Liberty Advanced Client Overview

Conor P. Cahill
Principal Engineer
Intel Corporation

Evolution of Liberty related Clients

- Phase 1: Liberty Enabled Client/Proxy (LECP)
- Phase 2: Active Client
- Phase 3: Advanced Client (aka Intelligent Client)
- Phase 4: Robust Client

Evolution: LECP

- Liberty Enabled Client/Proxy
- Facilitate SSO and Federation operations
  - Especially IDP Discovery
  - Authentication Request Direction
- Browser plug-in and/or Proxy server
- Incorporated into SAML 2.0 as ECP

Evolution: Active Client

- AKA: LUAD
- Local Web Services Consumer (WSC)
  - Radio Service client
  - Calendar Service client
- Liberty ID-WSF Authentication Service
  - SOAP profile of SASL
    - Supports *any* authentication protocol
  - Enabled SSO into Web Services

Advanced Client

- The client as an extension of the IdP
  - Off-line and privacy enabling modes
  - Strong local authentication
- Provisioning
  - Functionality and/or data
- Locally hosted/managed services
- Reporting

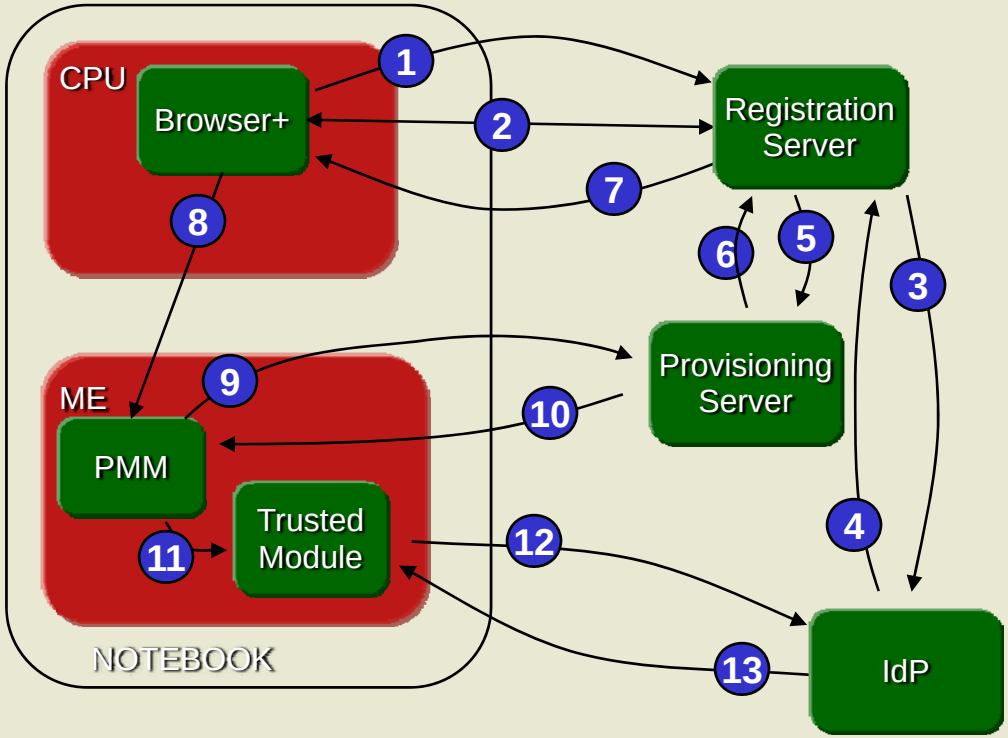
Future: Robust client

- **Provisioning (pulled into Advanced Client)**
  - **Over-the-wire/air distribution of client modules**
  - **Support for trusted environments**
- **Mobility**
  - **Moving service instances and/or client modules**
- **Strong Authentication**
  - **multi-factor**

Advanced Client: Provisioning

- Functionality and/or data provisioning
- Full Lifecycle Management
  - Provision, Delete, Activate, Deactivate, Update
- Push and Pull (poll) options

Provisioning A Module



1. User initiates registration through the browser
2. Registration Server (RS) collects information from the user
3. RS asks IdP for initial token
4. IdP generates token and returns to RS
5. RS creates Provisioning Descriptor (PMDesc) which includes initial token from IdP and registers PMDesc with Provisioning Server (ProvS)
6. ProvS generates Provisioning Handle (PH) and returns it to RS
7. RS sends PH to plug-in in browser
8. Browser plug-in sends PH to Provisioned Module Manager (PMM)
9. PMM decodes PH and invokes ProvS to get PMDesc
10. ProvS returns PMDesc to PMM
11. PMM Instantiates Trusted Module (TM) and passes in initialization data from PMDesc
12. TM generates private keys and registers them with IdP using initial token
13. IdP acknowledges registration

Advanced Client: Delegated SSO

- Trusted Module (TM)
  - Extension of an IdP
  - Usually in some form of protected environment
    - Closed
    - Tamper resistent
    - E.g.: SIM
  - Drive SSO and/or Federation operations
  - Able to manufacture and/or store assertions
  - Able to function when IdP is not present

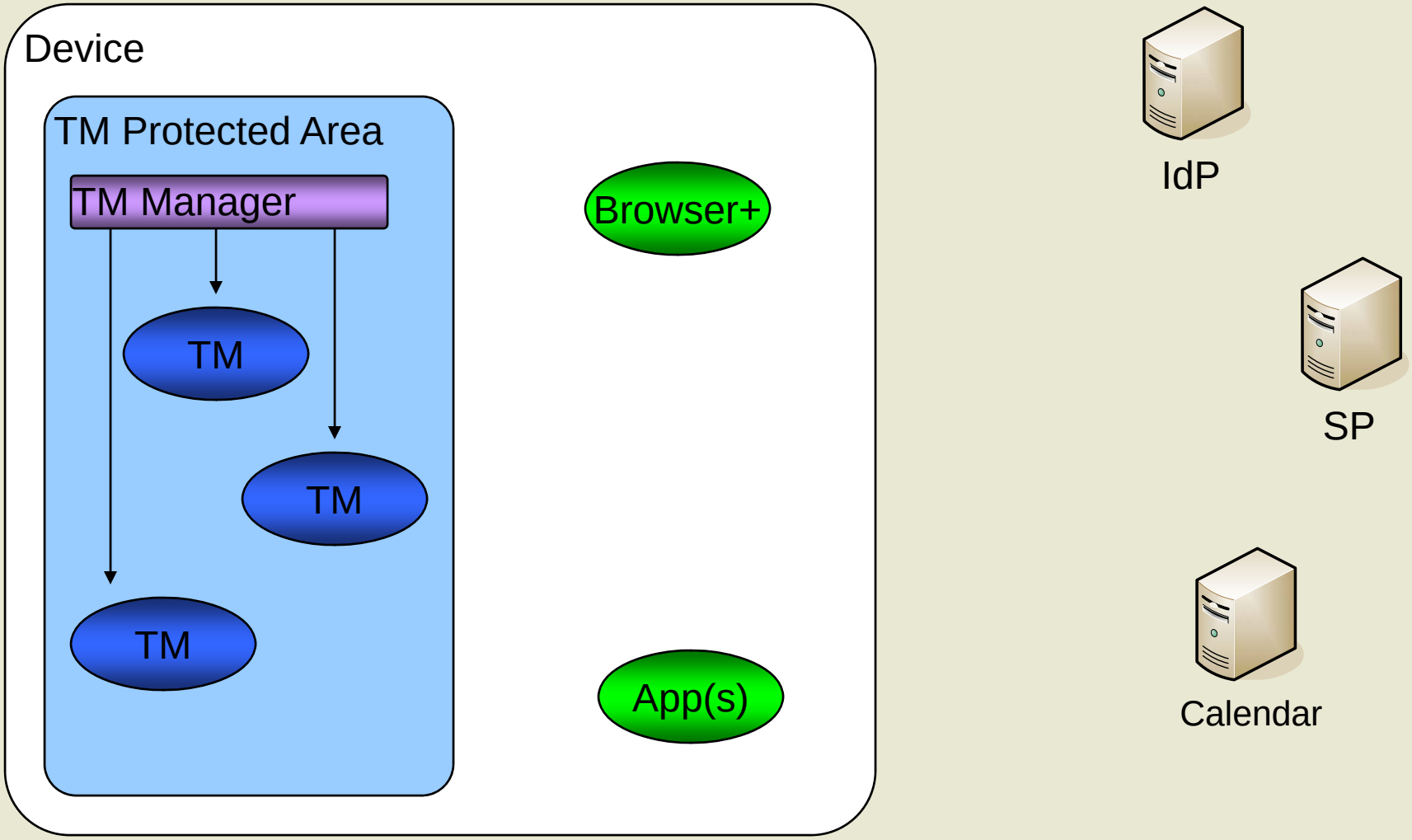
TM: Design Considerations

- Privacy
  - not shared by many users
  - ID of TM could be a correlation handle
  - Same for any public key used by TM for security
- Security
  - Mostly out-of-scope for Liberty
  - Enable features necessary for secure distribution

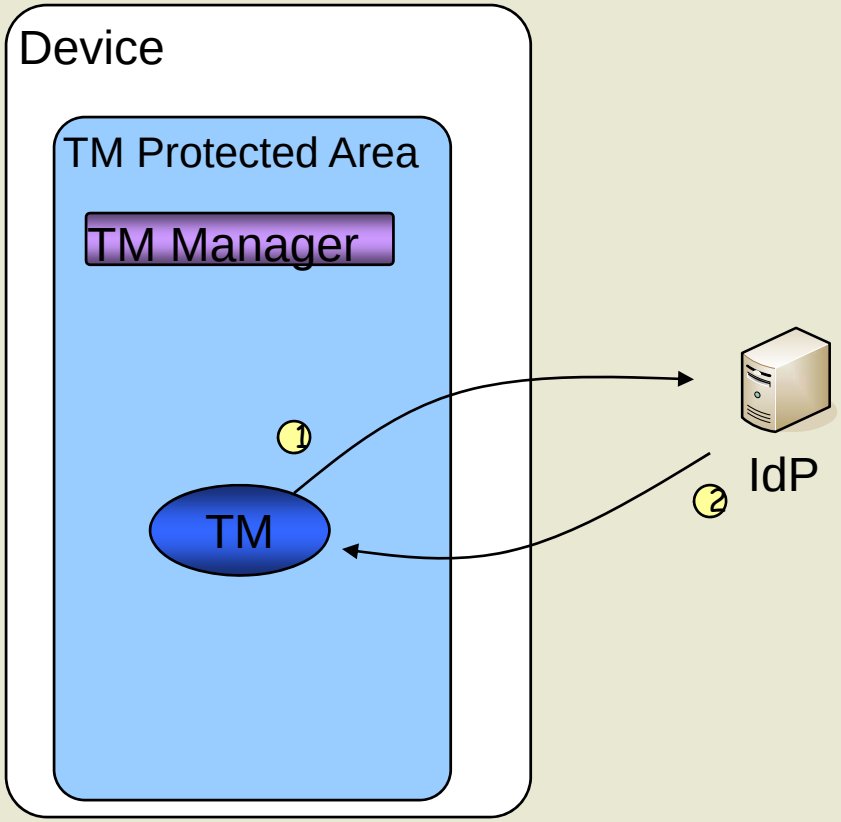
TM: Single Sign On Assertions

- Local manufacture of Assertions by TM (Minting)
  - IdP Authorizes TM to manufacture Assns
  - IdP controls facets of Assn
  - Relying Party (RP) can verify delegation
  - Privacy Protected by using uniqe keys for each RP
- Long term storage of IdP Issued Assns (Hoarding)
  - IdP issues Assns to TM
  - TM chooses when one of those Assns used for SSO

TM Conceptual Environment

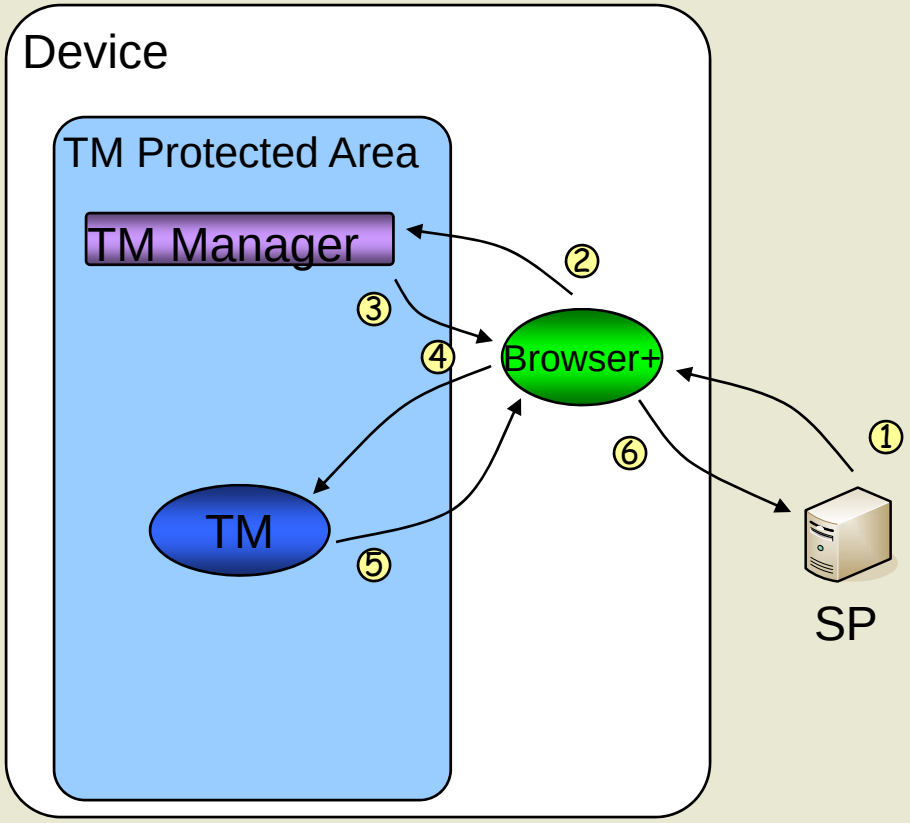


TM Pre-Separation (Minting)



1. TM Requests Minting Assertions
2. IDP responds with Minting Assns

TM Browser SSO



1. SP initiates SSO (AuthnRequest)
2. Browser “discovers” TMs
3. TM Manager returns TM EPR
4. Browser forwards AuthnReq to TM
5. TM Responds with AuthnReq for SP
6. Browser forwards response to SP – user is not SSO’d into SP

Advanced Client: Locally hosted Services

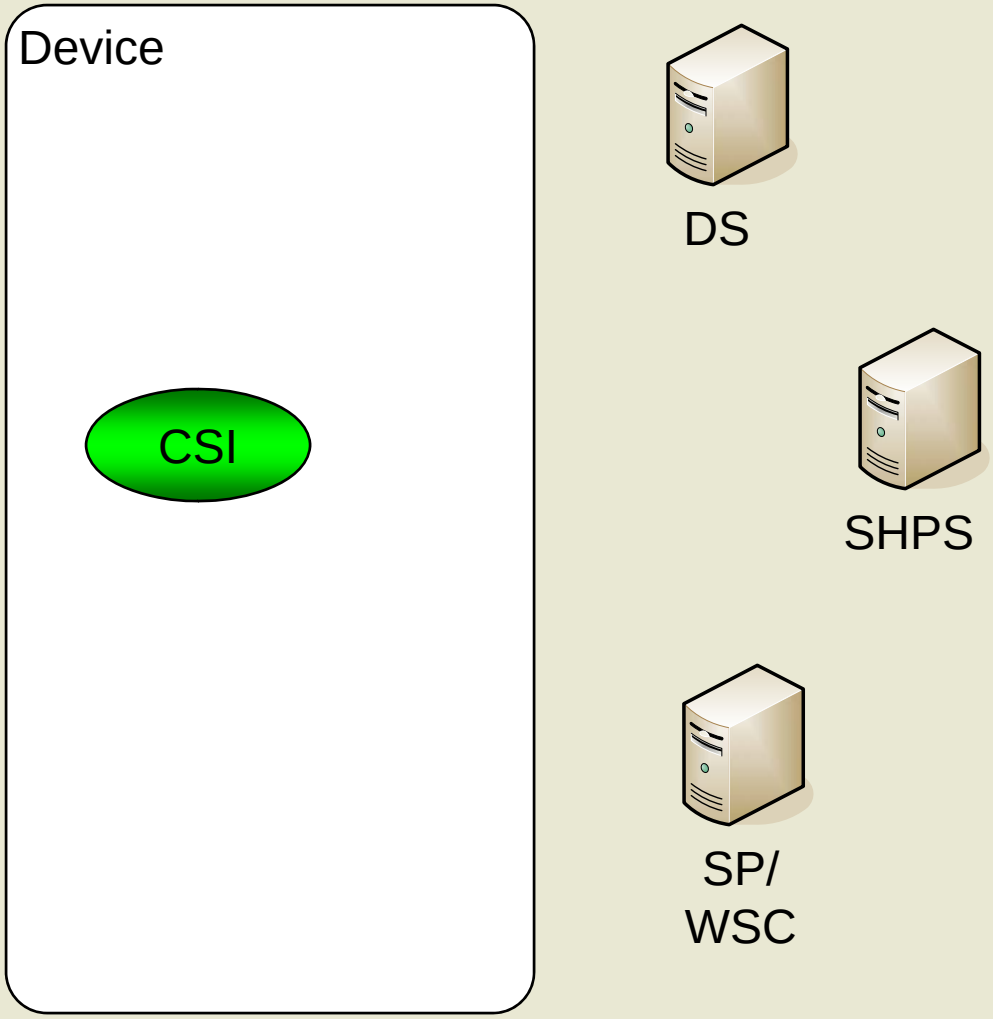
- Locally hosted service instance
  - E.g. Profile, Calendar, Payment, etc.
- May or may not be in a trusted environment
- Looks, feels, and acts like a typical ID-WSF or ID-SIS service
- Issues:
  - Privacy (location becomes correlation handle)
  - Availability/connectivity

LIBERTY ALLIANCE PROJECT

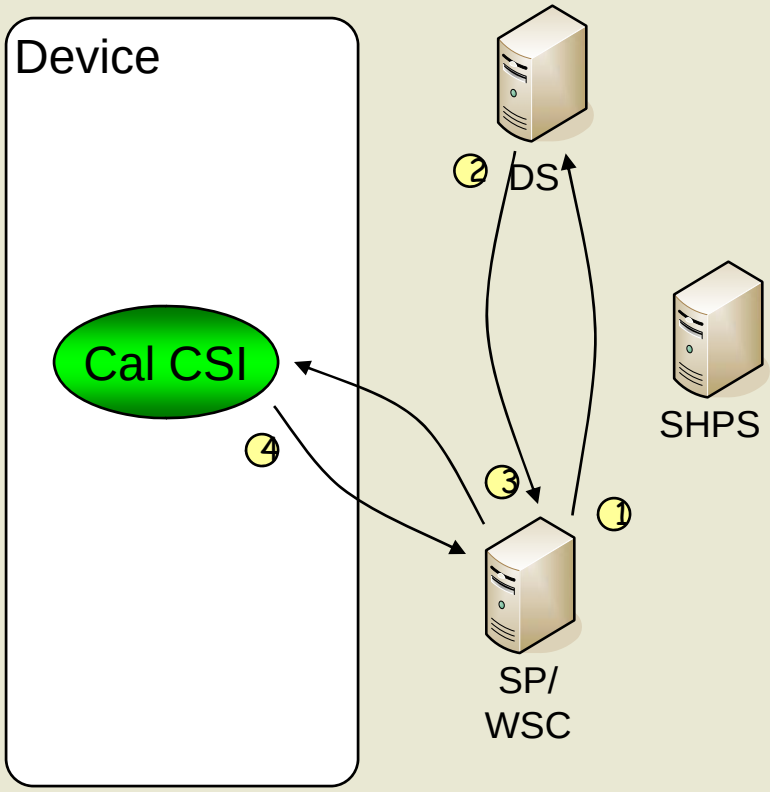
SHPS: A remote partner

- Service Hosting/Proxying Service
- Hosts a remote instance of service
  - Full implementation of service
  - Synchronization with Client Service Instance (CSI)
  - CSI seen as master, but WSCs interact with Hosted service
- Proxies remote service invocations
  - Forwards each invocation to CSI

CSI Conceptual Environment

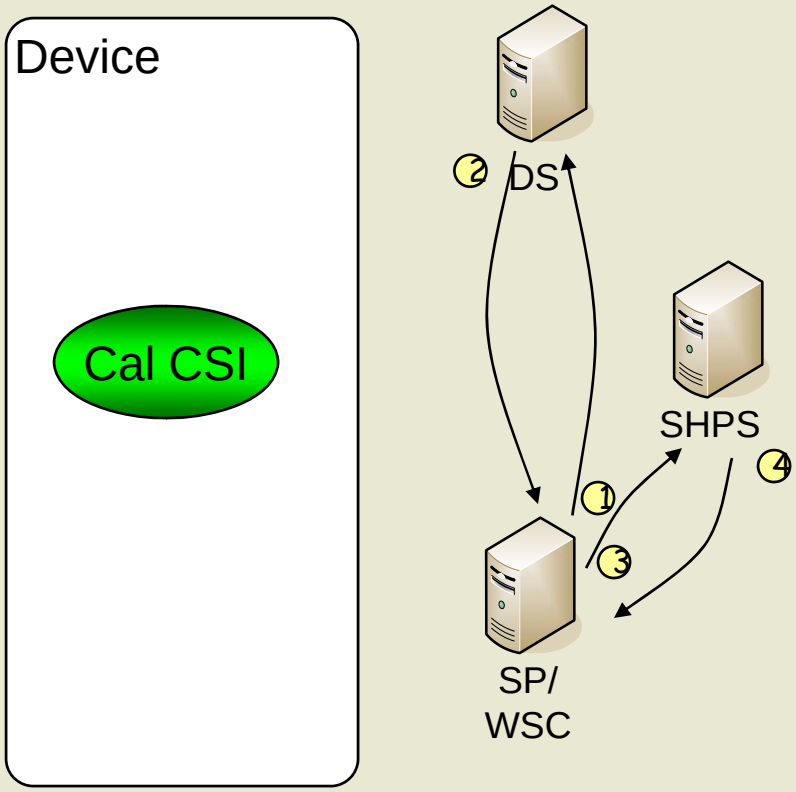


CSI normal ID-WSF invocation



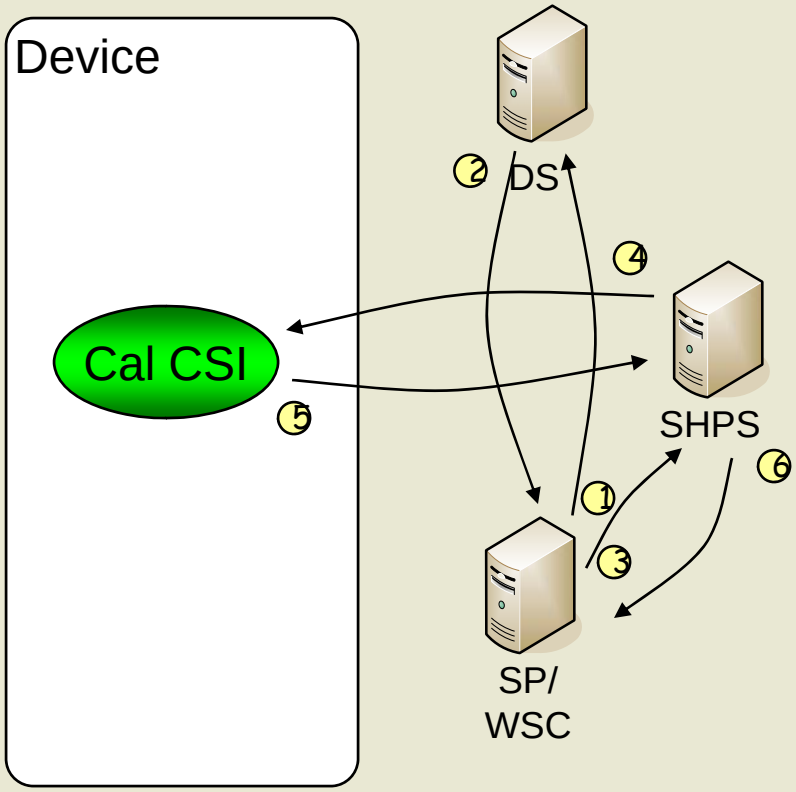
1. WSC discover's Calendar service
2. DS returns CSI's EPR to WSC
3. WSC invokes Calendar CSI
4. Calendar CSI returns data

CSI with SHPS hosting



1. WSC discover's **Calendar** service
2. DS returns Calendar EPR to WSC
3. WSC invokes Calendar Svc
4. SHPS Cal Svc returns response

CSI with SHPS Proxy



1. WSC discover's **Calendar** service
2. DS returns Calendar EPR to WSC
3. WSC invokes Calendar Svc
4. SHPS forwards req to Cal CSI
5. Cal CSI sends response to SHPS
6. SHPS returns response to WSC

Curent Status

- **Advanced client**
  - **Final specs published 12/07**
- **Robust Client**
  - **Requirements completed**
  - **Some specifications work (Provisioning, PM)**
  - **No public estimates as to spec release**

More Information

- **Web: http://www.projectliberty.org**
- **Demo: RSA Security Conference 4/6/08**
- **My blog: http://conorcahill.blogspot.com**
- **Email: Conor.P.Cahill – at - intel.com**

LIBERTY ALLIANCE PROJECT

Questions?