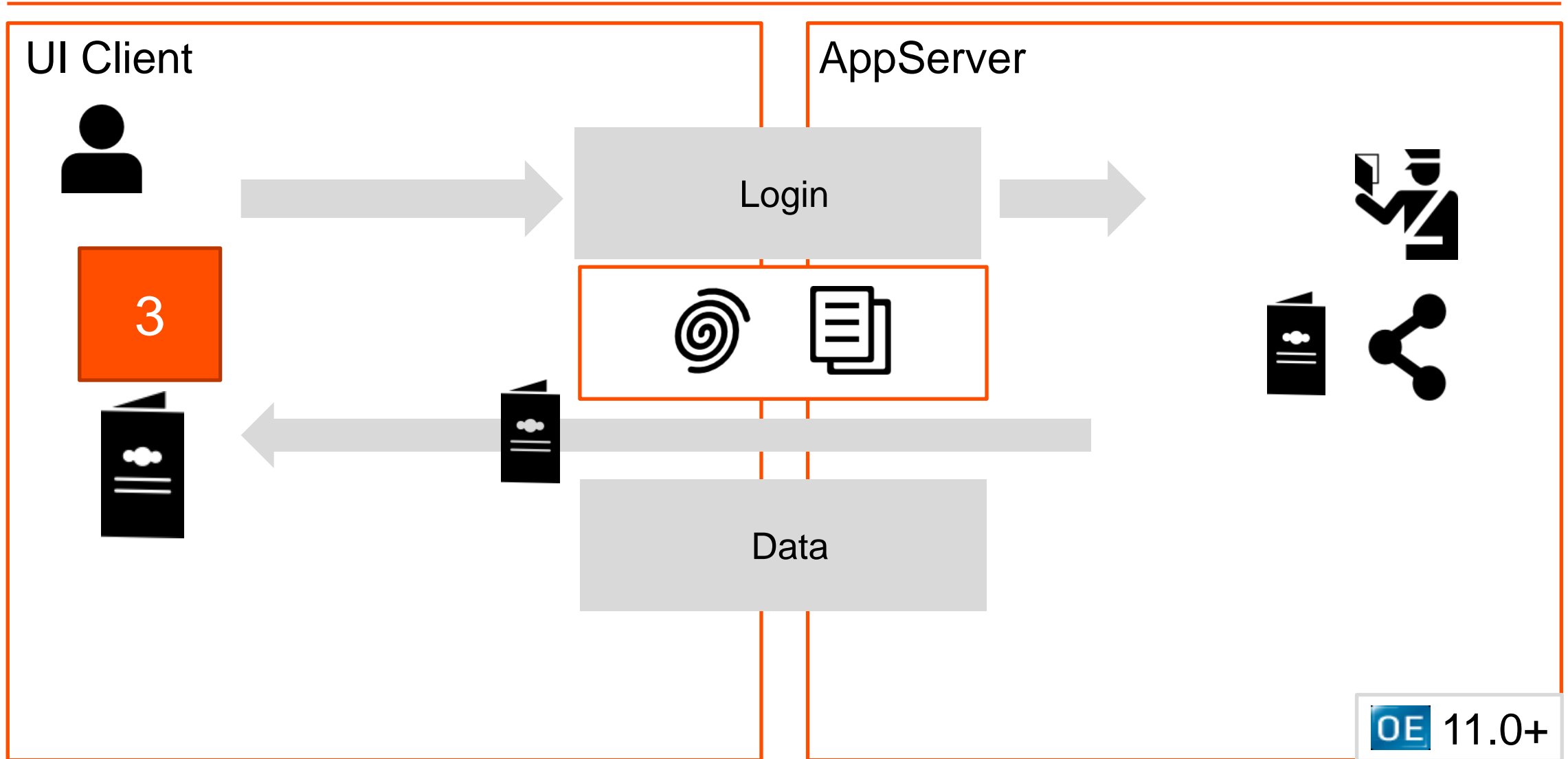
Managing Security Context: Client



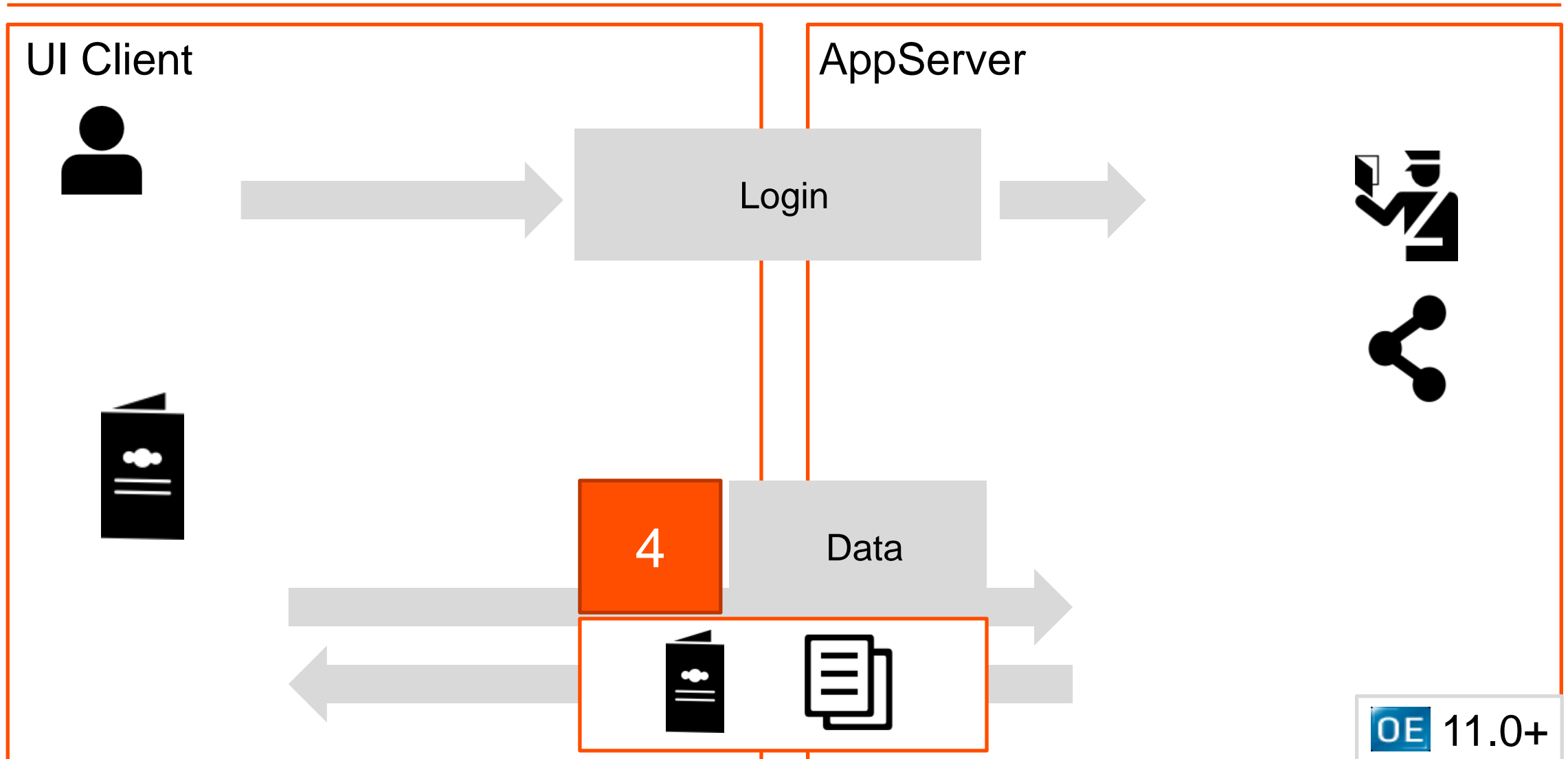
Managing Security Context: Client



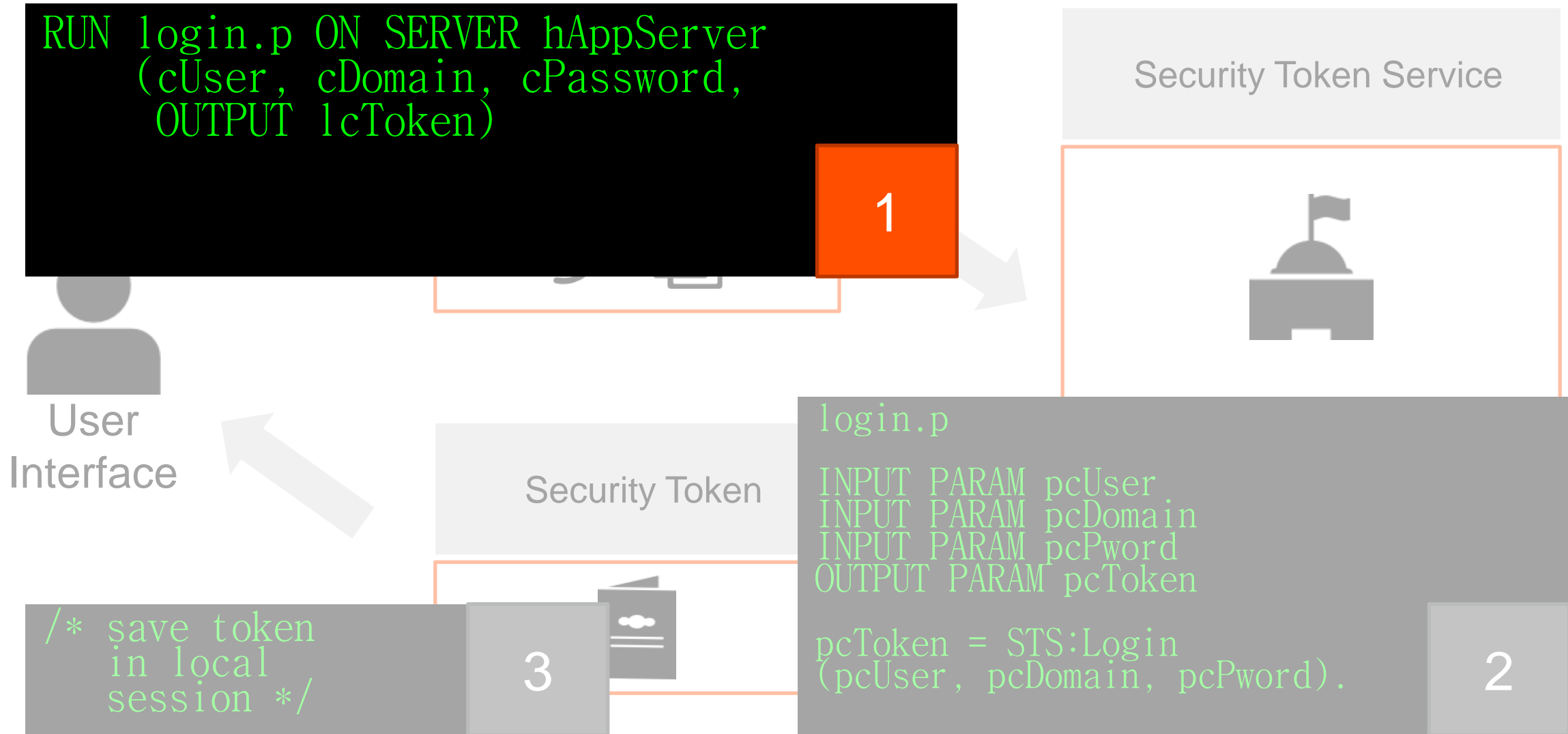
Managing Security Context: Client



Managing Security Context: Client




Application Flow: Login

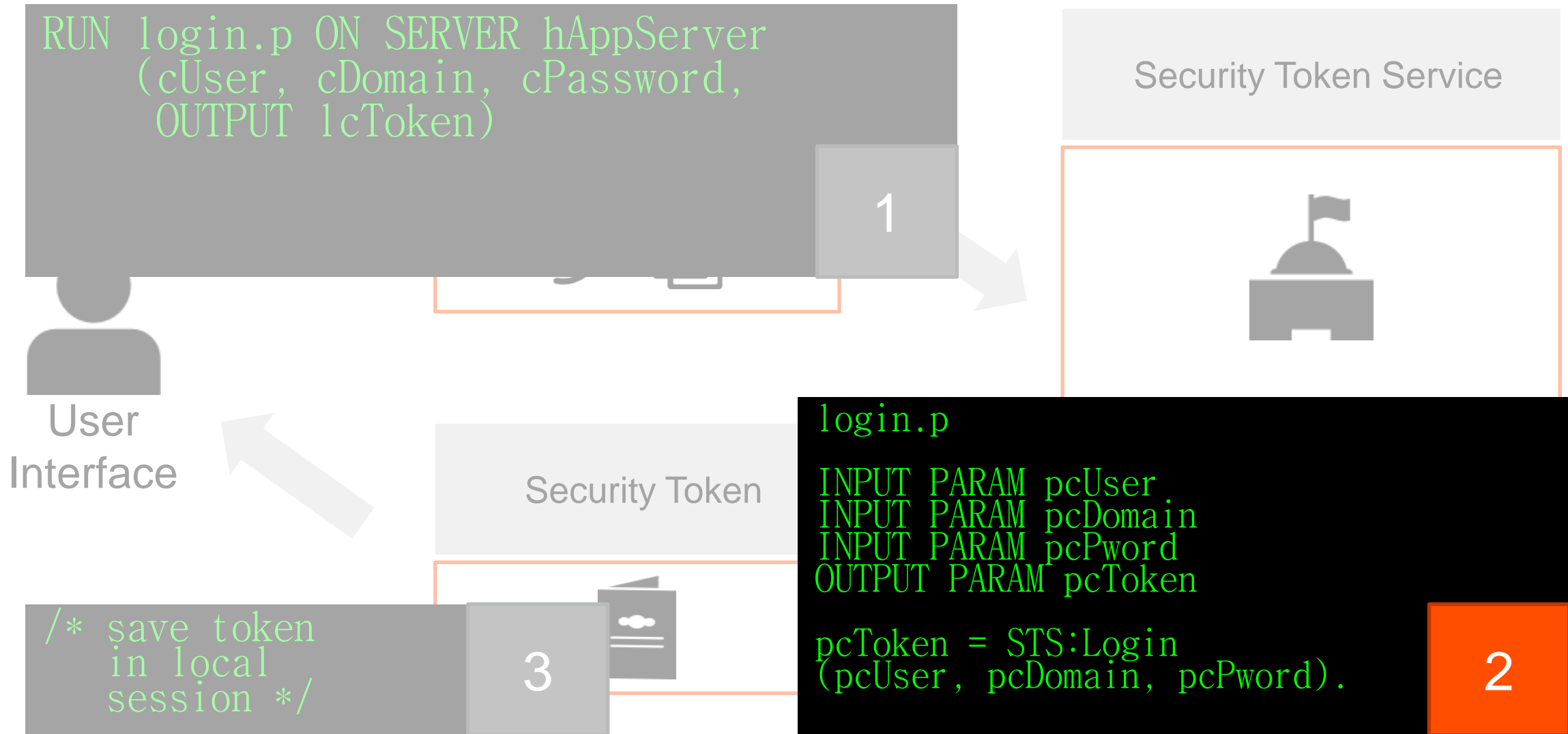


Desktop.MainForm.cls

```
method public logical LoginUser(  
    input pcUserName as char,  
    input pcDomain as char,  
    input pcPassword as char):  
  
    run Security/Login.p on hAppServer (  
        pcUserName, pcDomain, pcPassword,  
        output cUserContextId).  
    if cUserContextId eq '' then return false.  
  
    /* set the CCID on the business logic server so that it's  
       transported with every request. */  
    hAppServer:request-info:ClientContextId = cUserContextId.  
  
    return true.  
end method.
```



Application Flow: Login



Security/Login.p

```
define input    parameter pcUser as character no-undo.  
define input    parameter pcDomain as character no-undo.  
define input    parameter pcPassword as character no-undo.  
define output   parameter pcToken as character no-undo.
```

```
pcToken = Security.SecurityTokenService:Instance  
          :LoginUser(pcUser, pcDomain, pcPassword).
```



Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
    substitute('&1@&2', pcUserName, pcUserDomain),
    ?, /* unique session id */
    add-interval(now, 8, 'hours'), /* login expiration */
    pcPassword).

  /* passes authentication work off to authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```



Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
    substitute('&1@&2', pcUserName, pcUserDomain),
    ?, /* unique session id */
    add-interval(now, 8, 'hours'), /* login expiration */
    pcPassword).

  /* passes authentication work off to authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```



_sec-authentication-system

```
create _sec-authentication-system.  
_Domain-type          = 'TABLE-ApplicationUser'.  
_Domain-type-description =  
    'The ApplicationUser table serves as  
    the authentication domain'.  
_PAM-plugin           = true.  
  
_PAM-callback-procedure =  
    'Security/AppUserAuthenticate.p'.
```



Security/AppUserAuthenticate.p

```
procedure AuthenticateUser:
  def input  param phClientPrincipal  as handle no-undo.
  def input  param pcSystemOptions as character extent no-undo.
  def output param piPAMStatus as integer init ? no-undo.
  def output param pcErrorMsg as character no-undo.

  find ApplicationUser where
    ApplicationUser.LoginName eq phCP:user-id and
    ApplicationUser.LoginDomain eq phCP:domain-name
    no-lock no-error.

  if not available ApplicationUser then
    piPAMStatus = Progress.Lang.PAMStatus:UnknownUser.
  else
    if ApplicationUser.Password ne
      encode(phCP:primary-passphrase) then
      piPAMStatus = Progress.Lang.PAMStatus:AuthenticationFailed.
    else
      /* we're good to go */
      piPAMStatus = Progress.Lang.PAMStatus:Success.

  return.
end procedure.
```



Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
    substitute('&1@&2', pcUserName, pcUserDomain),
    ?, /* unique session id */
    add-interval(now, 8, 'hours'), /* login expiration */
    pcPassword).

  /* passes authentication work off to authentication system */
  security-policy:set-client(hClientPrincipal).


  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```



Security.SecurityTokenService.cls

```
method protected void WriteClientPrincipalToStore(  
    input phClientPrincipal as handle):  
    define buffer lbSecurityContext for SecurityContext.  
  
    find lbSecurityContext where  
        lbSecurityContext.SessionId eq phClientPrincipal:session-id  
        exclusive-lock no-wait no-error.  
    if not available lbSecurityContext then  
        do:  
            create lbSecurityContext.  
            lbSecurityContext.SessionId = phClientPrincipal:session-id.  
        end.  
        lbSecurityContext.ClientPrincipal =  
            phClientPrincipal:export-principal().  
        lbSecurityContext.LastAccess = now.  
    end method.
```



Security.SecurityTokenService.cls

```
method public char LoginUser(input pcUserName as char,
                             input pcUserDomain as char,
                             input pcPassword as char):
  define variable hClientPrincipal as handle no-undo.

  create client-principal hClientPrincipal.
  hClientPrincipal:initialize(
    substitute('&1@&2', pcUserName, pcUserDomain),
    ?, /* unique session id */
    add-interval(now, 8, 'hours'), /* login expiration */
    pcPassword).

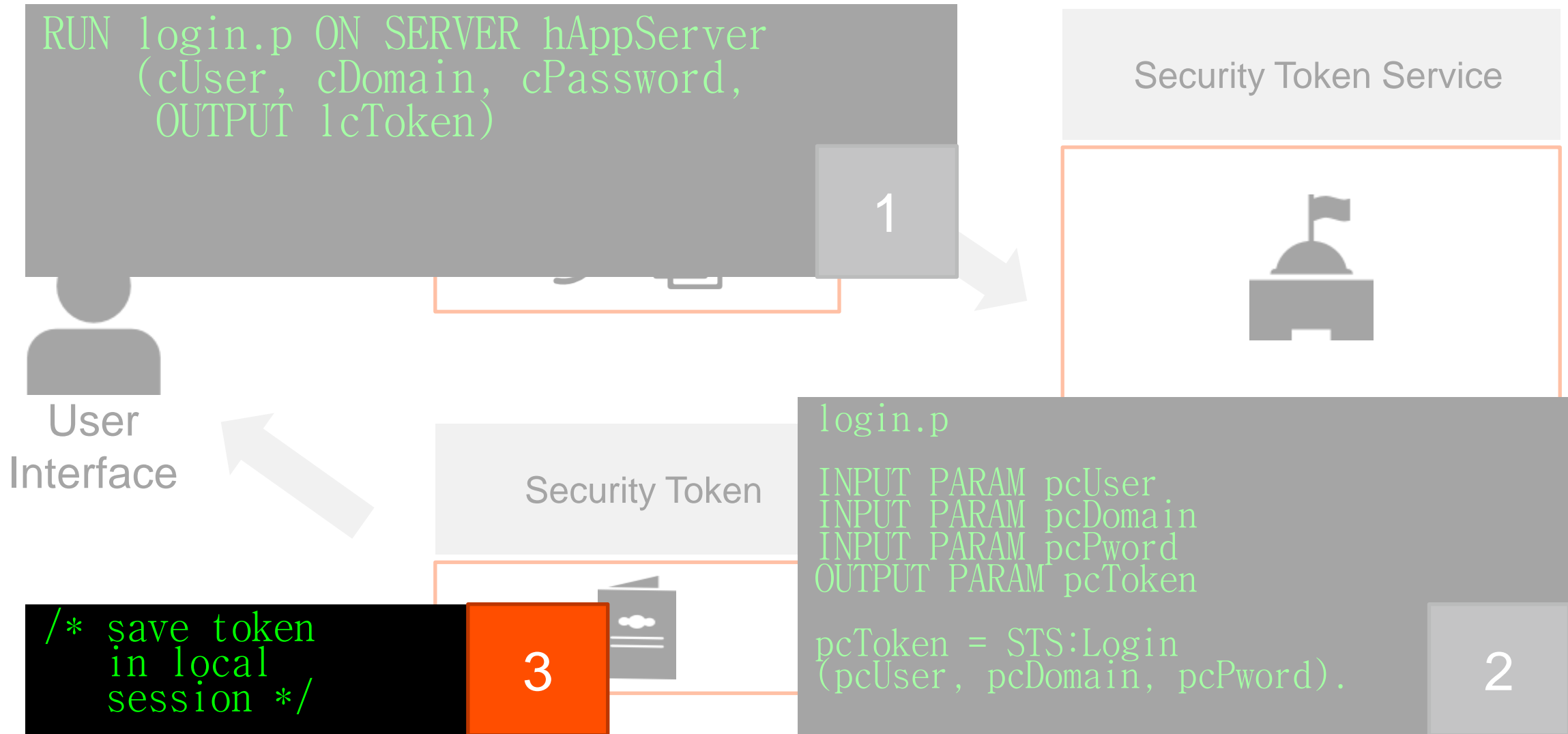
  /* passes authentication work off to authentication system */
  security-policy:set-client(hClientPrincipal).

  /* writes security context into DB */
  WriteClientPrincipalToStore(hClientPrincipal).

  /* return character value */
  return hClientPrincipal:session-id.
end method.
```



Application Flow: Login

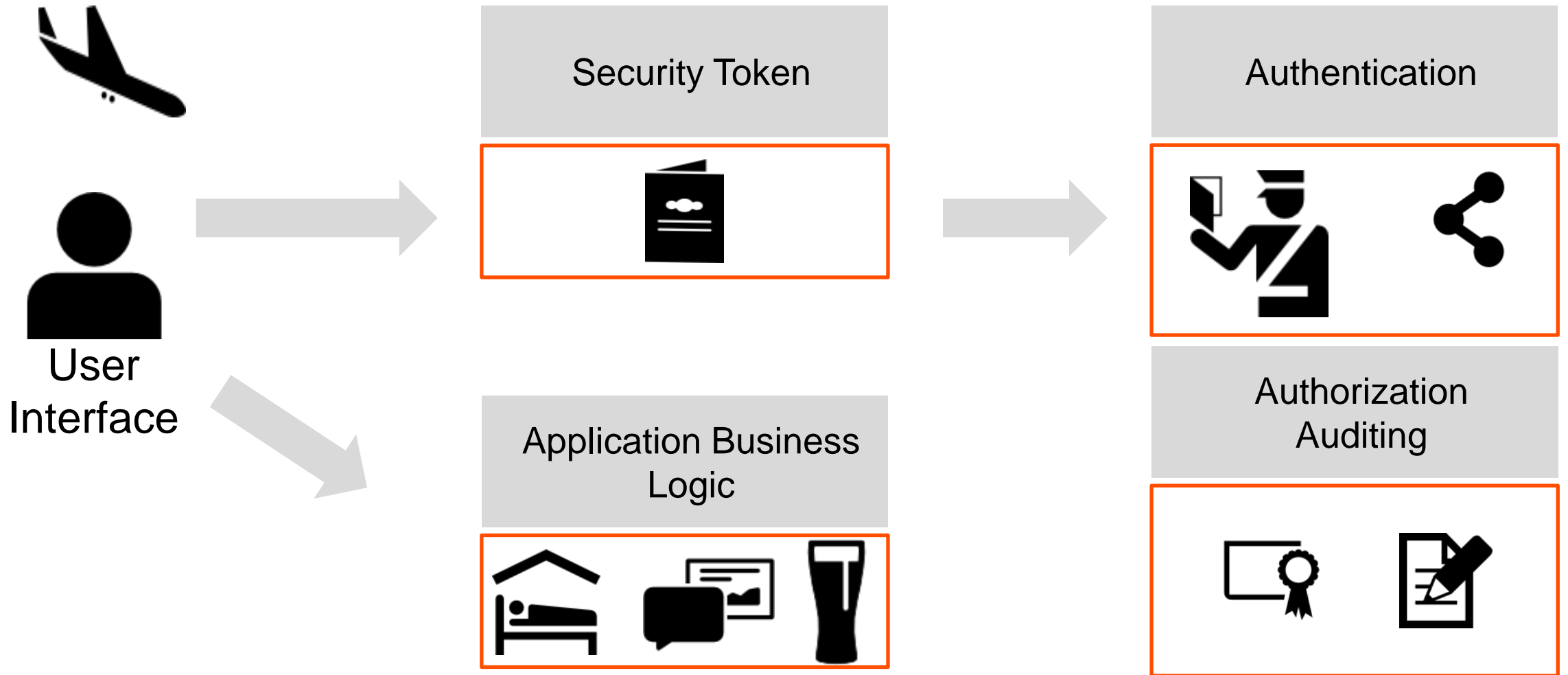


Desktop.MainForm.cls

```
method public logical LoginUser(  
    input pcUserName as char,  
    input pcDomain as char,  
    input pcPassword as char):  
  
    run Security/Login.p on hAppServer (  
        pcUserName, pcDomain, pcPassword,  
        output cUserContextId).  
    if cUserContextId eq '' then return false.  
  
    /* set the CCID on the business logic server so that it's  
       transported with every request. */  
    hAppServer:request-info:ClientContextId = cUserContextId.  
  
    return true.  
end method.
```



Application Flow: Business Logic



Application Flow: Business Logic

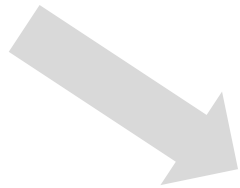
```
RUN getcustomerlist.p ON SERVER hAppServer  
(OUTPUT DATASET dsCustomer)
```

1

Authentication



User Interface



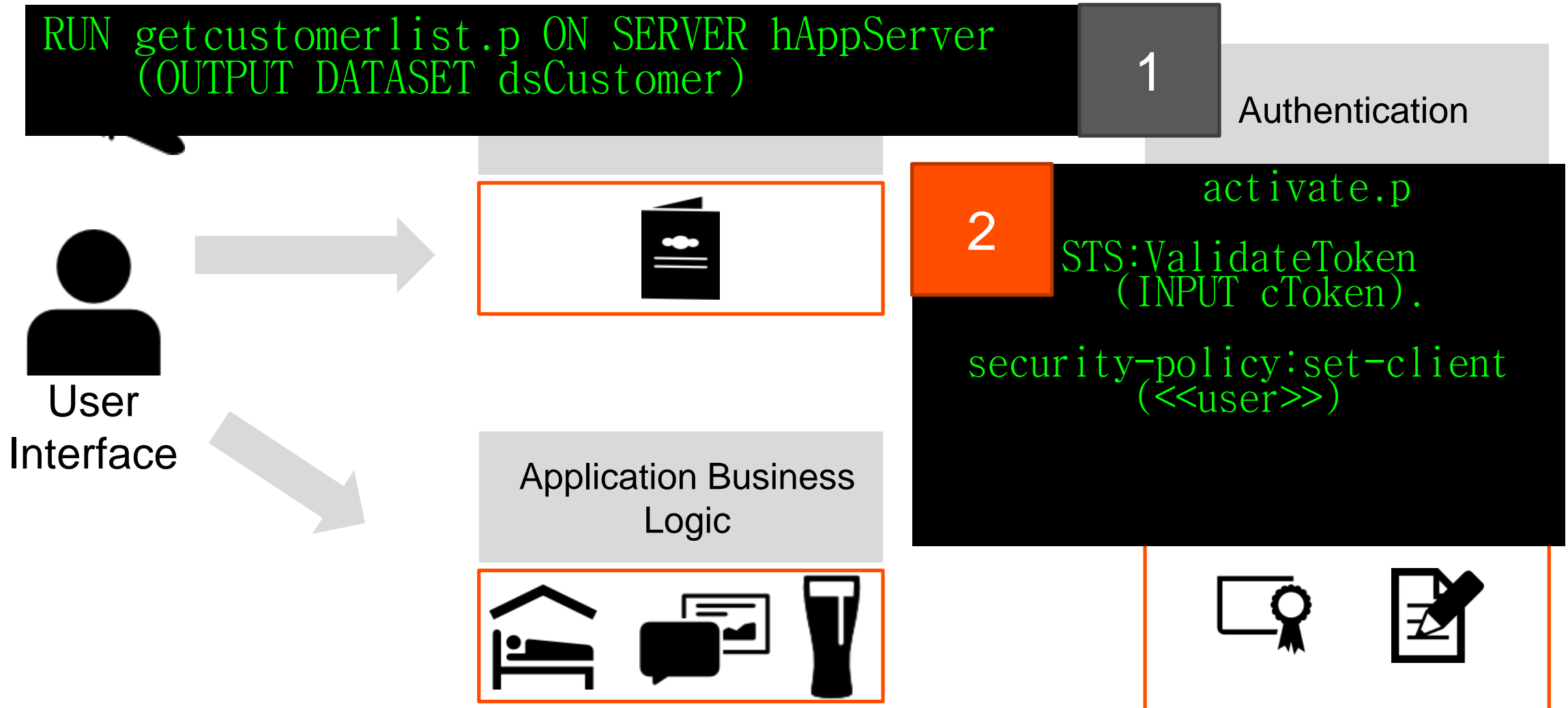
Application Business Logic



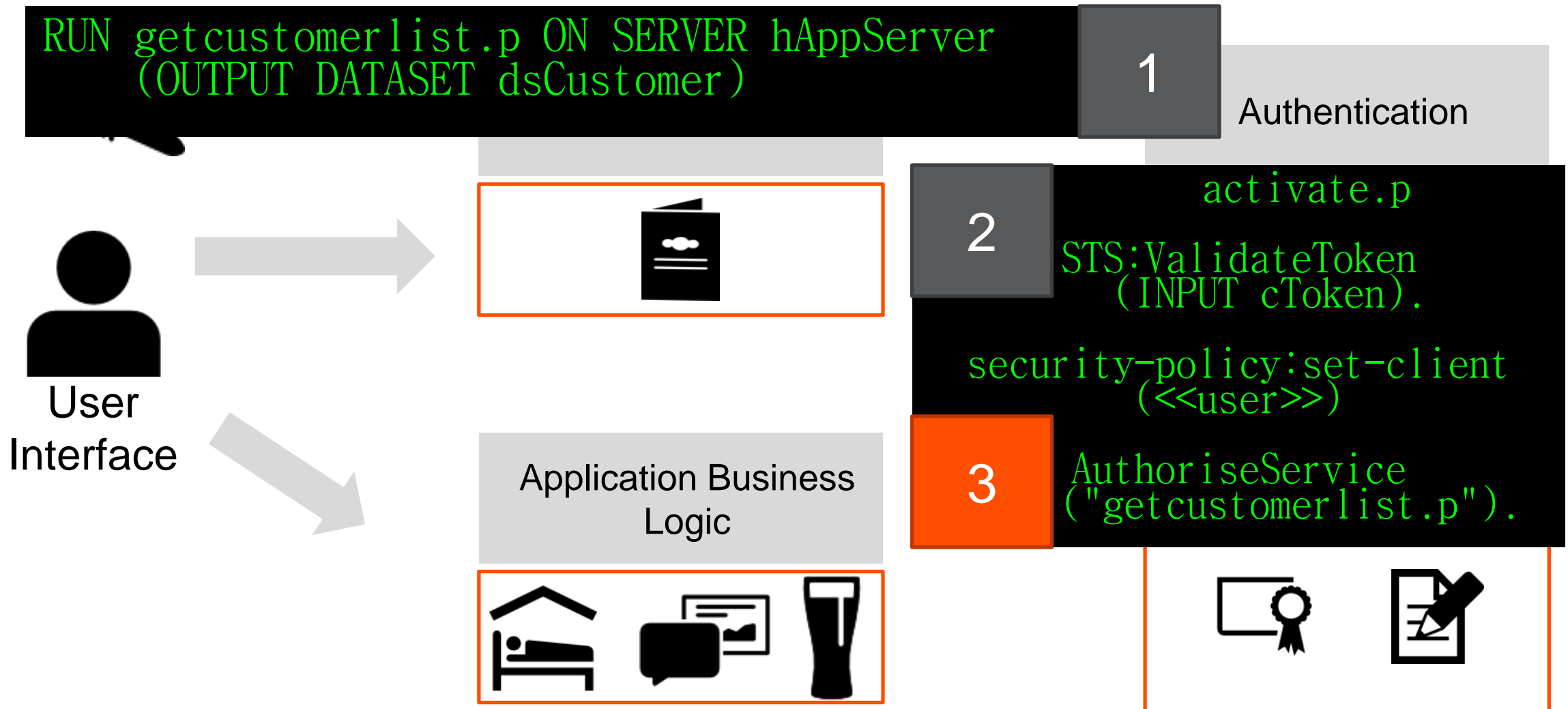
Authorization
Auditing



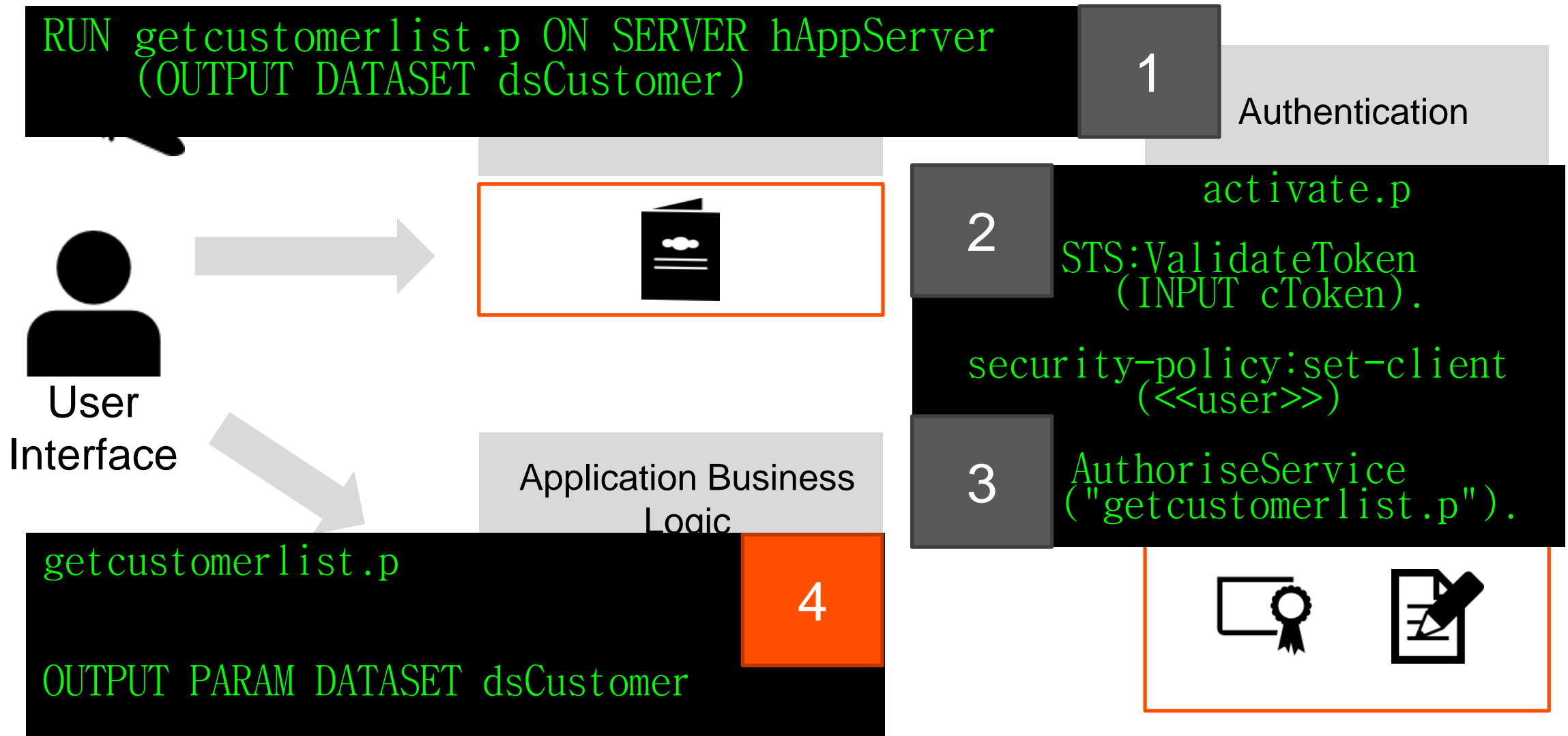
Application Flow: Business Logic



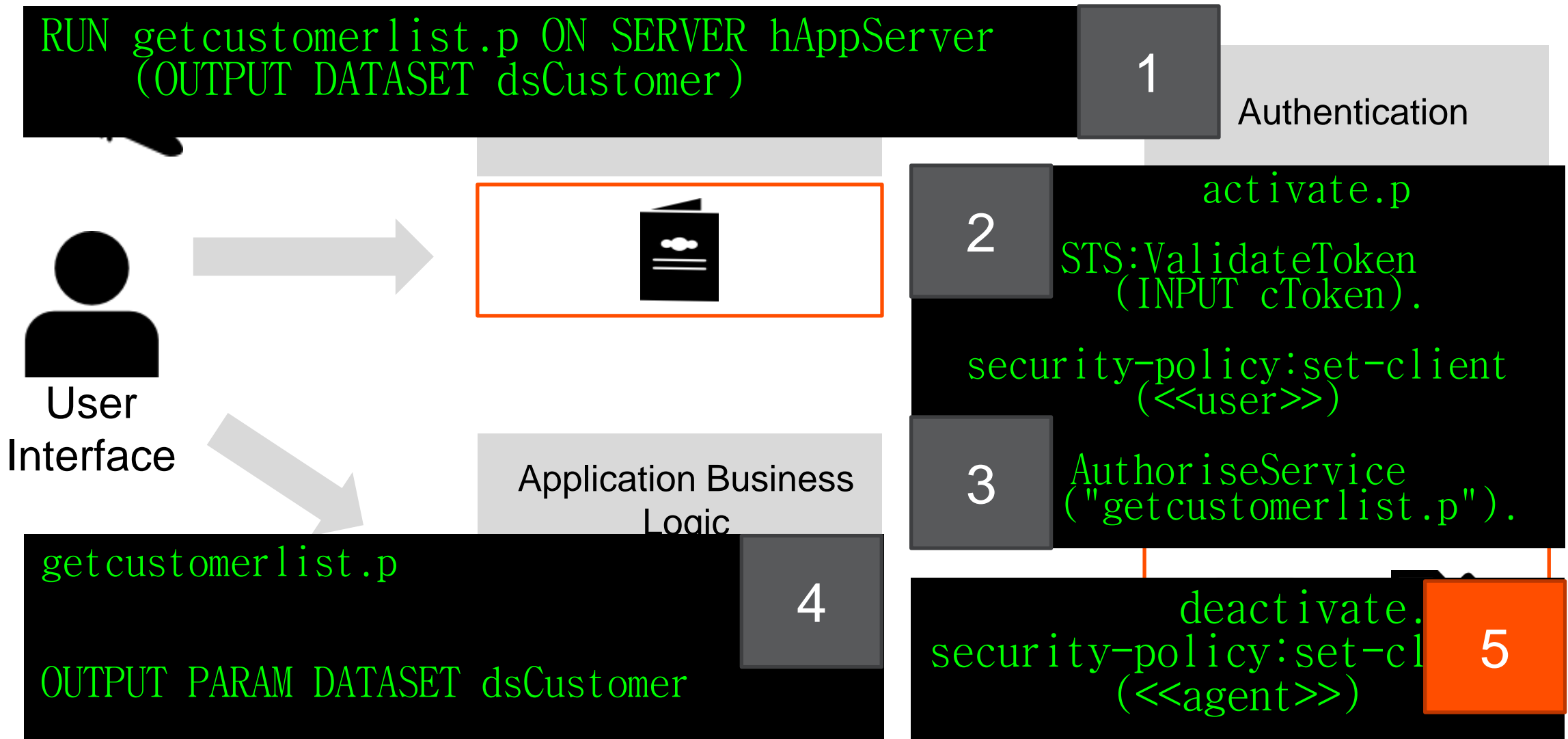
Application Flow: Business Logic



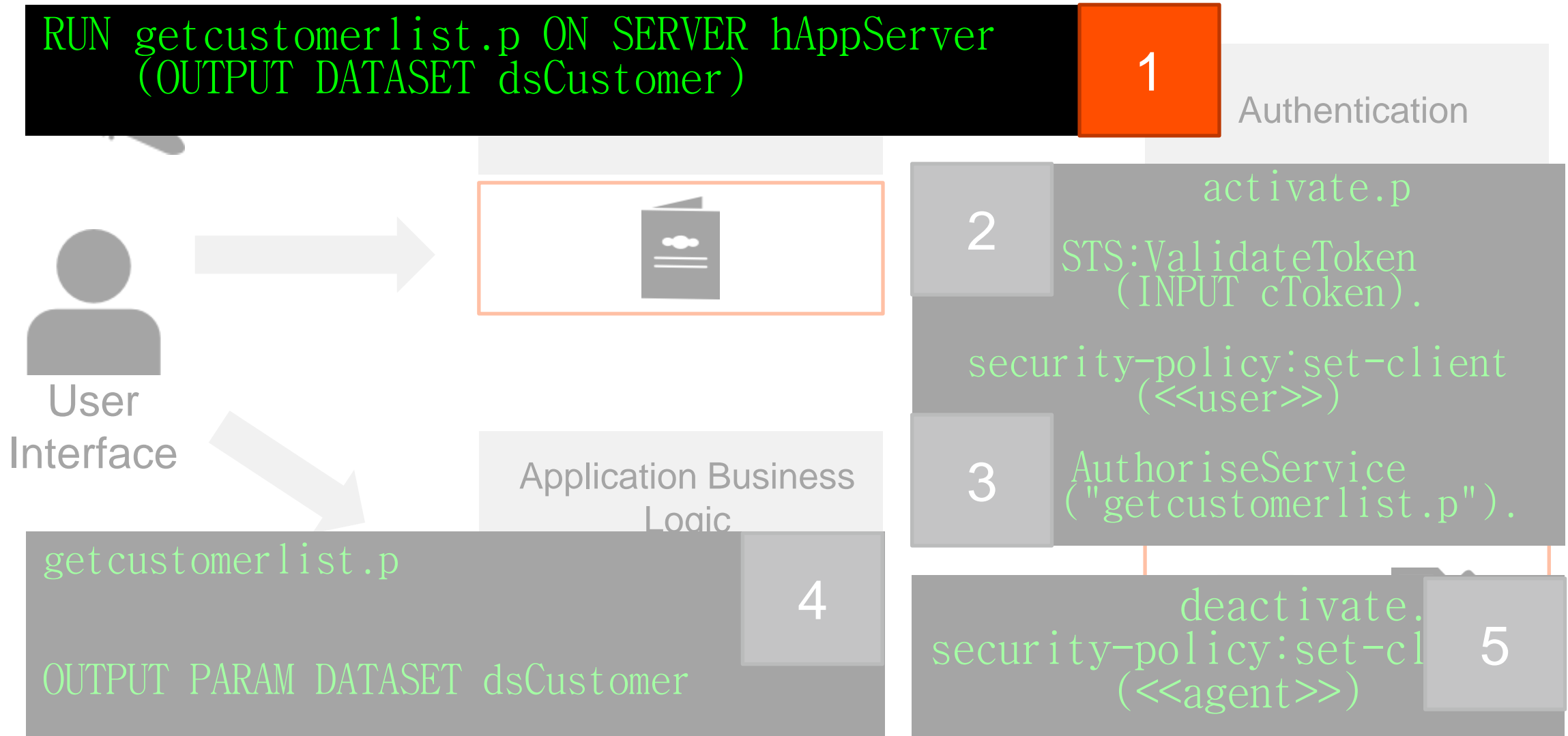
Application Flow: Business Logic



Application Flow: Business Logic



Application Flow: Business Logic



Desktop.MainForm.cls


```
method protected void RefreshCustomerList():
  define variable hAppServer as handle no-undo.

  run BusinessLogic/GetCustomerList.p on hAppServer
    (output dataset dsCustomerOrder).

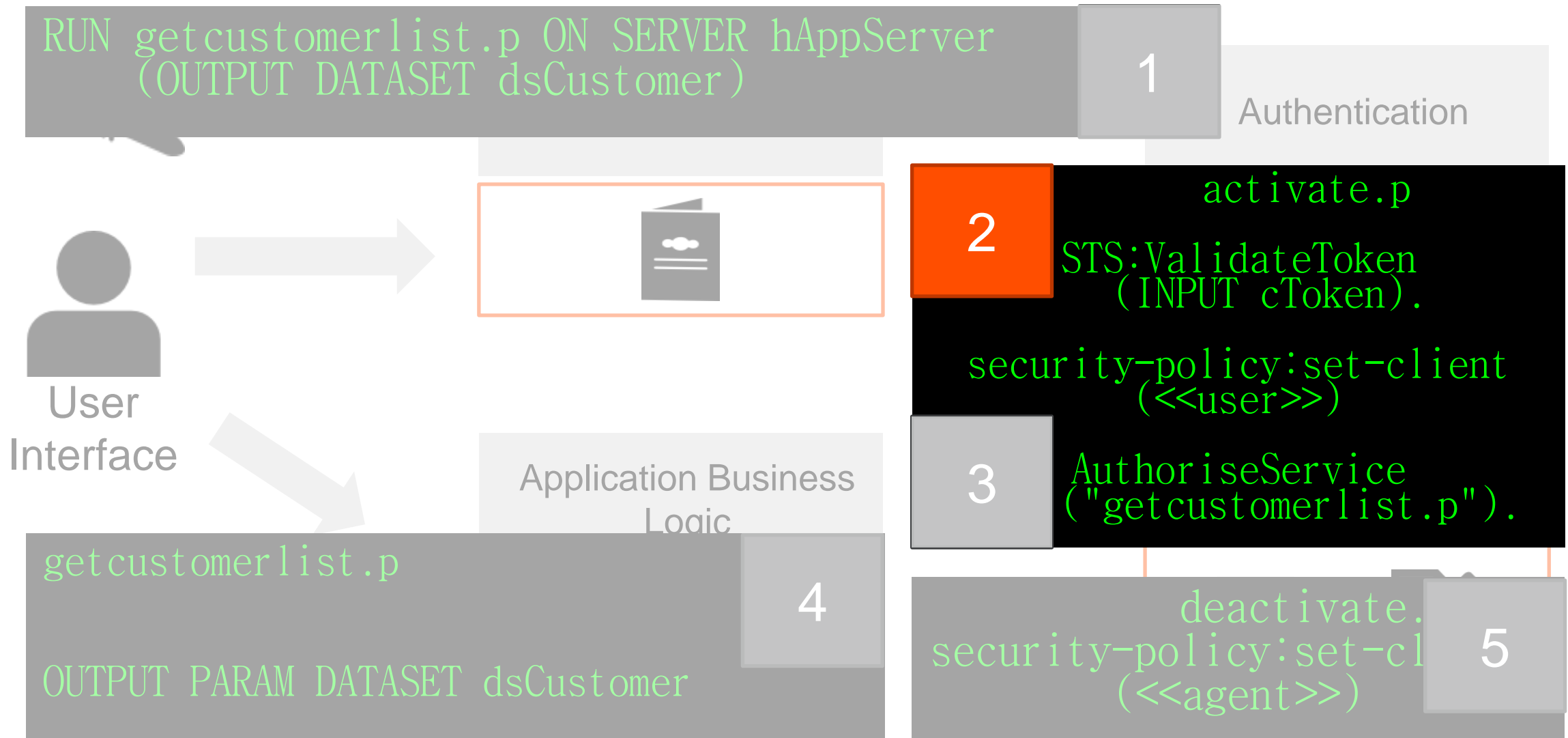
  open query qryCustomer preselect
    each ttCustomer by ttCustomer.CustNum.

  bsCustomer:Handle = query qryCustomer:handle.

  query qryCustomer:reposition-to-row(1).
end method.
```



Application Flow: Business Logic



Security/Activate.p

```
hClientPrincipal = Security.SecurityTokenService:Instance:  
    GetClientPrincipal(  
        session:current-request-info:ClientContextId).
```



```
/* authenticate client-principal */  
security-policy:set-client(hClientPrincipal).
```

Security.SecurityTokenService.cls

```
method public handle GetClientPrincipal(input pcContextId as char):
  define variable hClientPrincipal as handle no-undo.
  define variable rClientPrincipal as raw no-undo.
  define buffer lbSecurityContext for SecurityContext.

  find lbSecurityContext where lbSecurityContext.SessionId eq
pcContextId
    exclusive-lock no-wait no-error.
  if not available lbSecurityContext then
    undo, throw new AppError('Context does not exist').
  assign rClientPrincipal = lbSecurityContext.ClientPrincipal
    lbSecurityContext.LastAccess = now.

  create client-principal hClientPrincipal.
hClientPrincipal:import-principal(rClientPrincipal).

  return hClientPrincipal.
end method.
```



Security/Activate.p

```
hClientPrincipal = Security.SecurityTokenService:Instance:
  GetClientPrincipal(
    session:current-request-info:ClientContextId).

/* authenticate client-principal */
security-policy:set-client(hClientPrincipal).
```



_sec-authentication-system

```
create _sec-authentication-system.  
_Domain-type          = 'TABLE-ApplicationUser'.  
_Domain-type-description =  
    'The ApplicationUser table serves as  
    the authentication domain'.  
_PAM-plugin           = true.  
  
_PAM-callback-procedure =  
    'Security/AppUserAuthenticate.p'.
```



Security/AppUserAuthenticate.p

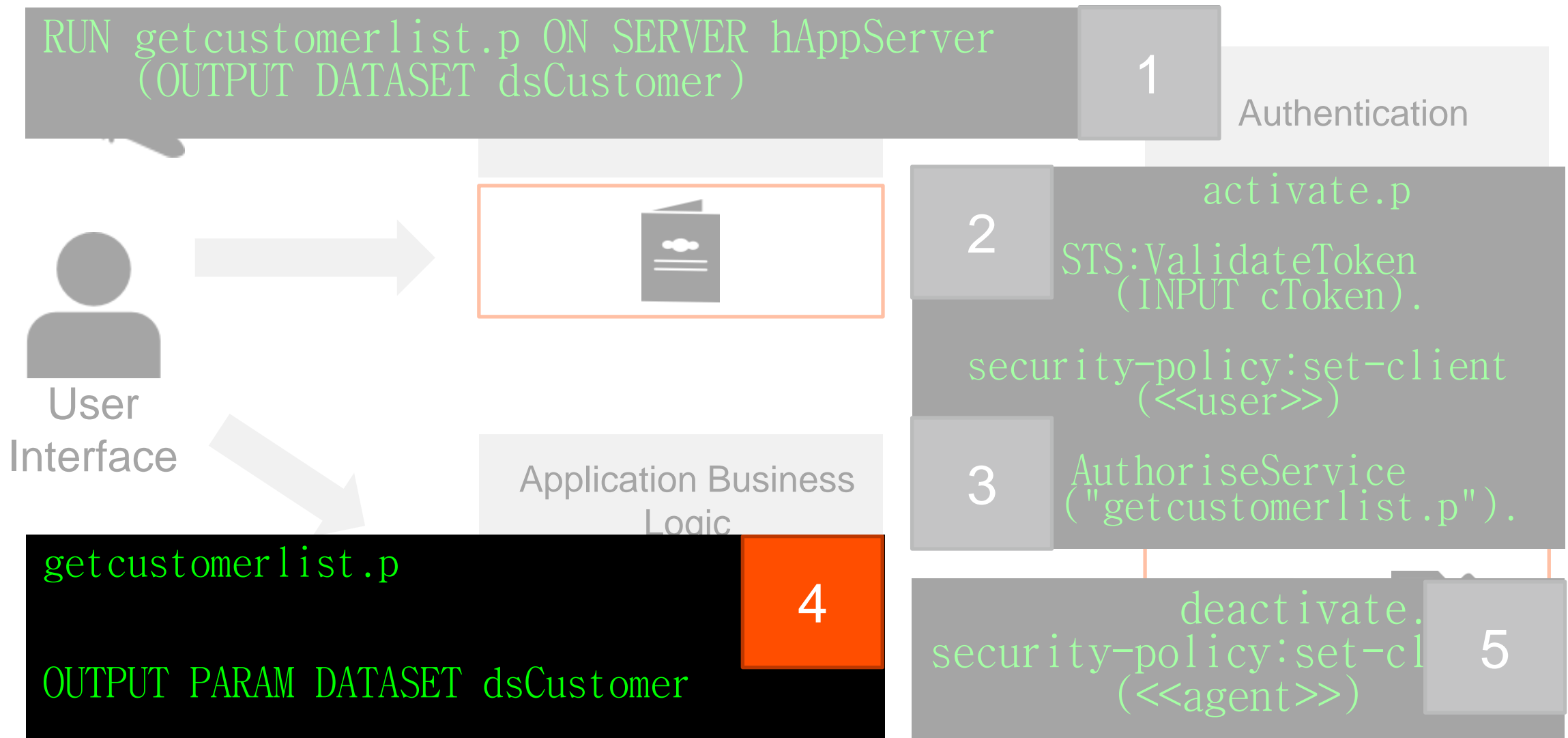
```
procedure AfterSetIdentity:
  def input param phClientPrincipal  as handle no-undo.
  def input param pcSystemOptions as character extent no-undo.

  /* At this point the CLIENT-PRINCIPAL is sealed and the
     user authenticated */

  /* Load user/application (as opposed to security)
     context here */

  return.
end procedure.
```

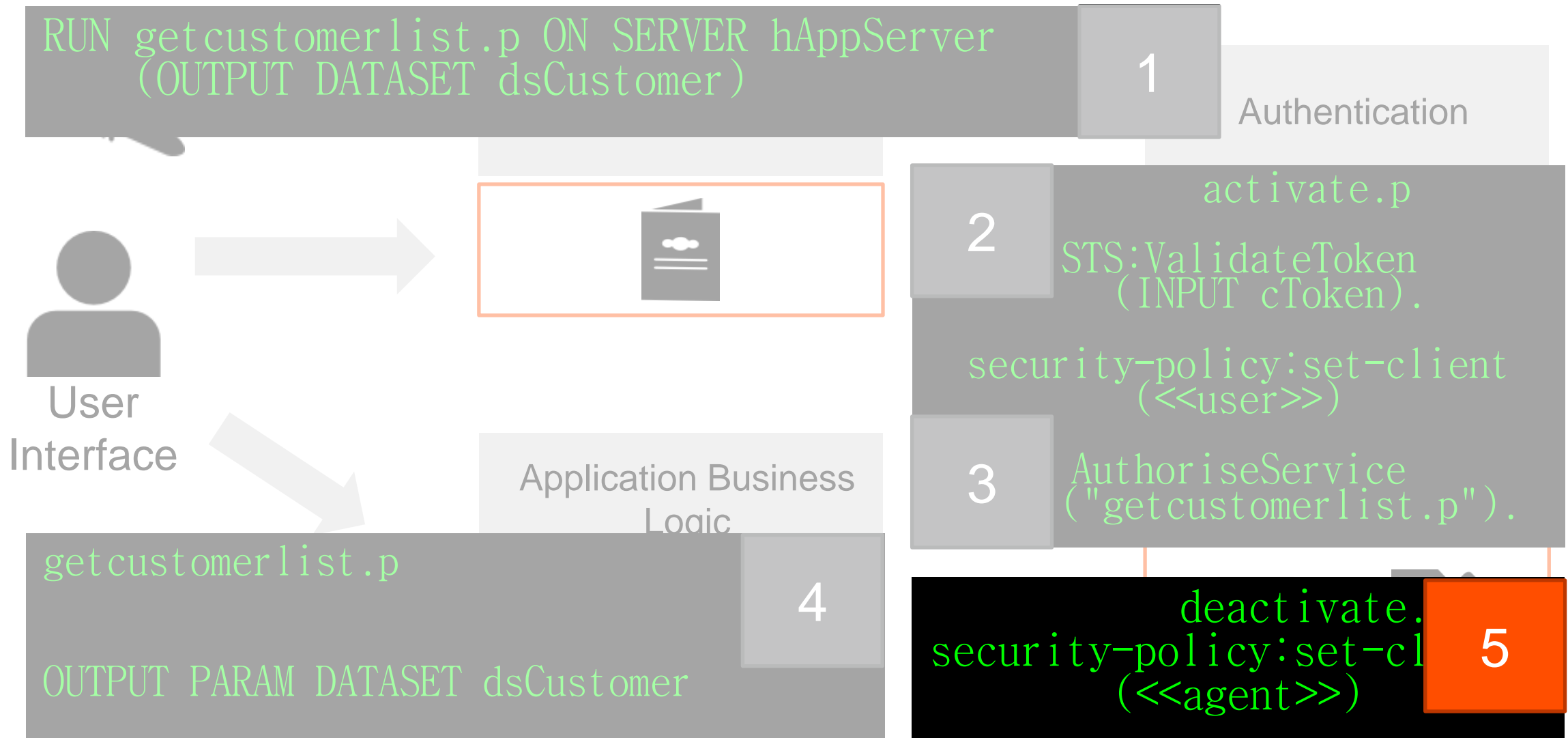
Application Flow: Business Logic



BusinessLogic/GetCustomerList.p

```
{BusinessLogic/dsCustomerOrder.i}  
define output parameter dataset for dsCustomerOrder.  
define variable oBusinessEntity as CustomerOrderBE no-undo.  
oBusinessEntity = new CustomerOrderBE().  
oBusinessEntity:GetCustomers(output dataset dsCustomerOrder).  
/* eof */
```

Application Flow: Business Logic



Security/Deactivate.p

```
define variable hClientPrincipal as handle no-undo.  
  
hClientPrincipal = dynamic-function(  
    'GetAgentClientPrincipal' in hStartupProc)  
  
security-policy:set-client(hClientPrincipal).  
  
/* eof */
```



Desktop.MainForm.cls


```
method protected void RefreshCustomerList():
    define variable hAppServer as handle no-undo.

    run BusinessLogic/GetCustomerList.p on hAppServer
        (output dataset dsCustomerOrder).

    open query qryCustomer preselect
        each ttCustomer by ttCustomer.CustNum.

    bsCustomer:Handle = query qryCustomer:handle.

    query qryCustomer:reposition-to-row(1).
end method.
```



Application Flow: Business Logic

```
RUN getcustomerlist.p ON SERVER hAppServer  
(OUTPUT DATASET dsCustomer)
```

1

Authentication



User
Interface

```
startup.p  
security-policy:load-domains()  
STS:Login('agent', 'system').  
security-policy:set-client  
(<<agent>>).
```

0

```
activate.p  
S:ValidateToken  
(INPUT cToken).  
security-policy:set-client  
(<<user>>)  
authoriseService  
(getcustomerlist.p").
```

```
getcustomerlist.p
```

```
OUTPUT PARAM DATASET dsCustomer
```

4

```
deactivate.  
security-policy:set-cl  
(<<agent>>)
```

5

Security/Startup.p

```
define input parameter pcStartupData as character no-undo.
define variable cAgentSessionId as character no-undo.
define variable hClientPrincipal as handle no-undo.

/* load domains */
security-policy:load-domains('sports2000').

/* immediately set session user to a low-privilege agent user */
cAgentSessionId = Security.SecurityTokenService:Instance
                  :LoginUser('agent', 'system', 'oech1::3c373b2a372c3d').

hClientPrincipal = Security.SecurityTokenService:Instance
                  :GetClientPrincipal(cAgentSessionId).

security-policy:set-client (hClientPrincipal).

function GetAgentSessionId returns character ():
    return cAgentSessionId.
end function.

function GetAgentClientPrincipal returns handle():
    return hClientPrincipal.
end function.
```



Security/Shutdown.p

```
Security.SecurityTokenService:Instance
  :LogoutUser(
    dynamic-function('GetAgentSessionId' in hStartupProc)).

/* eof */
```

Progress OpenEdge Provides ...

- A security token
 - CLIENT-PRINCIPAL available in multiple clients
 - Automatic creation in some cases
 - Available in activate procedure
- Configurable, plug-in architecture (PAM modules)
 - Guaranteed, consistent, trusted code-paths

Progress OpenEdge Does Not ...

- Have a prescriptive model
- Manage security context for an entire application
- Automatically import external systems' tokens
 - For example, SAML for federated authentication

Coming Soon ... {std/disclaimer.i}

- More authentication systems / PAM modules
 - LDAP
 - ActiveDirectory
- Upgraded security for _User
- OpenEdge realm for BPM & REST

Progress.Security.Realm.IHybridRealm

Summary

- Identity management is a process that helps protect your business data
- Applications must have security designed in
 - Delegation of responsibility
 - Multiple layers
- OpenEdge provides components of identity management
 - CLIENT-PRINCIPAL
 - Authentication Systems
 - Transportation of security token

Extra Materials

- This session's slides to be posted on Progress Exchange site
 - Supporting code at https://github.com/nwahmaet/IdM_Sample
- Other Exchange sessions
 - Coding with Identity Management & Security (Part 2)
Peter Judge, PSC
 - Workshop: Progress OpenEdge Security
Brian Bowman, Rob Marshall et al
 - Transparent Data Encryption
Doug Vanek
 - Introduction to Multi-tenancy
Gus Bjorklund
 - Security and Session Management with Mobile Devices
Mike Jacobs & Wayne Henshaw
- Image Credits:
Passport designed by Catia G, Time designed by wayne25uk, Database designed by Anton Outkine, Code designed by Nikhil Dev, Imposter designed by Luis Prado, User designed by T. Weber, Fingerprint designed by Andrew Forrester, Document designed by Samuel Green, Certificate designed by VuWorks, Network designed by Ben Rex Furneaux, Beer designed by Leigh Scholten; all from The Noun Project



PROGRESS