Leo Moesgaard (lemo@dk.ibm.com) – manager of IBM Crypto Competency Center
GSE event - December 2011

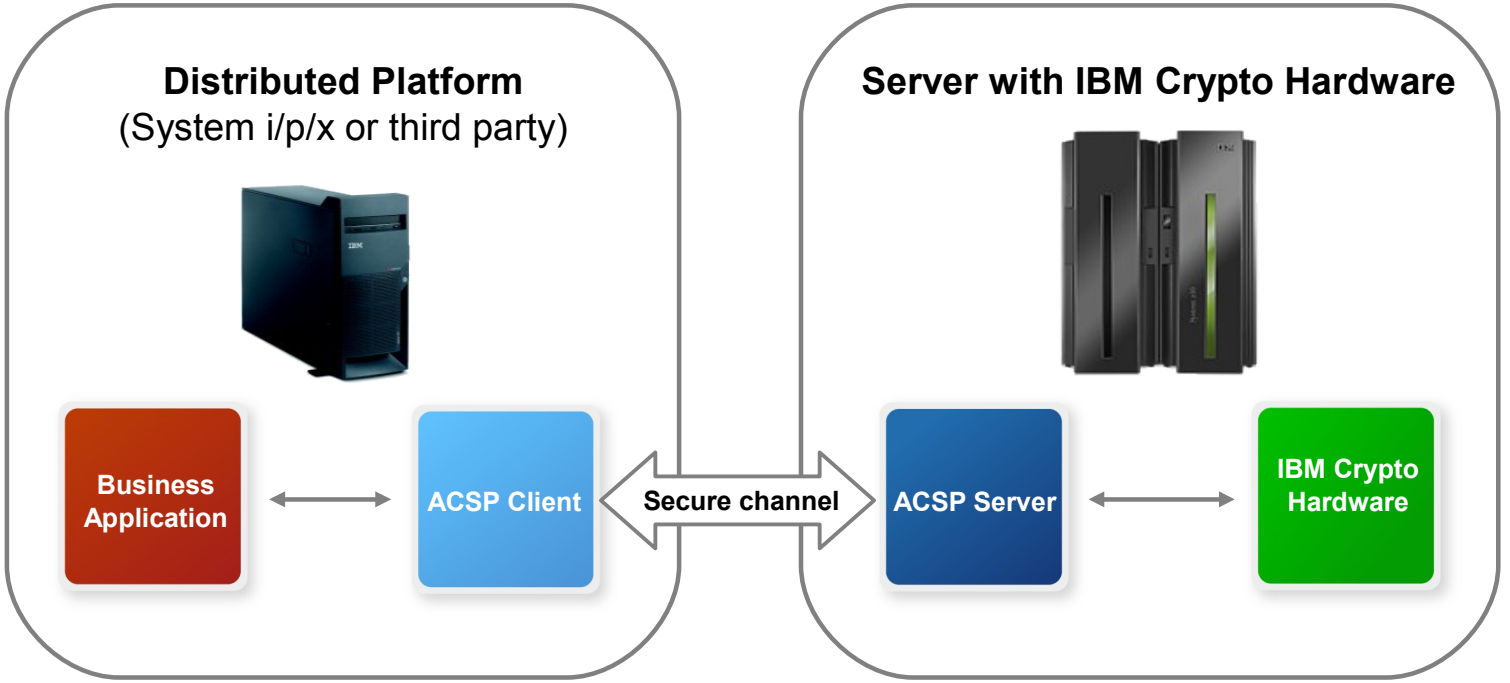
IBM DKMS ACSP
Advanced Crypto Service Provider

Crypto Competence
Center Copenhagen

Advanced Crypto Service Provider (ACSP) Vision

Vision: To capitalize on an existing scalable infrastructure to add security to new applications and platforms

"Mainframe centric security"



Advanced Crypto Service Provider
Application development and test

On the client side:

Programming the right CCA calls

Keys are in place

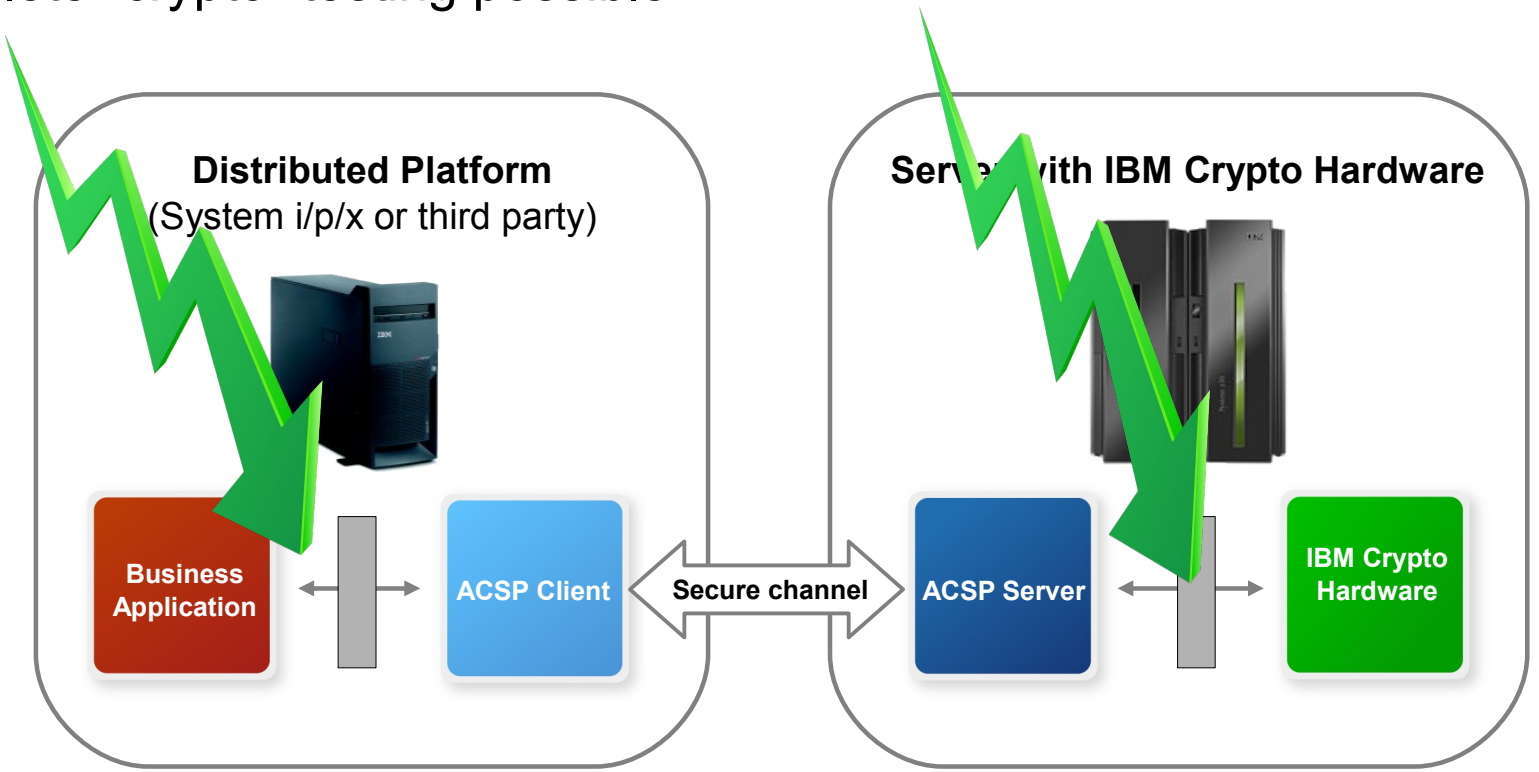
Complete “crypto” testing possible

On the server side:

The “right” function set available

Central key management

and access control



Advanced Crypto Service Provider Interfaces

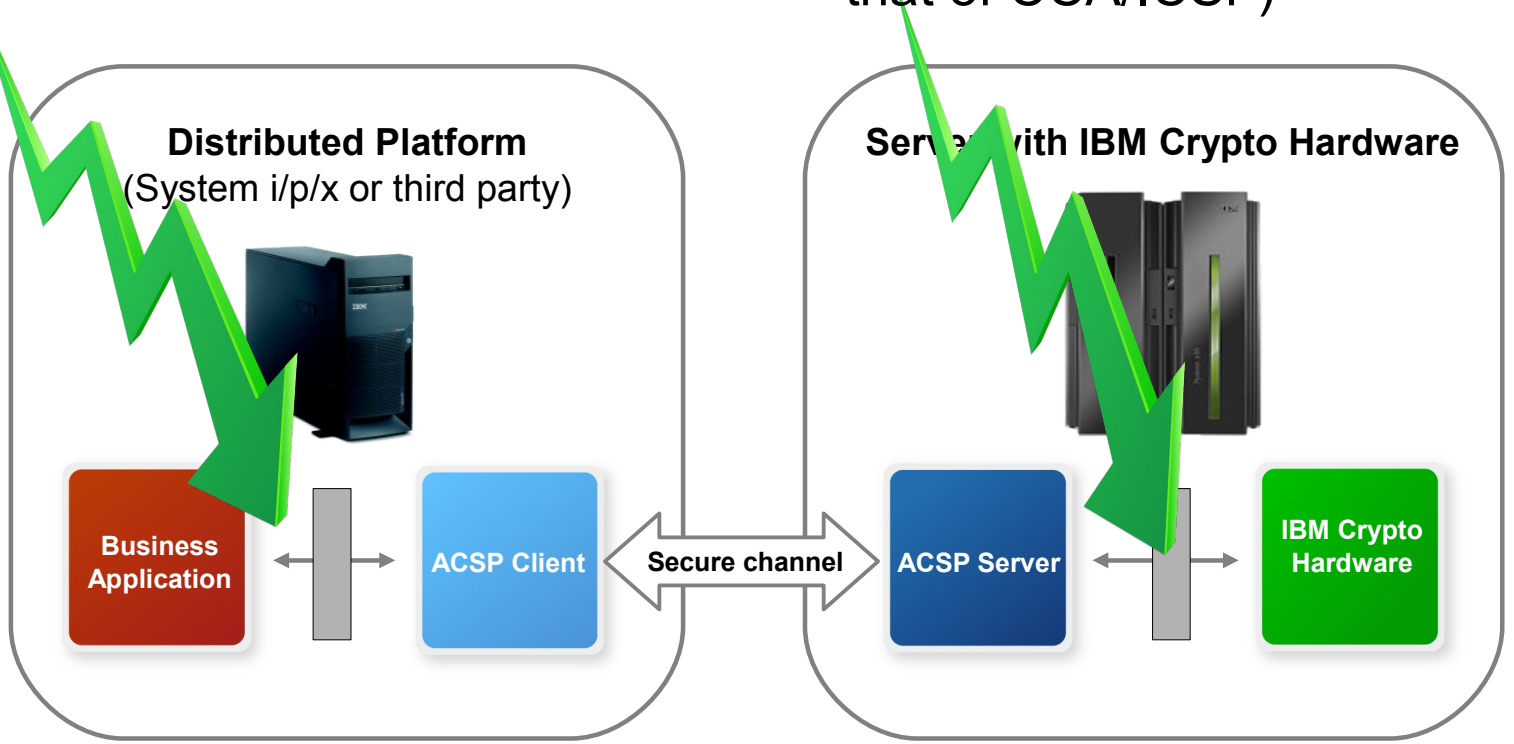
On the client side:

IBM CCA (C, Java), PKCS#11 JCE provider being developed

UDF and UDX supported

On the server side:

IBM CCA / ICSF (mapping JCE and PKCS#11 to IBM CCA functions and methods, which limits functionality to that of CCA/ICSF)



ACSP Interfaces and Functions supported

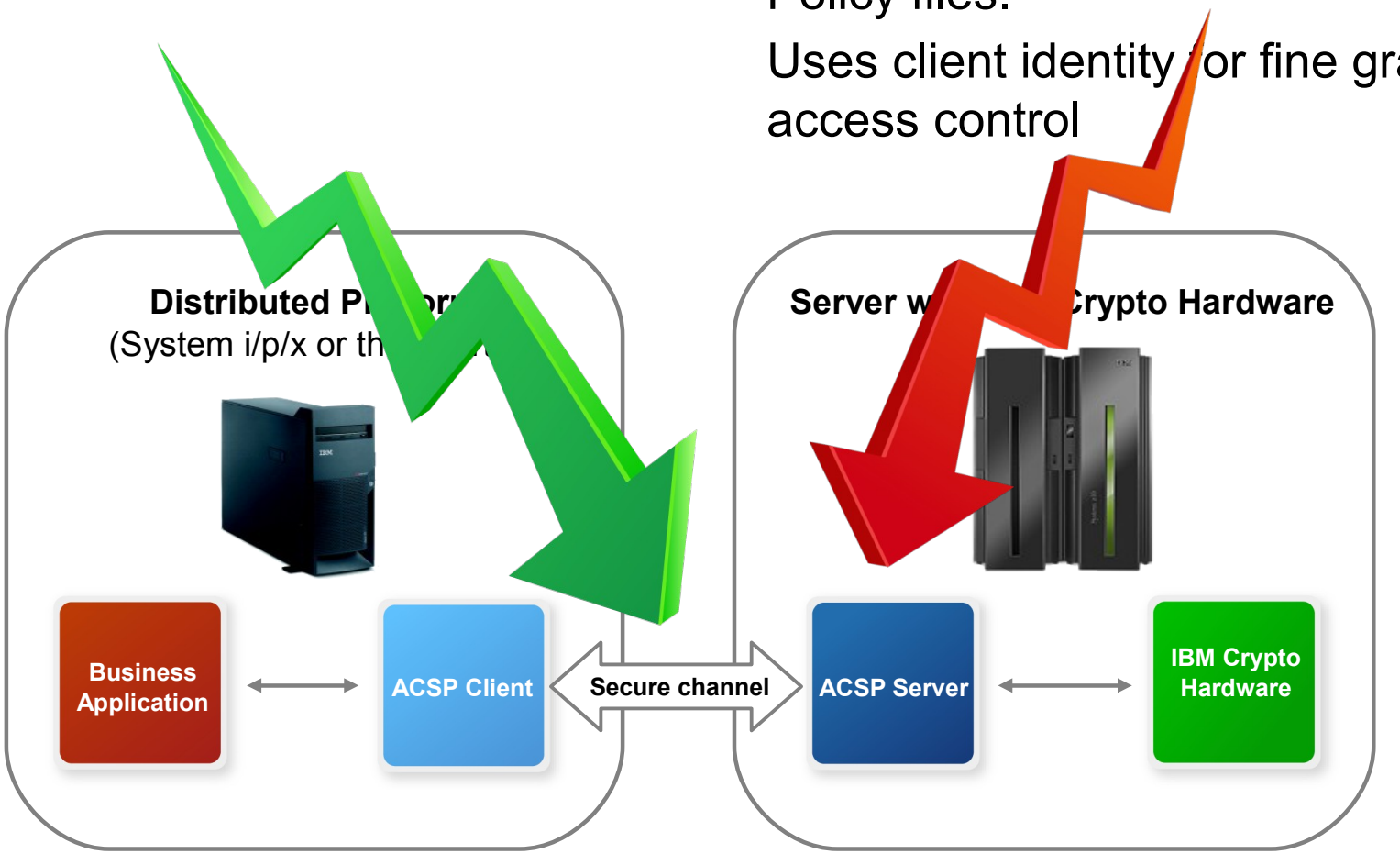
- IBM CCA – in C and Java
  - All CCA functions available
    (symmetric, asymmetric, hash, PIN,++)
  - UDX and UDF
- PKCS#11 v.2.20 (CCA based)
  - Reduced set of functions and algorithms supported
    (symmetric, asymmetric, hash,++)
- JCE (CCA based)
  - Digital signature related functions
- Integrates with applications in Java, C, C++, .Net C#

Advanced Crypto Service Provider Security

Secure channel: SSL with client authentication

RACF control of functions and keys available to the ACSP server task. Policy files.

Uses client identity for fine grained access control



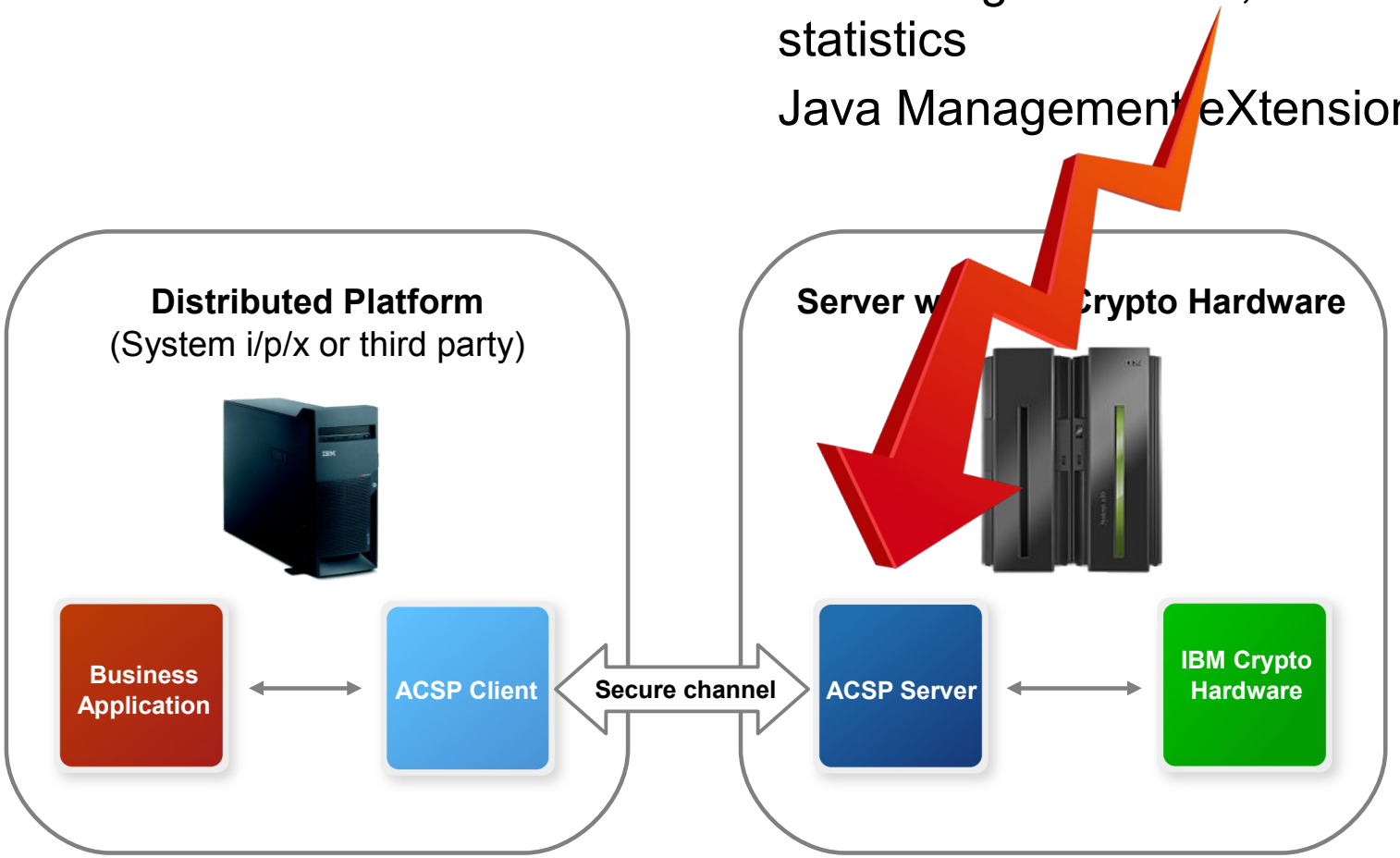
Advanced Crypto Service Provider Operations

No running service on client side

Started task (MVS command I/F)

Monitoring and alerts, zOS Syslog, statistics

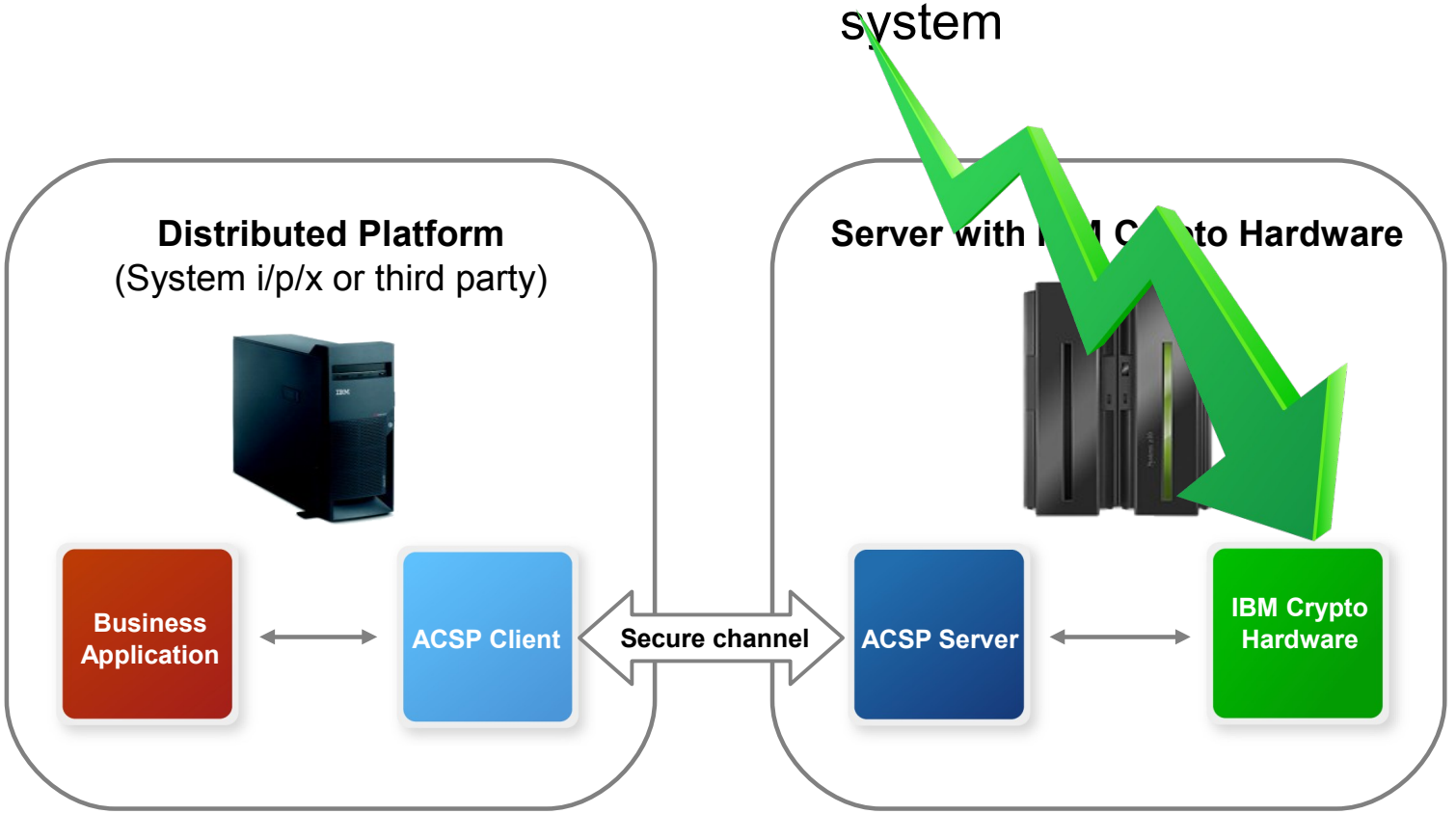
Java Management eXtensions



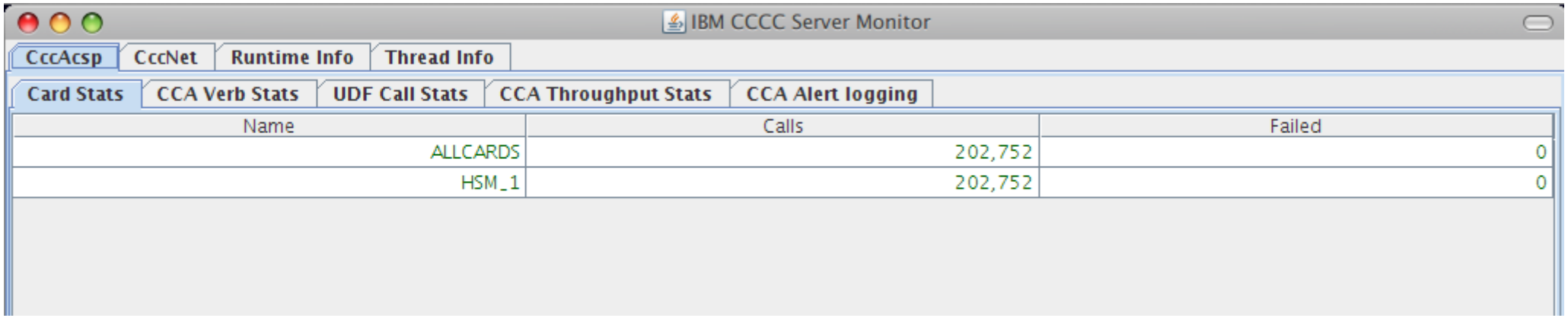
Advanced Crypto Service Provider
Key management

- Keys and certificates for SSL

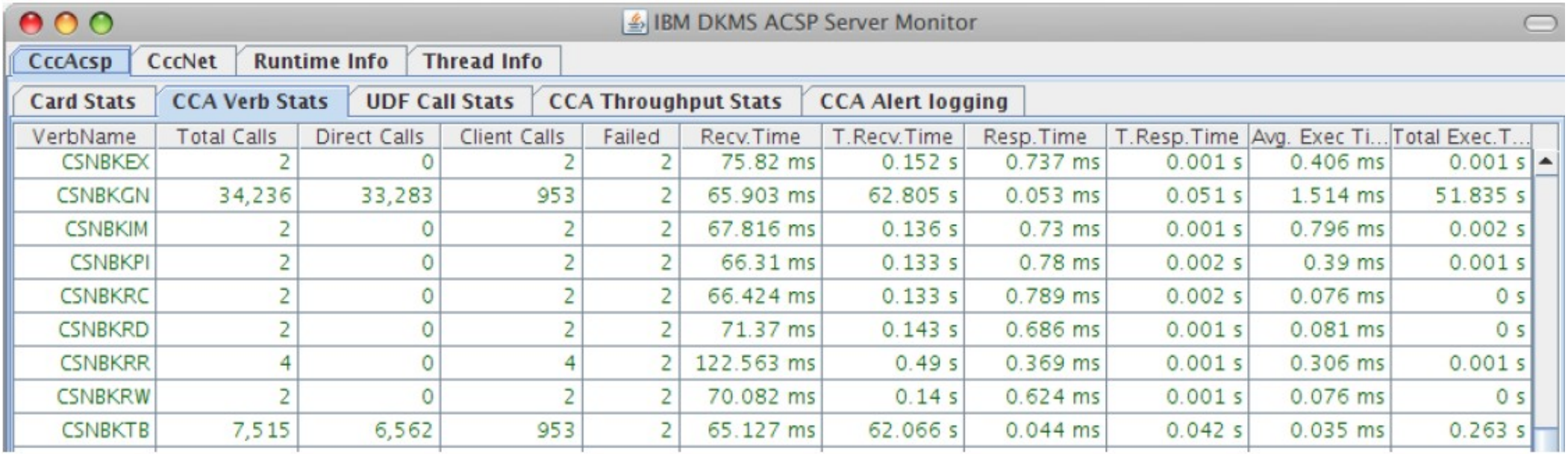
- Existing keys can be used
- Centralized management possible with IBM DKMS key management system



ACSP Monitoring

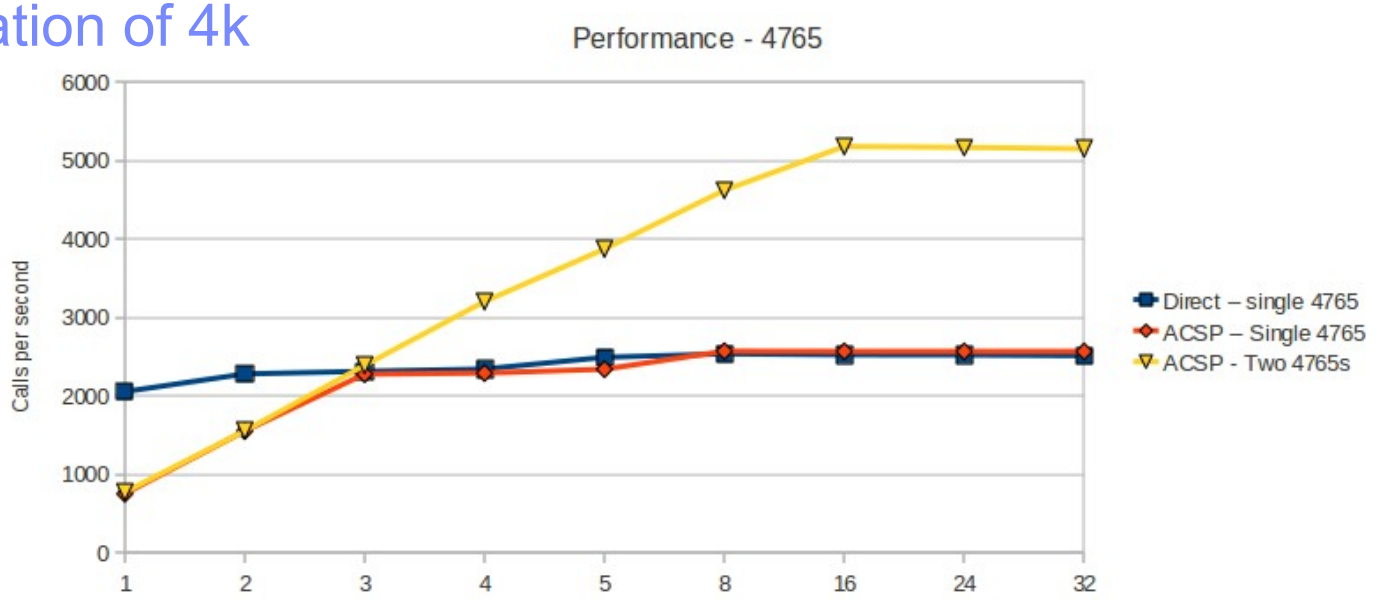


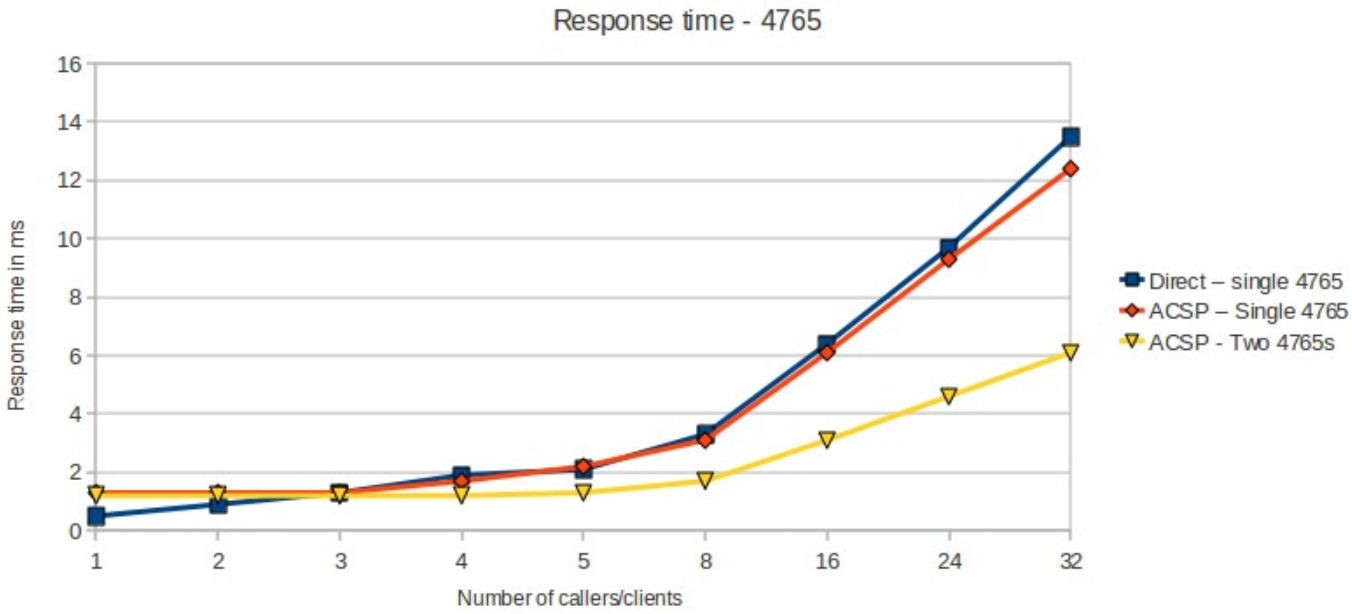
Name	Calls	Failed
ALLCARDS	202,752	0
HSM_1	202,752	0



VerbName	Total Calls	Direct Calls	Client Calls	Failed	Recv.Time	T.Recv.Time	Resp.Time	T.Resp.Time	Avg. Exec Ti...	Total Exec.T...
CSNBKEX	2	0	2	2	75.82 ms	0.152 s	0.737 ms	0.001 s	0.406 ms	0.001 s
CSNBKGN	34,236	33,283	953	2	65.903 ms	62.805 s	0.053 ms	0.051 s	1.514 ms	51.835 s
CSNBKIM	2	0	2	2	67.816 ms	0.136 s	0.73 ms	0.001 s	0.796 ms	0.002 s
CSNBKPI	2	0	2	2	66.31 ms	0.133 s	0.78 ms	0.002 s	0.39 ms	0.001 s
CSNBKRC	2	0	2	2	66.424 ms	0.133 s	0.789 ms	0.002 s	0.076 ms	0 s
CSNBKRD	2	0	2	2	71.37 ms	0.143 s	0.686 ms	0.001 s	0.081 ms	0 s
CSNBKRR	4	0	4	2	122.563 ms	0.49 s	0.369 ms	0.001 s	0.306 ms	0.001 s
CSNBKRW	2	0	2	2	70.082 ms	0.14 s	0.624 ms	0.001 s	0.076 ms	0 s
CSNBKTB	7,515	6,562	953	2	65.127 ms	62.066 s	0.044 ms	0.042 s	0.035 ms	0.263 s

ACSP Performance (x86 server - Linux and IBM 4765)
MAC generation of 4k





Advanced Crypto Service Provider (ACSP)
Summing up

- Utilizes existing system z infrastructure – cost efficient
- Scalable solution – uses multiple crypto processors with ICSF doing the load balancing
- High performance – efficient ACSP implementation

- More efficient “crypto “application development

- Central management – efficient operation and ensuring policy compliance
- IBM DKMS Key Management System can manage the keys and certificates

Leo Moesgaard (lemo@dk.ibm.com) – manager of IBM Crypto Competency Center
December 2011

Charts with various ACSP details

Crypto Competence
Center Copenhagen

ACSP Performance

- ACSP server configurations
  - x86 + Linux with IBM 4765
  - z990 with CEX2C
  - z196 with CEX3C
- ACSP client configuration
  - Multiple java applications using ACSP Java client performing CCA MAC (double length key, key label, 4k data)
- Measured throughput and response time for
  - Direct calls (no ACSP)
  - ACSP server with one crypto card
  - ACSP server with two crypto cards
- Network turnaround time about 0.8 ms

PKCS#11 functions – mapped to CCA functions

PKCS11	CCA Verb	Parameters
C_Digest	CSNBOWH	rule_array: ONLY SHA-1 SHA-256
C_Encrypt	CSNBENC CSNBSAE CSNDPKE	rule_array: see mapping section
C_Decrypt	CSNBDEC CSNBSAD CSNDPKD	rule_array same as above
C_Sign	CSNBMGN CSNBDSG	
C_Verify	CSNBMVR CNSBDSV	
C_GenerateKey	CSNBKTB CSNBKGN	
C_GenerateKeyPair	CSNDPKB CSNDPKG	
C_GenerateRandom	CSNBRNGL	
C_FindObjects	CSNBKRL CSNBAKRL CSNDKRL	key_label: '*.*.*.*.*.*.*' key_type; CKK_DES, CKK_AES, CKK_RSA

PKCS#11 mechanisms/algorithms – mapped to CCA

Algorithm	Type	CCA identifiers
CKM_RSA_PKCS	E	PKCS-1.2
CKM_RSA_X_509	S	
CKM_SHA1_RSA_PKCS	E	
CKM_DES3_ECB	E	DES
CKM_DES3_CBC	E	DES
CKM_DES3_CBC_PAD	E	DES X9.23
CKM_DES3_MAC_GENERAL	MS	TDES-MAC
CKM_DES3_MAC	MS	TDES-MAC
CKM_SHA_1	H	ONLY, SHA-1
CKM_SHA_1_HMAC	MS	blank
CKM_SHA_1_HMAC_GENERAL	MS	blank

Algorithm	Type	CCA identifiers
CKM_RSA_PKCS_OAEP	E	
CKM_AES_ECB	E	AES KEYIDENT
CKM_AES_CBC	E	AES CBC KEYIDENT
CKM_AES_CBC_PAD	E	AES KEYIDENT
CKM_AES_MAC	MS	AES
CKM_AES_MAC_GENERAL	MS	AES
CKM_SHA256	H	SHA-256
CKM_SHA256_HMAC	MS	TDES-MAC
CKM_SHA256_HMAC_GENERAL	MS	TDES-MAC
CKM_SHA256_RSA_PKCS	S	

Type: E=Encryption, S= signing, MS = MAC & signing, H= hashing