ENTEGRITY Solutions

PKCS#11
Interoperability/Conformance Testing

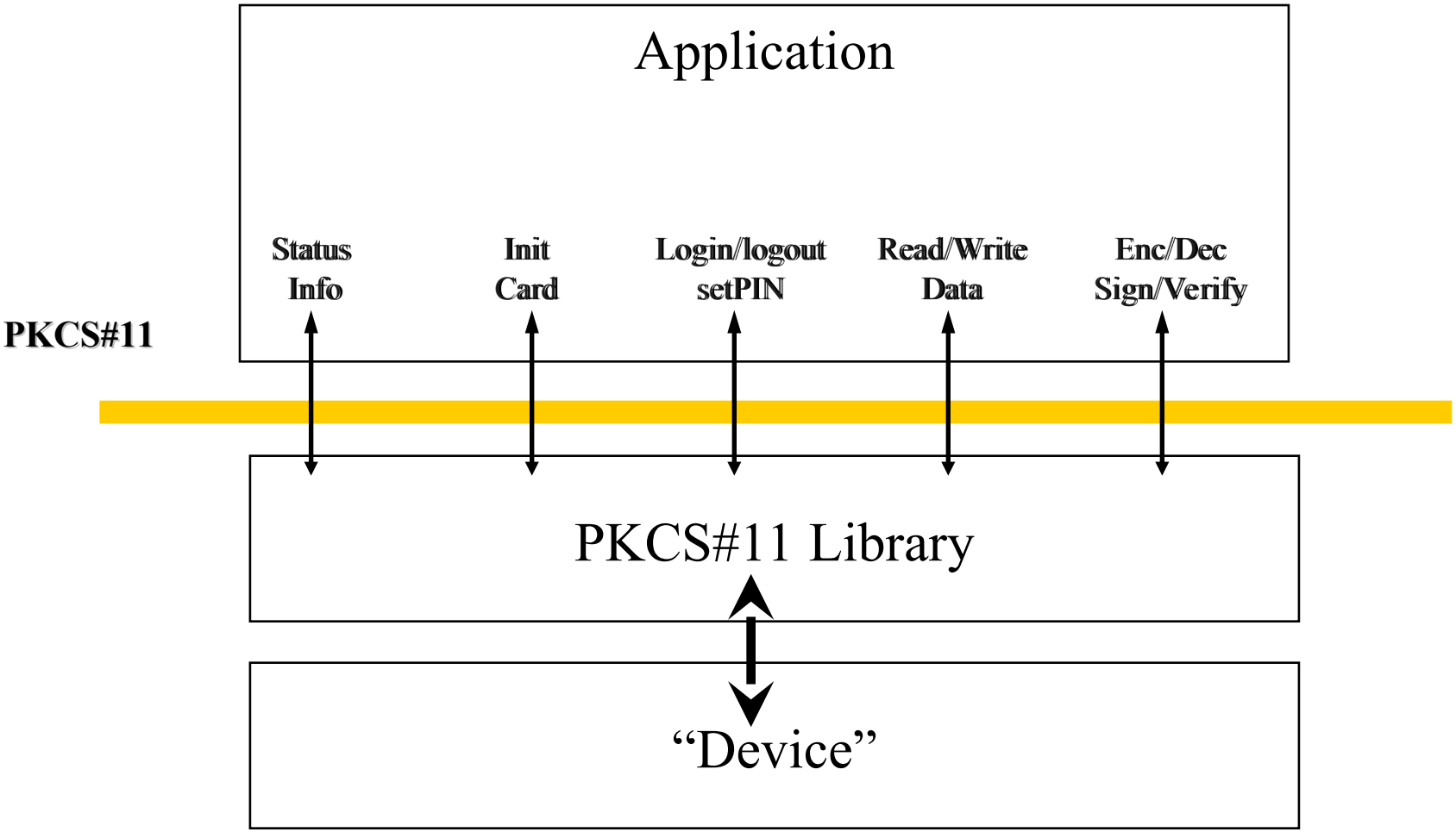
John Hughes

PKIForum meeting

Montreal - 14 September 00

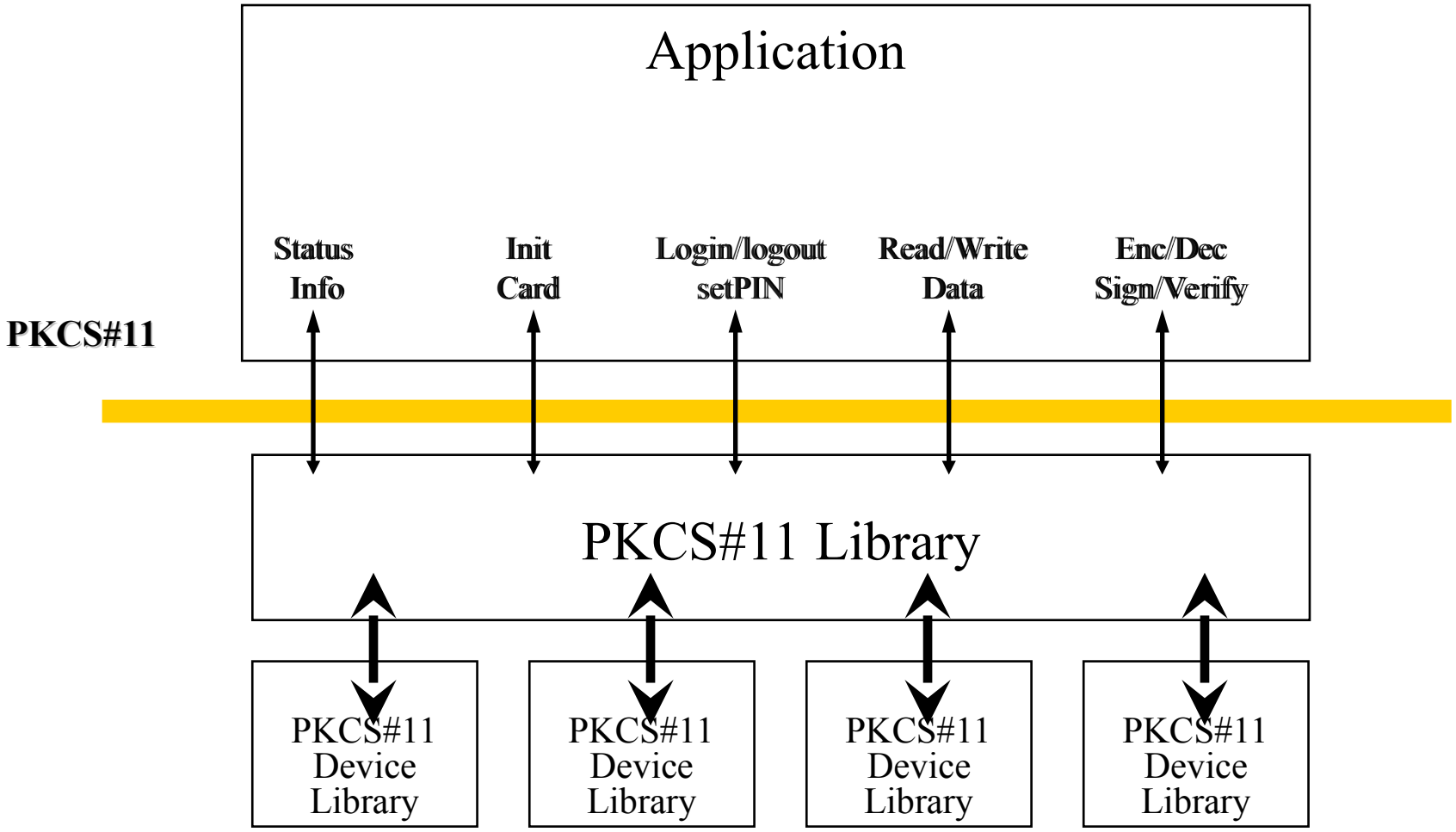
ENTEGRITY Solutions

Typical PKCS#11 Architecture



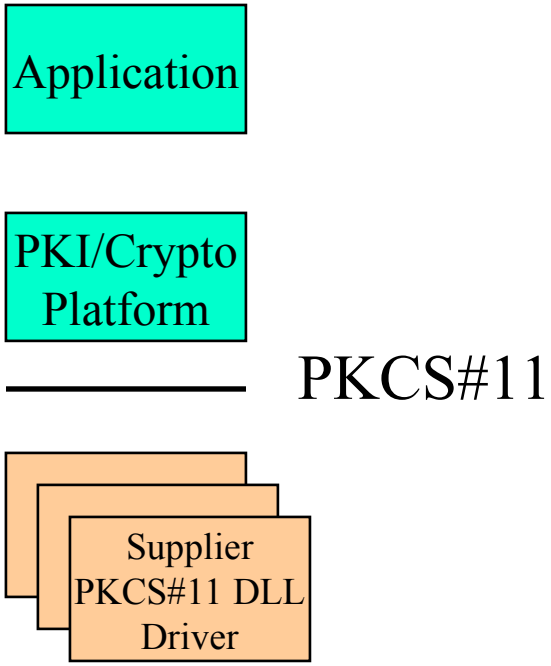
ENTEGRITY Solutions

Common Approach



ENTEGRITY Solutions

Why are we feeling pain - 1?



We do not want device specific changes to our code base

Why are we feeling pain - 2?

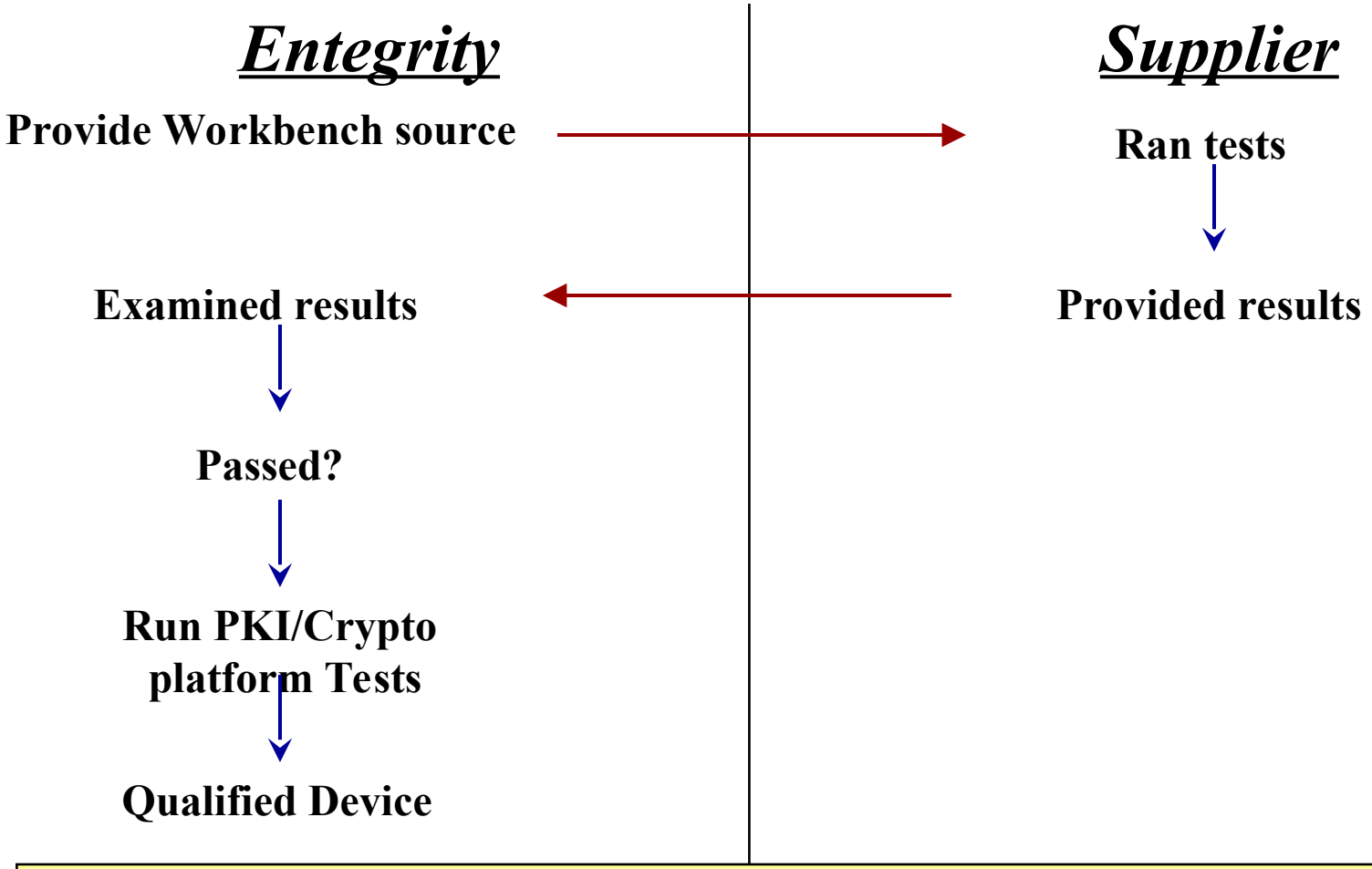
- We have a relativity sophisticated use - our full PSE "profile" is:
  - support for data objects - varying size
  - mechanisms
    - `CKM_RSA_PKCS_KEY_PAIR_GEN`
    - `single part CKM_RSA_PKCS decryption`
    - `CKM_RSA_PKCS verification SIGN/SIGN_RECOVER`
  - C_GenerateRandom
  - 2 key pairs stored on card
  - optional storage of certificates
- Our Universal Token Support (UTS) uses a subset of this for existing Tokens (e.g iD2 tokens with PKCS#15)

What did we do?

- Created a PKCS#11 workbench that simulated how our PKI/Crypto engine used PKCS#11. ("*Entegrity PKCS#11 Workbench*")
- Provided it as source (under license) to PKCS#11 device supplier

ENTEGRITY Solutions

Qualification Process (simplified)



Aim of workbench to resolve “most” of errors prior to full tests

Evolution

- As we test more and more devices we are adding in extra “nuances” and tests

Status

- We have/are testing 13 PKCS#11devices from 6 suppliers working on either Wintel or Solaris platforms
- Total of 20 implementations
- Statistics:
  - only 6 implementations have fully passed our tests
  - we are waiting for patches from 4 of the suppliers

Common problems - 1

- Inverted parameters for public and private keys in C_GenerateKeyPair:
  - a change occurred between PKCS#11 1.x and 2.x, Netscape did not change and several vendors decided to be compatible with them rather than following the standard.
- Version of PKCS#1 padding.
  - Most use 1.5 - but 1 started to use 2.0
- Incomplete or wrong mechanism lists and key usage attributes

Common problems - 2

- Disallowing of generating keys with a given usage if the library does not support a mechanism.
  - Some vendors refuse to allow CKA_ENCRYPT attribute if they don't support decryption on the card. Our view is that if they don't support encryption, they just shouldn't list the mechanism as available, this will prevent us from using the key for that purpose even if the key itself is marked as supporting encryption.
- Device supplier being more lenient on the Attributes assigned when an object is being created.
- No support for Data Objects
- PIN problems (min, max sizes and changing values)

Entegrity PKCS#11 Workbench tests

- Login/logout/session
  - successful/unsuccessful logins
  - changing passwords, min password size
- Data Objects
  - object creation/search/read/modify/deletion (small and large) within a session and across sessions (public and private)
- Status Information
  - version, manufacturer, status flags
  - mechanism list
- Cryptographic operations
  - key generation, random no generation
  - asymmetric - sign/verify/encrypt/decrypt tests (RSA)
  - symmetric - encrypt/decrypt (DES)

Workbench principles - 1

- Designed to be extensible. Although focused on RSA and 3DES relatively easy to change to use other algos:

```
cout << "Starting BASIC CRYPTO: simple sign, verify, encrypt and decrypt" << endl;
SHOULD_NOT_THROW( openSession( theSelectedSlotID, &s1 ), true );
SHOULD_NOT_THROW( login( s1, "1111" ), true );
SHOULD_NOT_THROW( destroyAllObjects( s1 ), true );
SHOULD_NOT_THROW( openSession( theSelectedSlotID, &s2, true ), true );
SHOULD_NOT_THROW( testAsymm( s1, s2,
            CKM_RSA_PKCS_KEY_PAIR_GEN, CKM_RSA_PKCS, 1024, 2 ), true );
SHOULD_NOT_THROW( destroyAllObjects( s1 ), true );
SHOULD_NOT_THROW( testSymm( s1, s2, CKM_DES3_KEY_GEN,
             CKM_DES3_ECB, 64, 2 ), true );
SHOULD_NOT_THROW( closeSession( s2 ), false );
SHOULD_NOT_THROW( logout( s1 ), false );
SHOULD_NOT_THROW( closeSession( s1 ), false );
cout << "Ended BASIC CRYPTO: simple sign, verify, encrypt and decrypt" << endl;}
```

Workbench principles - 2

- Error handling

```
rv = (*theFunctionList->C_GetMechanismInfo)( theSelectedSlotID, aMechId, &info);
errorCheck( rv, "C_GetMechanismInfo" );

// Error handling routine
void errorCheck(CK_RV rv, string funcName, CK_RV expectedResponse ) {
      if( rv == CKR_FUNCTION_NOT_SUPPORTED )
      throw Pkcs11_Exc_FNS( funcName );
      else if( rv != expectedResponse ) {
      cout << "Expected " << hex << expectedResponse << " ("
                  << getErrorDescription(expectedResponse) << ") from " << funcName
       << ", received rv = " << hex << rv
       << " (" << getErrorDescription(rv) << ")" << endl;
      throw Pkcs11_Exc( funcName, rv );
      }
}
```

So how can we progress?

- In discussion with RSA concerning making the Entegrity PKCS#11 Workbench “open source”
- How this can be accomplished and successfully managed is going to be discussed at the PKCS workshop in Boston
- Issues:
  - who maintains and develops the source?
  - do we need an accreditation scheme for the emerging profiles - and who does the testing?

It’s in all our interests that PKCS#11 devices become as “plug and play” as possible