DB2 Security: From the Data Center to the Cloud

September 2, 2016

Presented by: Roger E. Sanders

Safe Harbor Statement

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON CURRENT THINKING REGARDING TRENDS AND DIRECTIONS, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. FUNCTION DESCRIBED HEREIN MY NEVER BE DELIVERED BY I BM. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

IBM, the IBM logo, ibm.com and DB2 are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Security Versus Compliance

While security and compliance are deeply intertwined, they are *not* the same:

- The purpose of **security** is to ensure that something is only accessed by identified individuals who have been authorized to access it, and that critical accesses are tracked for future accountability
- The purpose of **compliance** is to ensure that specific controls for how a system or solution is managed are put in place, and that adherence to those controls can be proven

Mechanisms DB2 Uses To Secure Data

DB2 uses the following mechanisms to secure data:

- Authentication
- Authorities and Privileges
- Trusted Contexts and Trusted Connections
- Row and Column Access Control (RCAC)
- Label-Based Access Control (LBAC)
- Native Encryption

Authentication

The first security portal users must pass through to gain access to a DB2 instance or database is ***authentication***. The purpose of authentication is to *verify that a user really is who they say they are*.

Usually, authentication is performed by a security facility that sits outside of DB2. DB2 communicates with this facility through an authentication *security plug-in module*, which is a dynamically loadable library that provides authentication security services.

Often, a user ID and password must be presented before a user can be authenticated.

Security Plug-In Modules

In most cases, the following security plug-in modules are available:

- Operating system
  - CLIENT
  - SERVER
  - SERVER_ENCRYPT
  - DATA_ENCRYPT
  - DATA_ENCRYPT_CMP
- Kerberos
  - KERBEROS
  - KRB_SERVER_ENCRYPT
- Generic Security Service API (GSSAPI)
  - GSSPLUGIN
  - GSS_SERVER_ENCRYPT

How and where authentication takes place is usually determined by the value assigned to the **AUTHENTICATION** *database manager configuration parameter.*

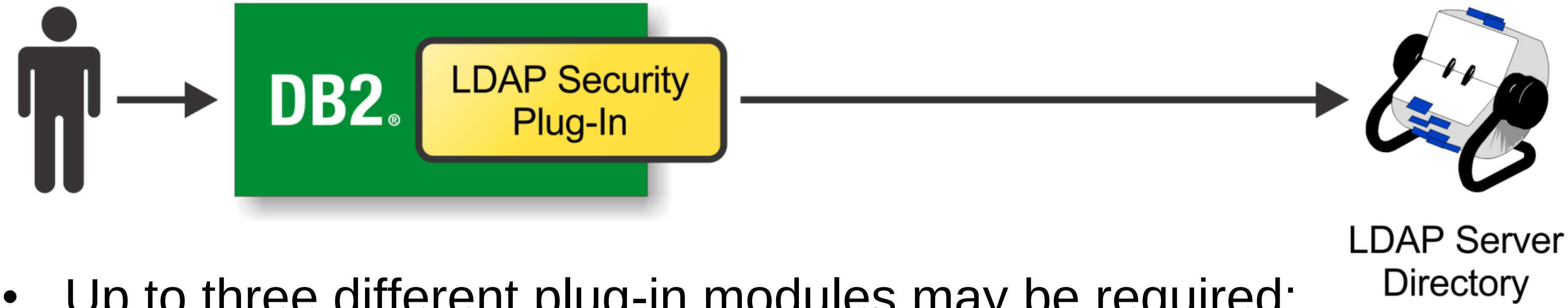
Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) is a directory service protocol that runs on a layer above the TCP/IP stack. A common use of LDAP is to provide a central place to store user IDs and passwords – this allows many different applications and services to validate users by connecting to a single LDAP server.

LDAP is another technology that can be used to authenticate DB2 users; two methods of LDAP authentication are available:

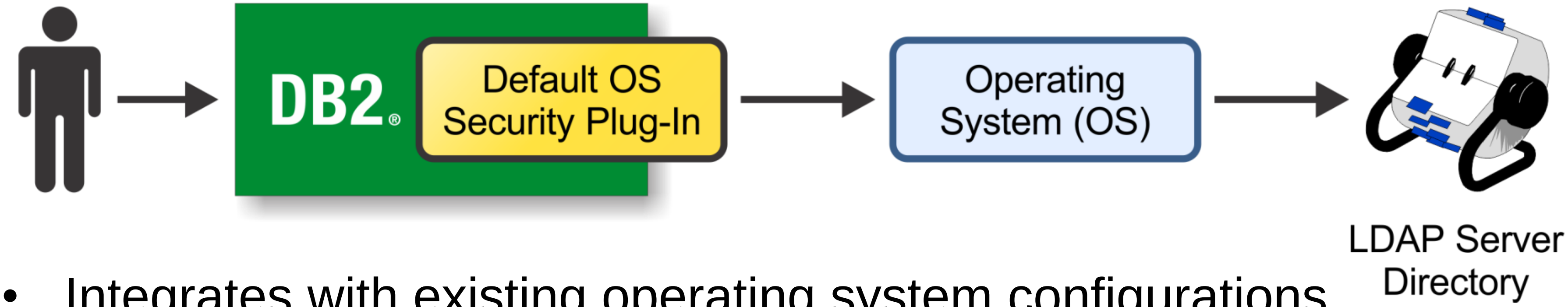
- **LDAP Security Plug-In Authentication**
- **Transparent LDAP Authentication**

LDAP Security Plug-In Authentication



- Up to three different plug-in modules may be required:
  - One for server-side authentication, one for client-side authentication, and one for group lookup
- All users, as well as any groups that are required for authentication, must be defined on the LDAP server
  - This includes both the DB2 instance owner *and* the fenced user
  - This also includes the SYSADM, SYSMAINT, SYSCTRL and SYSMON groups that are defined in the database manager configuration file

Transparent LDAP Authentication



- Integrates with existing operating system configurations
  - DB2 will authenticate users and acquire their groups through the OS
  - The OS will, in turn, perform authentication through an LDAP server
- Easy to implement
  - Configure the OS to authenticate with Pluggable Authentication Modules (PAMs)
  - Set the **DB2AUTH** registry variable to **OSAUTHDB**

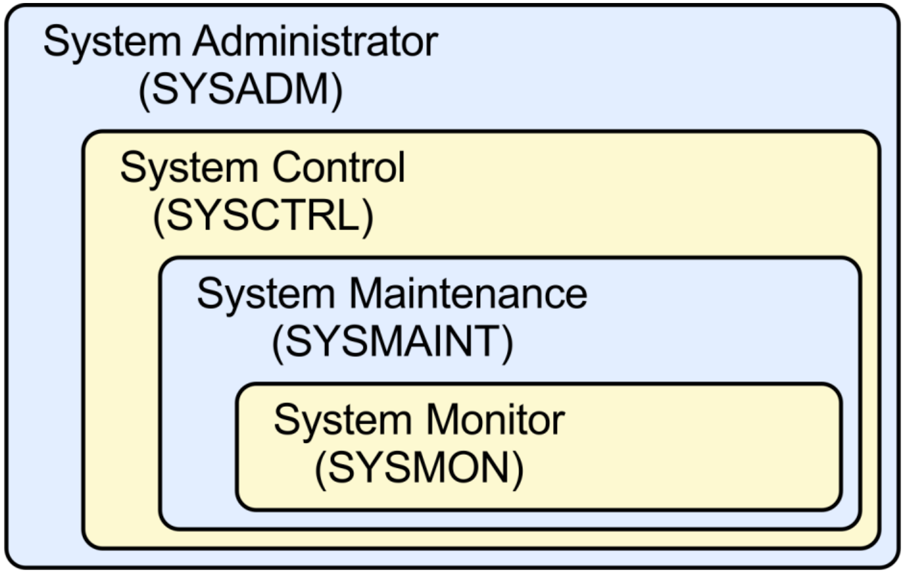
Authorities and Privileges

Once a user has been authenticated and an attachment to an instance or a connection to a database has been made, the DB2 database manager evaluates any *authorities* and/or *privileges* the user holds to determine what operations he or she is allowed to perform.

Authorities convey the right to perform high-level administrative or maintenance operations against a particular instance or database; **Privileges** convey the right to perform certain actions against database objects (such as tables, views, and indexes).

Instance-Level Authorities

Instance-level authorities enable a user to perform instance-related functions, such as creating and upgrading databases, managing table spaces, and monitoring performance; the following instance-level authorities are available:


System Administrator (SYSADM)

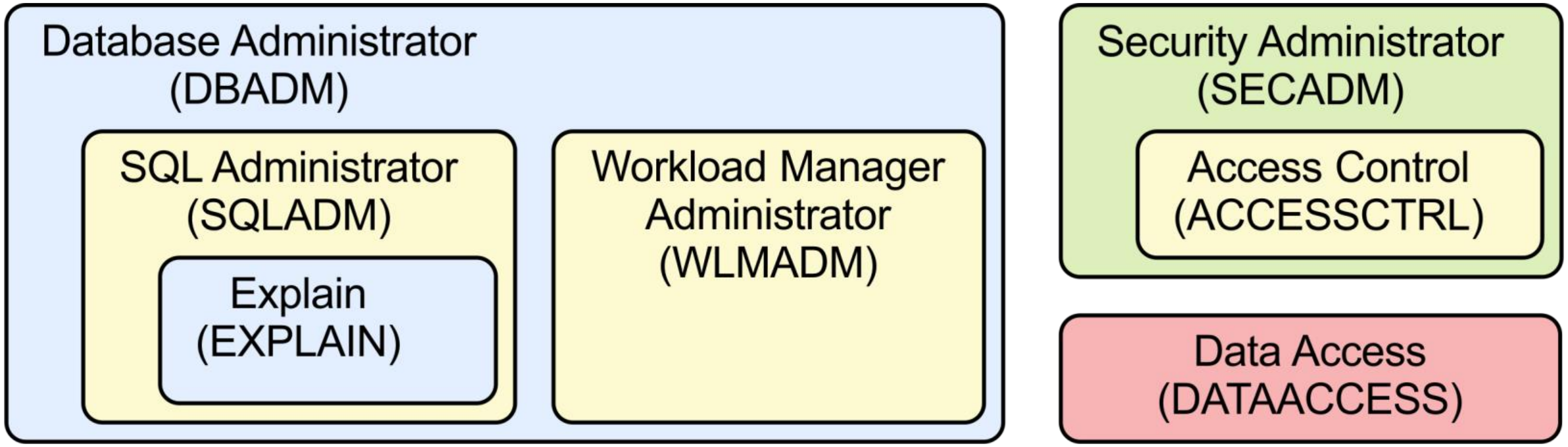
System Control (SYSCTRL)

System Maintenance (SYSMAINT)

System Monitor (SYSMON)

Database-Level Authorities

Database-level authorities enable a user to perform functions within a database, such as granting and revoking privileges and inserting, updating, deleting and retrieving data; the following database-level authorities are available:


Database Administrator (DBADM)
SQL Administrator (SQLADM)
Explain (EXPLAIN)
Workload Manager Administrator (WLMADM)
Security Administrator (SECADM)
Access Control (ACCESSCTRL)
Data Access (DATAACCESS)

A Word About Security Administrator (SECADM) Authority

Security Administrator (SECADM) authority is a database-level authority that allows select individuals to create and manage security-related objects, as well as grant and revoke database-level authorities and object privileges. Individuals with SECADM authority can also execute audit system routines, as well as grant that ability to others.

The purpose of SECADM authority is to clearly separate the duties of system administrators, database administrators, and security administrators, as well as limit who is allowed to perform security-related tasks. It also prevents administrators from “automatically” getting access to database data.

Object Privileges

Privileges enable a user to perform a specific action or task, such as executing a package or routine; one or more privileges exist for the following objects:

- Databases
- Global variables
- Schemas
- Table spaces
- Tables
- Indexes
- Views
- Sequences
- Modules
- Routines
- Packages
- Workloads
- Nicknames
- Servers
- XSR objects

Obtaining (and Losing) Authorities and Privileges

There are three ways in which users can obtain authorities and/or privileges:

- **Indirectly** (for example, when a package is executed)
- **Implicitly** (for example, when a new object is created)
- **Explicitly** (when they are *granted* by someone with the appropriate authority)

Users can lose individual authorities and privileges when they are explicitly *revoked*. The **GRANT** statement is used to explicitly give authorities and privileges to others; the **REVOKE** statement is used to take them away.

A Word About Roles

A **role** is a database object that is used to group one or more authorities and/or privileges together. Once created, a role can be granted to users, groups, PUBLIC, or other roles.

All the roles that have been assigned to a user are enabled when that user establishes a connection; all the authorities and privileges that are associated with those roles are taken into account when DB2 checks user authorization.

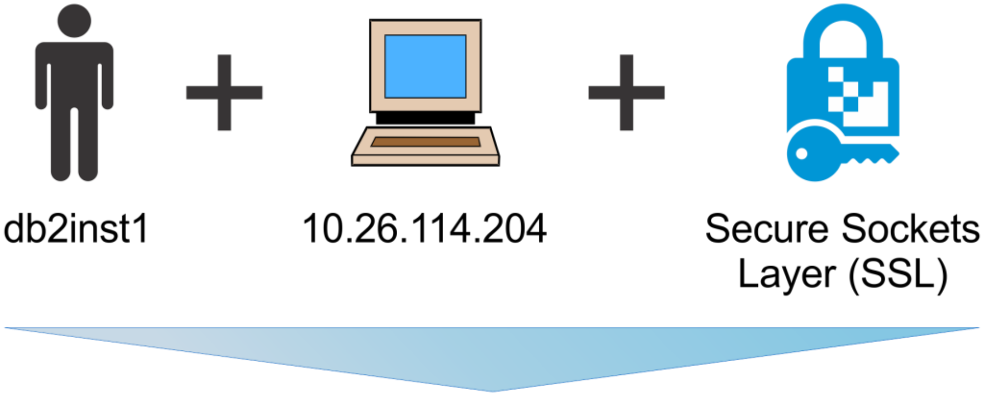
Roles are managed by someone with SECADM authority; however, management can be delegated to others via the WITH ADMIN OPTION of the GRANT ROLE statement.

Trusted Contexts

A **trusted context** is a database object that defines a “trust relationship” for a connection between a DB2 database and an external entity such as an application server. Trust relationships are built upon the following attributes:

- **System authorization ID** – the user ID that establishes a database connection
- **IP address or domain name** – the host from which a database connection is established
- **Data stream encryption** – the encryption setting used, if any, for communications between the database server and the client

Creating A Trusted Context



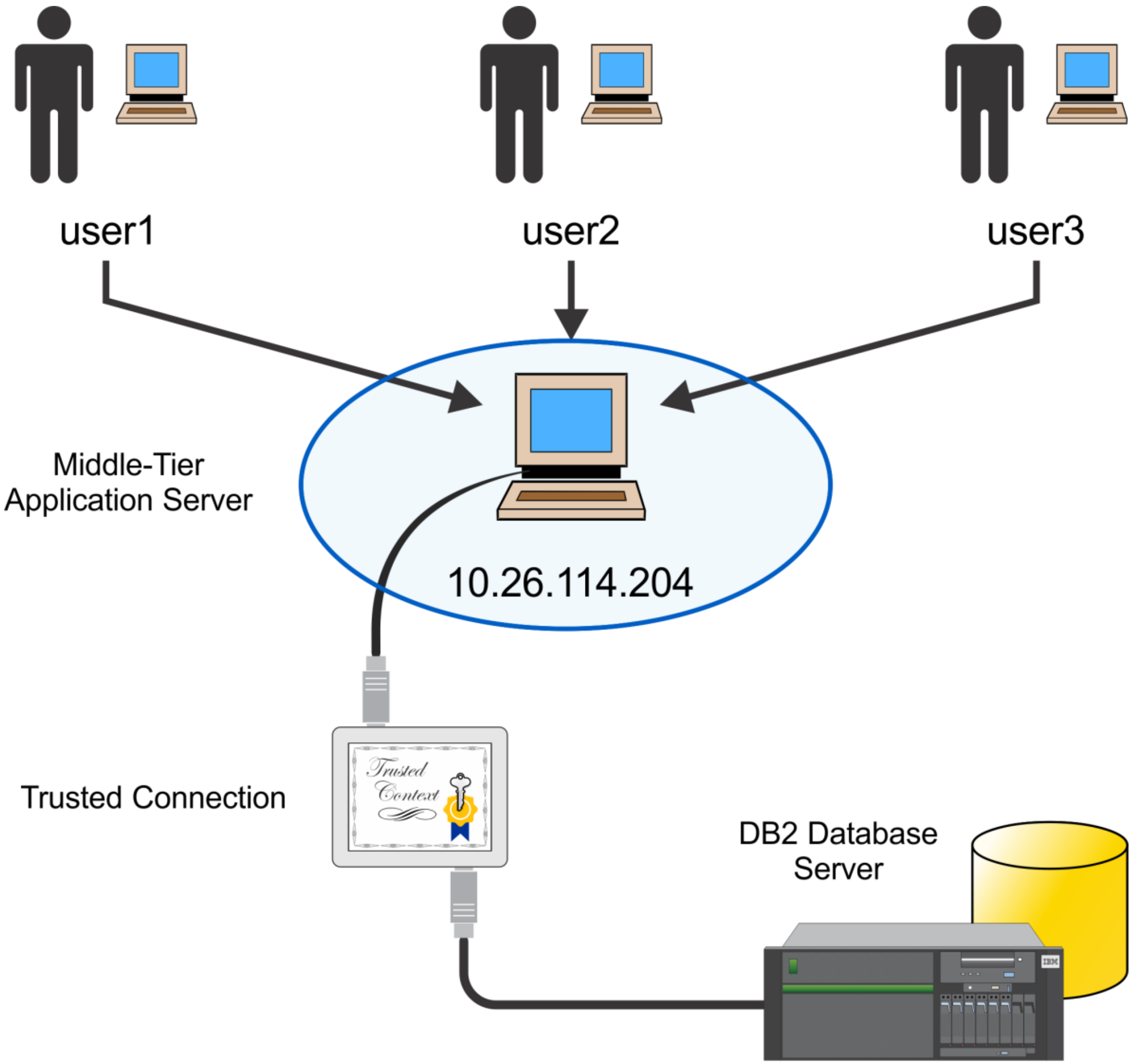
```
CREATE TRUSTED CONTEXT appserver
BASED UPON CONNECTION USING SYSTEM AUTHID db2inst1
ATTRIBUTES (ADDRESS '10.26.114.204' ENCRYPTION 'HIGH')
DEFAULT ROLE sales
WITH USE FOR user1, user2, user3
```

Trusted Connections

When a user attempts to connect to a database, DB2 checks to see if the connection matches a trusted context definition. If a match is found, the connection is said to be a "**trusted connection**." Two types of trusted connections can exist:

- **Implicit** – results from a normal connection request; allows a user to inherit a role that is not available to them outside the scope of the trusted connection
- **Explicit** – established by making a connection request within an application; allows an application to switch the connection's user to a different authentication ID without having to re-authenticate the new user at the database
  - Optionally, switching can require authentication (a password)

Trusted Connections – Illustrated



Row and Column Access Control (RCAC)

Row and Column Access Control (**RCAC**) is a security mechanism that can be used to restrict access to a table at the row level, the column level, or both; RCAC offers the following advantages over the authorities/privileges security model:

- Users are allowed to access only the data that is needed to perform a particular job/task
- Table data is protected regardless of how it is accessed by SQL
- When a result set is restricted due to RCAC, no warnings or errors are returned
- No database user, including a user with DATAACCESS authority, is inherently exempted from RCAC behavior

Row and Column Access Control Rules

Under RCAC, access to a table is restricted according to a set of rules that have been specified by a policy that is associated with the table. Two sets of rules exist – one operates on rows, the other on columns. They are:

- **Row permissions** – a database object that identifies an SQL search condition that describes what set of rows a user has access to
- **Column masks** – a database object that consists of an SQL CASE expression that describes what column values a user is permitted to see, and under what conditions they are permitted to see them

Creating Row Permissions

1. Patients can only access their own data.
2. Doctors can only access their patients’ data.
3. No one else can access patient data.

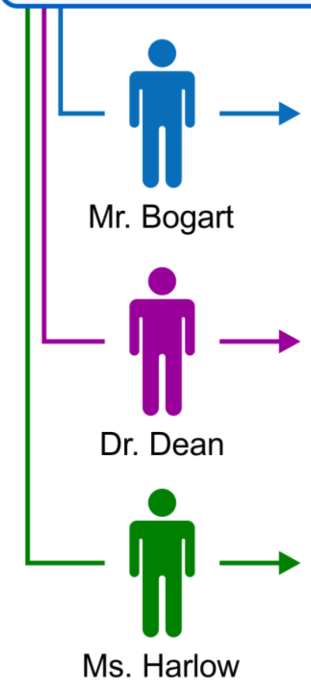
```
    CREATE PERMISSION row_access ON er.med_recs
    FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER, ‘PATIENT’) = 1
1      AND er.med_recs.patient_id = SESSION_USER)
    OR
      (VERIFY_ROLE_FOR_USER(SESSION_USER, ‘DOCTOR’) = 1
2      AND er.med_recs.doctor_id = SESSION_USER)
    ENFORCED FOR ALL ACCESS
    ENABLE;

    ALTER TABLE er.med_recs ACTIVATE ROW ACCESS CONTROL;
```

Putting Row Permissions To Work

ER.MED_RECS TABLE

PATIENT_ID	L_NAME	CHART_NO	MEDICATION	DOCTOR_ID
HBOG	BOGART	1025	LISINOPRIL	JDEAN
JCAG	CAGNEY	1908	NAPROXEN	CGABLE
KHEP	HEPBURN	2107	BENZOCAINE	JDEAN
MMON	MONROE	1845	AMPICILLIN	JDEAN
CGRA	GRANT	1560	TETRACYCLINE	CGABLE

```
SELECT * FROM er.med_recs
```



PATIENT_ID	L_NAME	CHART_NO	MEDICATION	DOCTOR_ID
HBOG	BOGART	1025	LISINOPRIL	JDEAN

PATIENT_ID	L_NAME	CHART_NO	MEDICATION	DOCTOR_ID
HBOG	BOGART	1025	LISINOPRIL	JDEAN
KHEP	HEPBURN	2107	BENZOCAINE	JDEAN
MMON	MONROE	1845	AMPICILLIN	JDEAN

PATIENT_ID	L_NAME	CHART_NO	MEDICATION	DOCTOR_ID

Creating Column Masks

1. Individuals who work in the PAYROLL department can see employee SSNs.
2. Developers can only see the last four numbers of employee SSNs.
3. No one else can see employee SSN data.

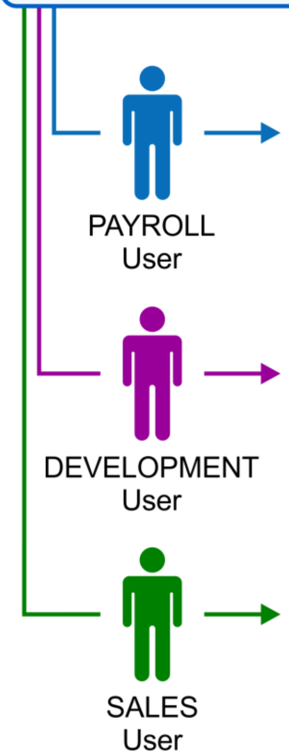
```
    CREATE MASK ssn_mask ON hr.employees
    FOR COLUMN ssn RETURN
      CASE
1       WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, ‘PAYROLL’) = 1
          THEN ssn
2       WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, ‘DEVELOPMENT’) = 1
          THEN ‘XXX-XX-’ || SUBSTR(ssn, 8, 4)
3       ELSE NULL
      END
    ENABLE;

    ALTER TABLE hr.employees ACTIVATE COLUMN ACCESS CONTROL;
```

Putting Column Masks To Work

HR.EMPLOYEES TABLE

EMP_ID	NAME	ADDRESS	SSN
104	BOGART, HUMPHREY	112 BEACON STREET	278-83-9437
283	CAGNEY, JAMES	705 HAUSER STREET	230-51-3482
112	HEPBURN, KATHARINE	221B BAKER STREET	392-57-1635
347	MONROE, MARILYN	1313 MOCKINGBIRD LANE	193-42-6008

```
SELECT * FROM hr.employees
```



EMP_ID	NAME	ADDRESS	SSN
104	BOGART, HUMPHREY	112 BEACON STREET	278-83-9437
283	CAGNEY, JAMES	705 HAUSER STREET	230-51-3482
112	HEPBURN, KATHARINE	221B BAKER STREET	392-57-1635
347	MONROE, MARILYN	1313 MOCKINGBIRD LANE	193-42-6008

EMP_ID	NAME	ADDRESS	SSN
104	BOGART, HUMPHREY	112 BEACON STREET	XXX-XX-9437
283	CAGNEY, JAMES	705 HAUSER STREET	XXX-XX-3482
112	HEPBURN, KATHARINE	221B BAKER STREET	XXX-XX-1635
347	MONROE, MARILYN	1313 MOCKINGBIRD LANE	XXX-XX-6008

EMP_ID	NAME	ADDRESS	SSN
104	BOGART, HUMPHREY	112 BEACON STREET	-
283	CAGNEY, JAMES	705 HAUSER STREET	-
112	HEPBURN, KATHARINE	221B BAKER STREET	-
347	MONROE, MARILYN	1313 MOCKINGBIRD LANE	-

Using RCAC With Views, User-Defined Functions, and Triggers

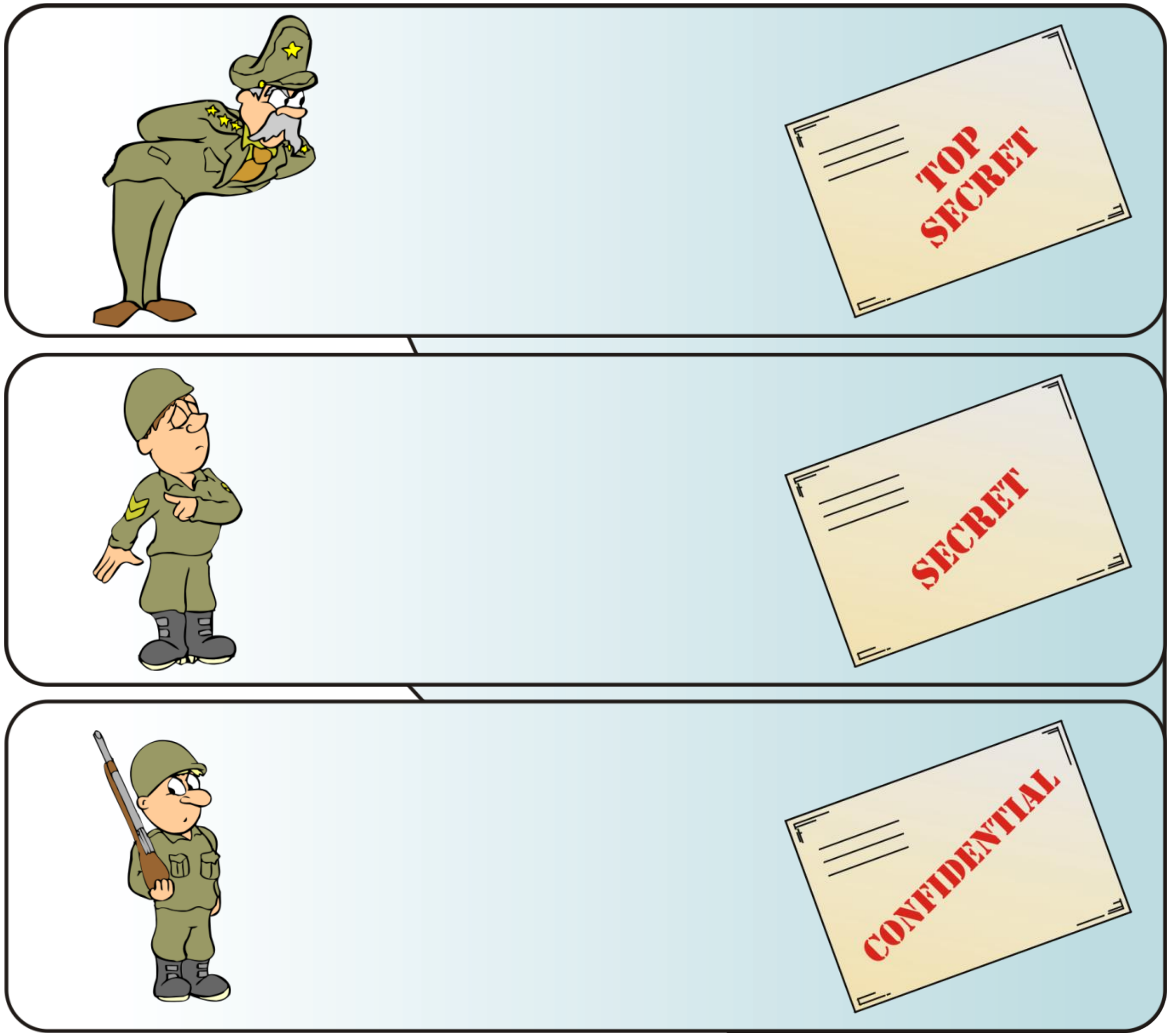
- Views can be created on RCAC-protected tables
  - When querying the view, data is returned based on any RCAC rules that have been defined on the underlying base table(s)
- User-defined functions (UDFs) that *are referenced in a row permission or column mask definition* must be defined as being SECURED
- UDFs that *are to be used against columns that are protected with a column mask* must be defined as being SECURED
- Triggers must be defined as being SECURED if the subject table is RCAC-protected

Discretionary Access Control Versus Mandatory Access Control

According to the *Common Criteria for Information Technology Security Evaluation* international standard (referred to as *Common Criteria* or *CC*), there are two ways to control data access:

- **Discretionary Access Control** – enforces a consistent set of rules for controlling and limiting access based on identified individuals who “need-to-know” the information
- **Mandatory Access Control** – enforces access control rules based directly on an individual's clearance, authorization for the information being sought, and the confidentiality level of the information being sought

An Example Of Mandatory Access Control



Label-Based Access Control

Label-Based Access Control (**LBAC**) is an implementation of Mandatory Access Control (MAC) at both the row and the column level. LBAC has the following MAC attributes:

- Data that must be protected is assigned a security label.
- Only security administrators (SECADM) are allowed to make changes to a resource's security label.
- Users are given access to data that is protected by a security label, according to the security labels they possess, LBAC rules, and any rule exemptions they may hold.

Setting Up An LBAC-Protected Environment

To construct an LBAC-protected system, a user with Security Administrator (SECADM) authority must:

- Create one or more **security label components**
- Create one or more **security labels**
- Create one or more **security policies**
- Grant appropriate security labels to users, groups, and/or roles

Then, a user with **proper LBAC credentials** must configure one or more tables for LBAC protection, as well as associate an appropriate security policy with the tables that are to be protected.

Security Label Components

Security label components represent criteria that will be used to decide who can access data. Three types of security label components are available:

- **Set** – a collection of elements (values) where the order in which each element appears is not important
- **Array** – an ordered set that can represent a simple hierarchy; the first element ranks higher than the second, the second ranks higher than the third, and so on
- **Tree** – a complex hierarchy that can have multiple nodes and branches

Security Labels

Security labels describe a set of security criteria and are used to protect data against unauthorized access or modification.

Security labels are granted to users who are allowed to access or modify protected data – when users attempt to perform these operations, their security label is compared to the security label that is protecting the data to determine whether the operation is allowed or denied.

Security Policies

Security policies determine exactly how a table is to be LBAC-protected. Specifically, a security policy identifies:

- Which security label components will be used with the security labels that are part of the policy
- What rules will be used when security label component elements are compared
- Which optional behaviors will be used when data protected by the security policy is accessed

Protecting Rows With LBAC

A table can be configured for row-level LBAC protection by adding a column with the **SYSPROC.DB2SECURITYLABEL** data type to its definition and associating a security policy with the table:

```
CREATE TABLE corp.sales (
  sales_rec_id  INTEGER NOT NULL,
  ...
  sec_label     SYSPROC.DB2SECURITYLABEL)
SECURITY POLICY sec_policy

ALTER TABLE corp.sales
 ADD COLUMN sec_label DB2SECURITYLABEL
 ADD SECURITY POLICY sec_policy
```

Protecting Columns With LBAC

A table can be configured for column-level LBAC protection by adding the **SECURED WITH [*Label*]** option to a column’s definition and associating a security policy with the table:

```
CREATE TABLE hr.employees (
  ssn CHAR(12) SECURED WITH confidential)
  ...
SECURITY POLICY sec_policy

ALTER TABLE hr.employees
 ALTER COLUMN ssn SECURED WITH confidential
 ADD SECURITY POLICY sec_policy
```

Encryption

Encryption is the process of transforming data into an unintelligible form so it cannot be obtained or it can only be obtained through a decryption process.

Encryption provides an effective way of protecting sensitive information that is stored on media (often referred to as data at rest) or that is transmitted through untrusted communication channels (referred to as data in transit).

Encryption is performed using an industry standards-compliant **cryptographic algorithm** and a **key**. The key, along with the algorithm, is also used to recover the original data.

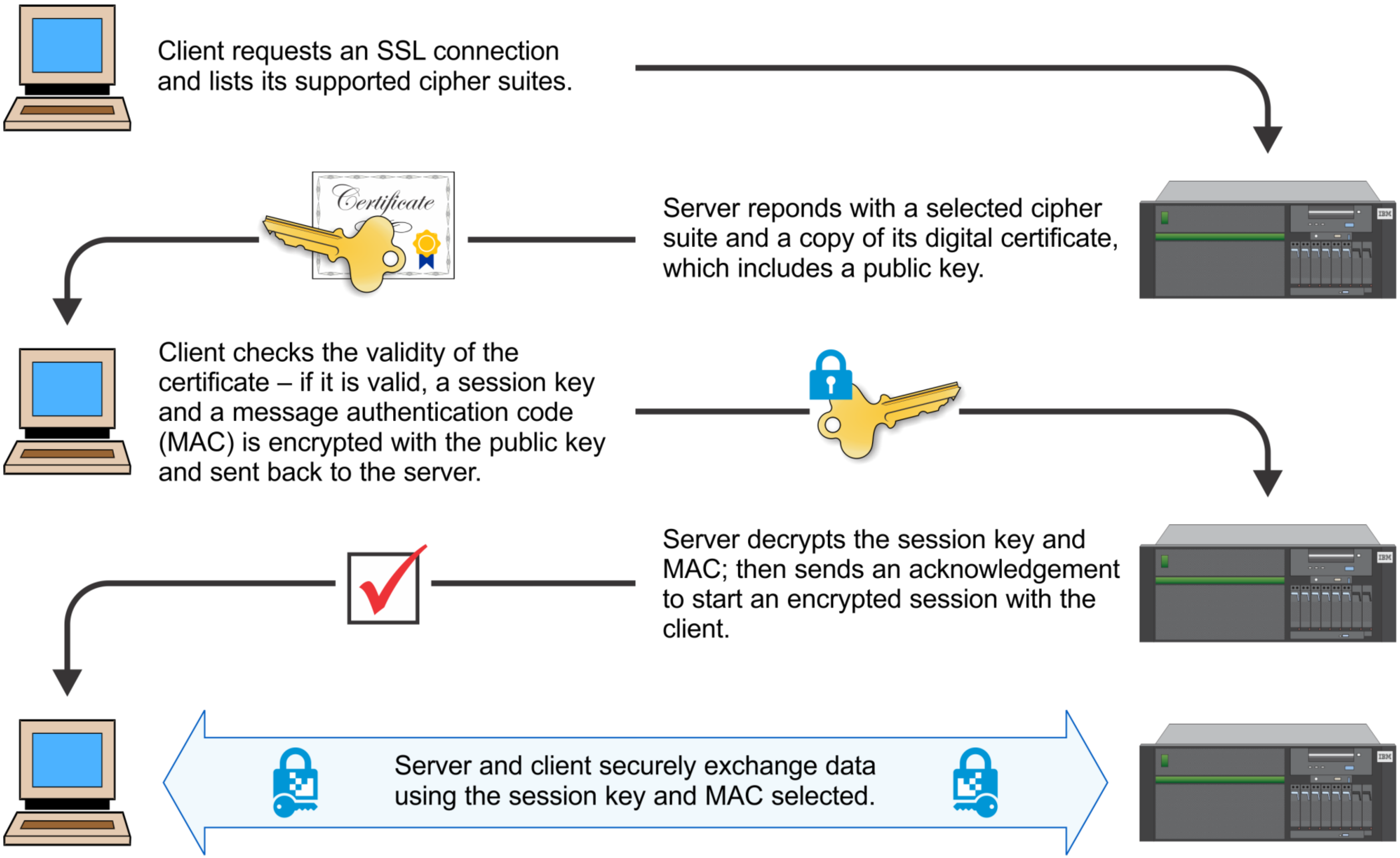
Encrypting Data In Transit

The **DATA_ENCRYPT** authentication type can be used to encrypt data in transit.

However, if you need to meet specific compliance or privacy standards, a better approach is to use **Secure Sockets Layer (SSL)** technology. SSL is a standard security protocol that is used to establish an encrypted link between a client and a server; it provides both data encryption and data integrity.

SSL utilizes industry standard encryption technology like the **Advanced Encryption Standard (AES)** cipher or the **Triple Data Encryption Standard (3DES)** cipher with 128-bit (or more) keys.

How SSL Works With DB2



Client requests an SSL connection and lists its supported cipher suites.

Server reponds with a selected cipher suite and a copy of its digital certificate, which includes a public key.

Client checks the validity of the certificate – if it is valid, a session key and a message authentication code (MAC) is encrypted with the public key and sent back to the server.

Server decrypts the session key and MAC; then sends an acknowledgement to start an encrypted session with the client.

Server and client securely exchange data using the session key and MAC selected.

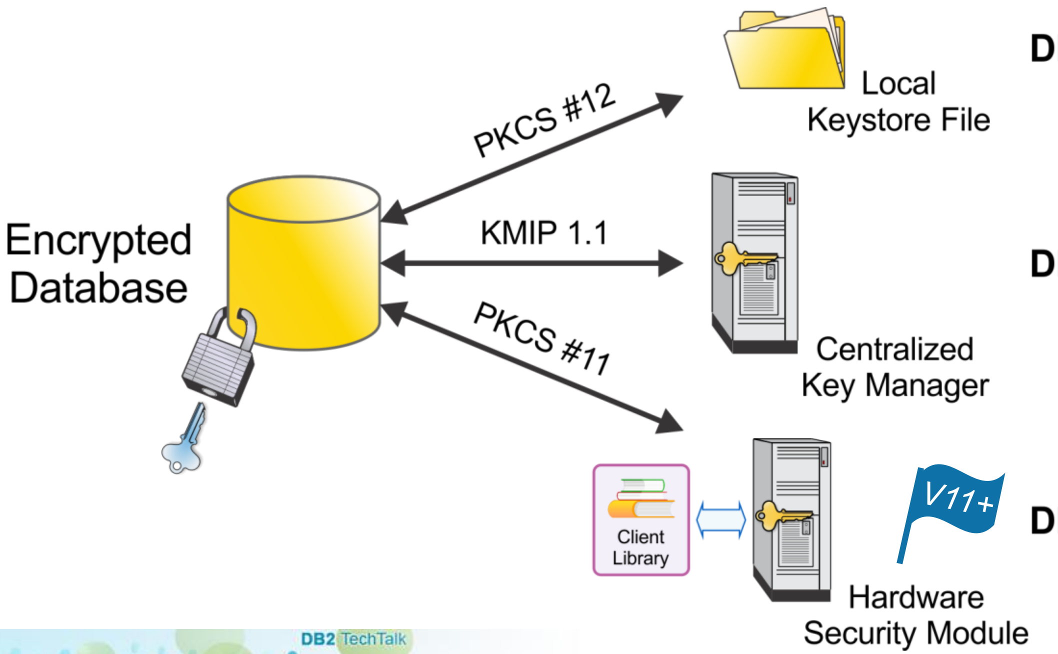
Encrypting Data At Rest

Data at rest encryption is performed using **DB2 Native Encryption** technology. This technology, which is implemented within the DB2 kernel itself, encrypts data *as it is written to disk*. As a result, sensitive data is *kept secure at all times*.

DB2 Native Encryption uses a symmetric encryption scheme (AES or 3DES) to encrypt both the database and any backup images created. It is transparent to applications and schemas. More importantly, it exploits Intel AES-NI hardware acceleration or native, in-core Power 8 vector instructions for AES encryption/de-encryption to minimize its impact on performance.

DB2 Native Encryption – Key Management

DB2 Native Encryption uses an industry standard, 2-tier model for key management – data is encrypted with a Data Encryption Key (DEK) and the DEK is encrypted with a Master Key (MK) that can be stored in a variety of locations:



DB2 v10.5 FP5

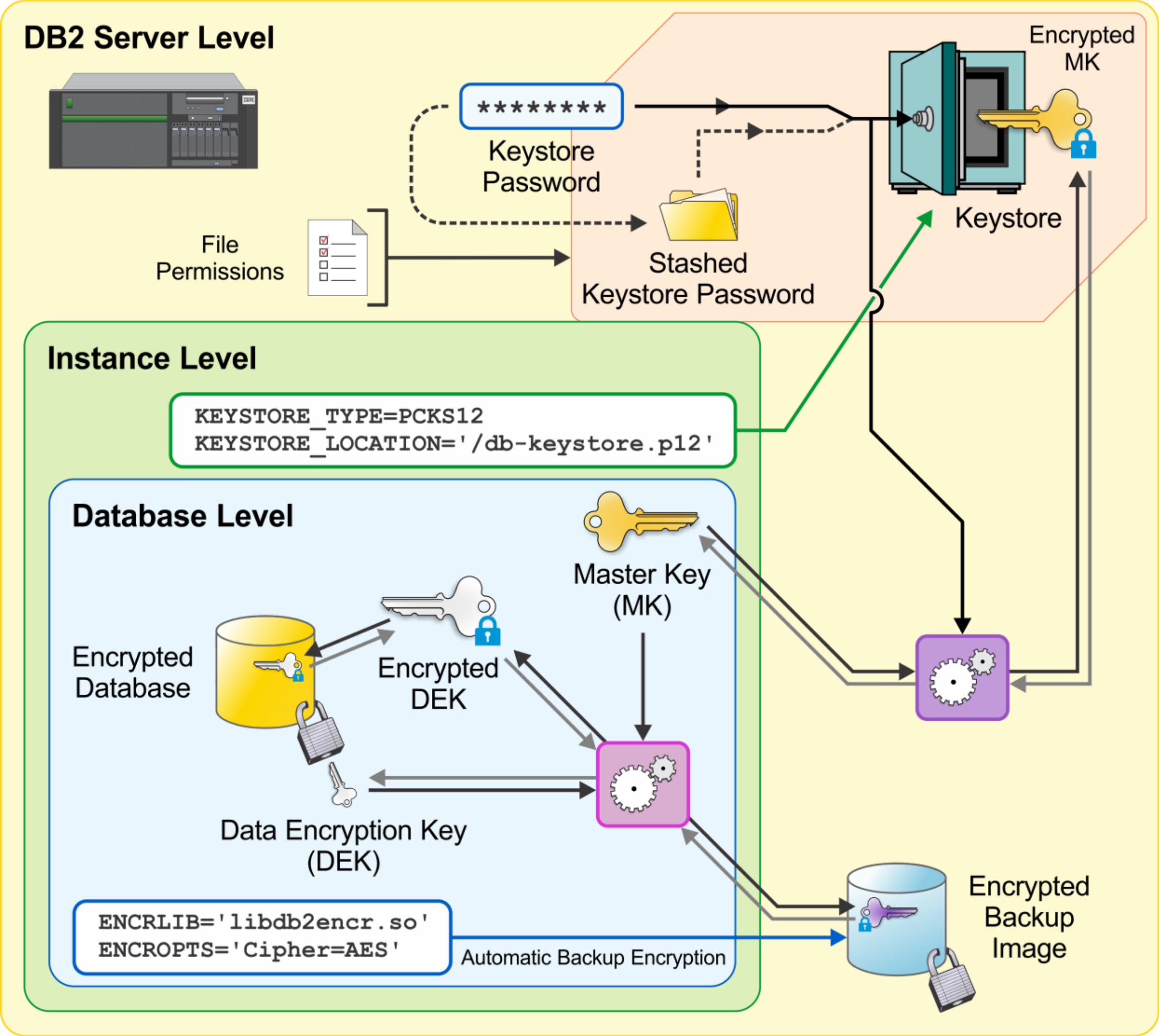
Simple Key Mgt : a local flat file used for a specific DB2 instance

DB2 v11.1

DB2 v11.1 FP*n*

Enterprise Key Mgt : a centralized key manager or HSM that can be used across many databases, file systems and other uses across an enterprise

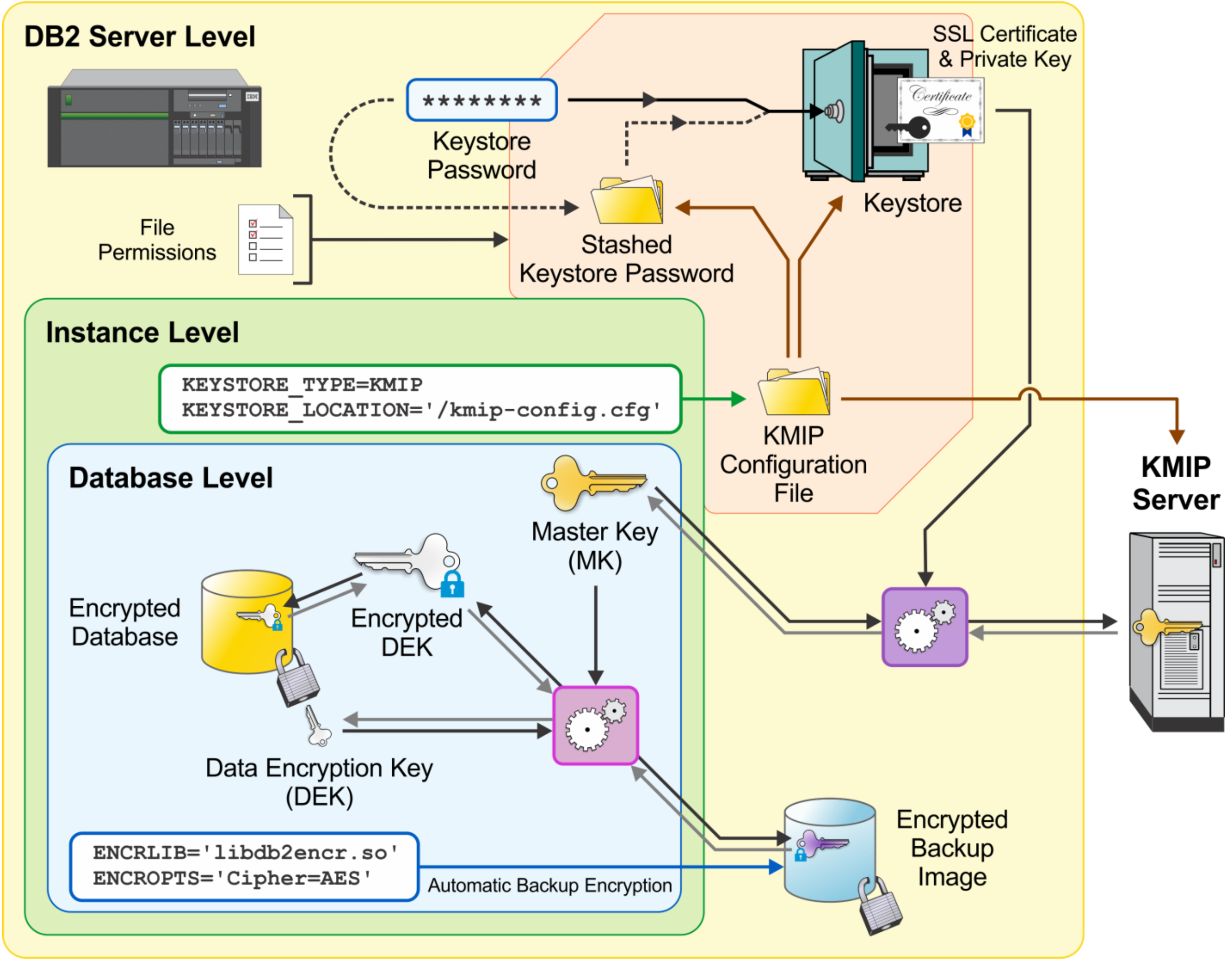
DB2 Native Encryption – Local Keystore



```
KEYSTORE_TYPE=PKCS12
KEYSTORE_LOCATION='/db-keystore.p12'
```

```
ENCRLIB='libdb2encr.so'
ENCROPTS='Cipher=AES'
```

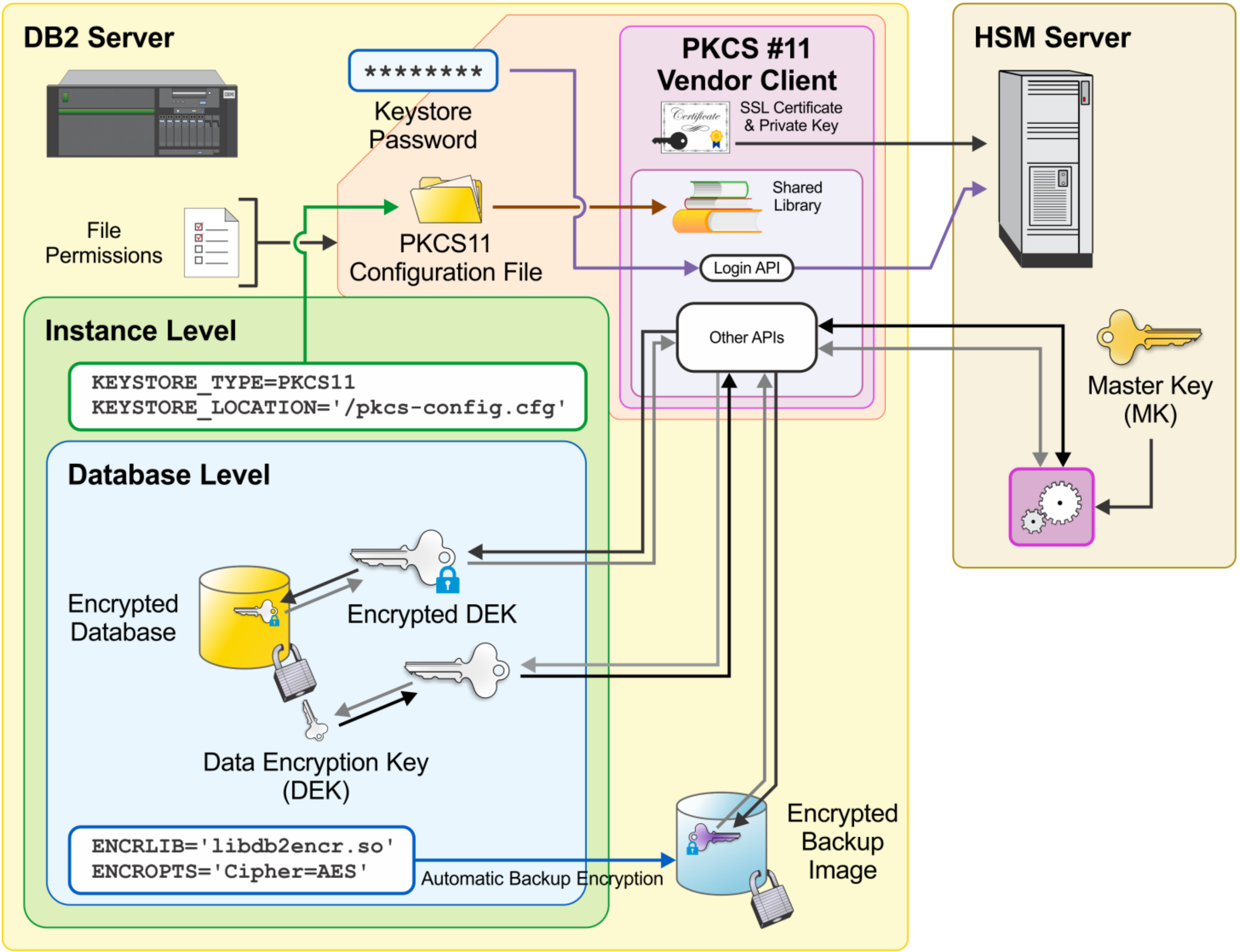
DB2 Native Encryption – Centralized Key Manager



```
KEYSTORE_TYPE=KMIP
KEYSTORE_LOCATION='/kmip-config.cfg'
```

```
ENCRLIB='libdb2encr.so'
ENCROPTS='Cipher=AES'
```

DB2 Native Encryption – Hardware Security Module



```
KEYSTORE_TYPE=PKCS11
KEYSTORE_LOCATION='/pkcs-config.cfg'
```

```
ENCRLIB='libdb2encr.so'
ENCROPTS='Cipher=AES'
```

What Gets Encrypted?

- All table spaces (system defined and user defined)
- Transaction logs (including logs in archives)
- LOAD COPY data
- LOAD staging files
- Dump .bin files
- All backup images
- Encryption keys stored in memory
- Keystore passwords when they are transparently communicated from one member or partition to another during a restart

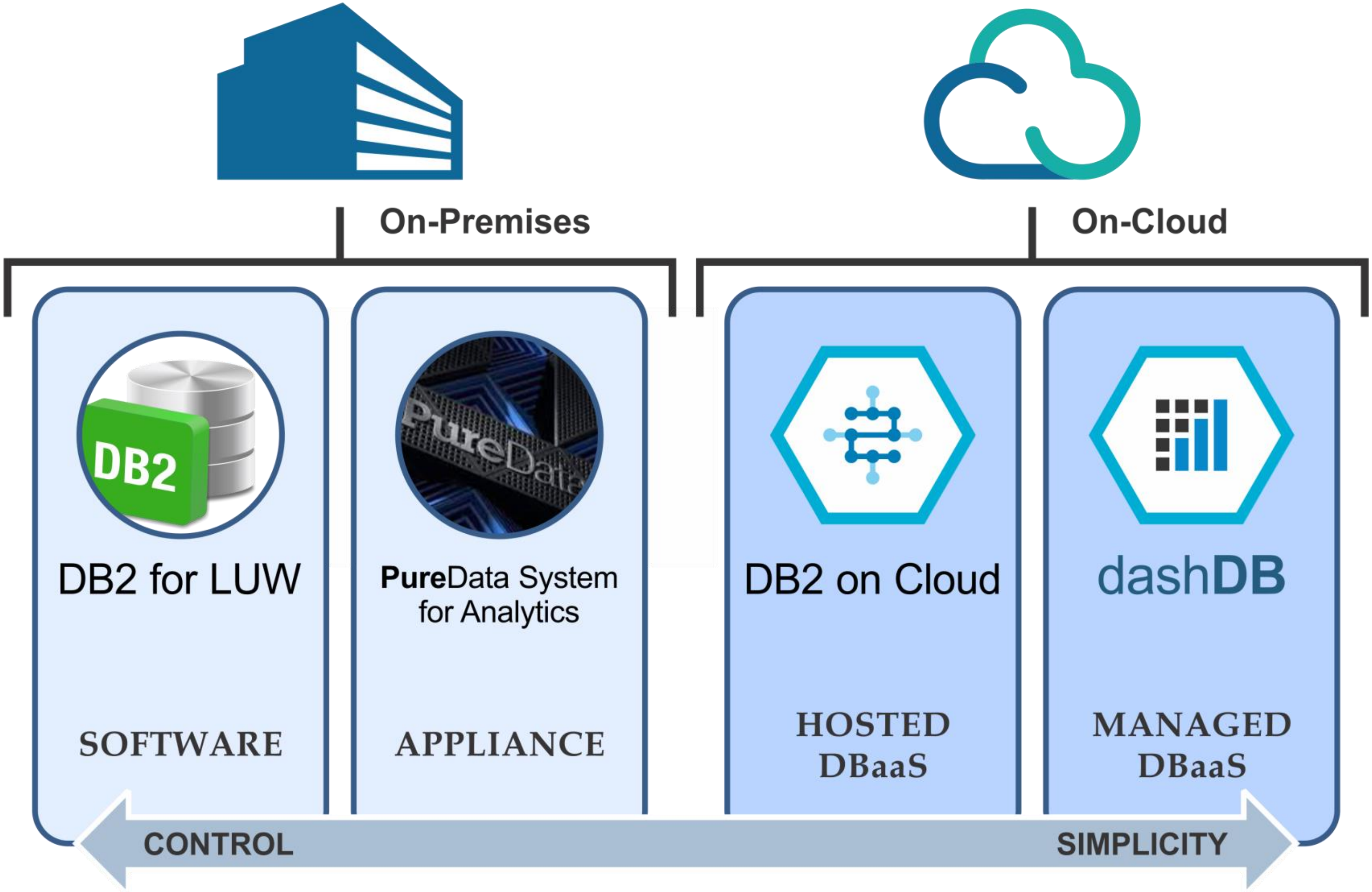
A Word About The DB2 Audit Facility

The **DB2 audit facility** is used to generate an audit trail for a series of predefined database events – records generated by this facility are kept in audit log files and close analysis of these files can provide insight into *who* did *what*, *when*, *where*, and *how*.

The db2audit command is used to configure the audit facility at the *instance level* as well as control when audit information is collected.

Audit policies, together with the AUDIT statement are used to configure and control auditing at the *database level*.

DB2 Deployment Options



DB2 on Cloud – Default Users

DB2 on Cloud

DB2 on Cloud has the same security features as on-premises DB2. However, because this offering is a hosted service, some security-related decisions have already been made by IBM. For example, the following users are preconfigured:

adminuser

- Home directory is /home/adminuser
- Password provided in introductory email sent to customer
- Allowed to perform root activities via the sudo command

db2inst1

- Home directory is /home/db2inst1
- Password is set by adminuser (**sudo passwd db2inst1**)

DB2 on Cloud – Environment Access

DB2 on Cloud

- A DB2 on Cloud server can be accessed via a **Secure Shell (SSH) client** (like PuTTY)
  - Server URL provided in introductory email sent to customer; port number is 22
  - Can log in as **adminuser**, **db2inst1**, or any user created by **adminuser**
- There is no console/GUI
  - However, tools like Data Studio and Data Server Manager (DSM) can be used to manage the environment
- Database clients can connect in the same way they can connect to an on-premises DB2 server
  - Trusted contexts should be utilized whenever possible

DB2 on Cloud – Remote Connectivity

DB2 on Cloud

Remote client connectivity is preconfigured by IBM as follows:

- TCP/IP communication and SSL is enabled via the DB2COMM registry variable:

```
DB2COMM=TCPIP,SSL
```

- A non-SSL listening port is defined in the /etc/services file:

```
db2c_db2inst1  50000/tcp # DB2 connection service port
```

- The iptables file (which is used by the Linux kernel firewall) is configured to accept incoming requests on SSL and non-SSL ports:

```
ACCEPT     tcp  --  anywhere      anywhere      tcp dpt:50001
ACCEPT     tcp  --  anywhere      anywhere      tcp dpt:db2c_db2inst1
```

DB2 on Cloud – Remote Connectivity (Continued)

DB2 on Cloud

- The DB2 database manager configuration is updated with port and SSL information:

```
TCP/IP Service name                          (SVCENAME) = db2c_db2inst1
SSL server keydb file                   (SSL_SVR_KEYDB) = /home/db2inst1/ssl_keystore/sqldb_ssl.kdb
 SSL server stash file                  (SSL_SVR_STASH) = /home/db2inst1/ssl_keystore/sqldb_ssl.sth
 SSL server certificate label           (SSL_SVR_LABEL) = sqldb_ssl
 SSL service name                        (SSL_SVCENAME) = 50001
```

dashDB – Default Users

Again, most of the security mechanisms available for DB2 on-premises are available with **dashDB**. However, because this offering is a managed service, many security-related decisions have already been made. For example, there are only two distinct types of users:

Administrative users

- Similar to on-premises database administrators
- Have the ability to create other users

Regular users

- Similar to on-premises users with DATAACCESS authority
- Can be assigned a specific IP address (which, together with the User ID form a trusted context)

dashDB – Environment Access

- A dashDB database is accessed through the **dashDB Console** (via IBM Bluemix)
  - User ID management is also done through the dashDB Console
- Applications can connect to a dashDB database, with or without using SSL
  - By default, dashDB listens for non-SSL connections on port **50000** and listens for SSL connections on port **50001**
  - *It is possible to configure dashDB so that it only accepts SSL connections*
  - Information needed to connect can be obtained from the **Connect** menu in the dashDB Console
  - You may need to download and install a driver package; links are provided via the dashDB Console

dashDB – Other Differences

dashDB

- Local, transparent LDAP authentication is used to authenticate users
- Only one database is available
  - That database has DB2 native encryption enabled
  - The AES algorithm is used, along with 256-bit keys – the longest allowed; (*AES is secure enough to be used to protect US Government Top Secret information*)
  - Master keys are stored locally, in a Public Key Cryptography Standard (PKCS) #12 keystore and are rotated/changed automatically every 90 days
  - Encryption ripples through to backup images, using the same master key used to encrypt the database

dashDB – Other Differences (Continued)

- If LBAC protection is desired, row-organized tables must be explicitly created first
  - Column-organized tables are created by default; LBAC does not work with column-organized tables
- QRadar and Guardium are used to provide audit coverage for both the operating system and the database
  - These are internal to the DevOps team; reports and configuration are not exposed to clients
- Physical security requirements for many compliance specifications (HIPPA, AICPA, etc.) are provided by SoftLayer

SoftLayer: Physical Security Compliance

Questions?

Thanks for attending!