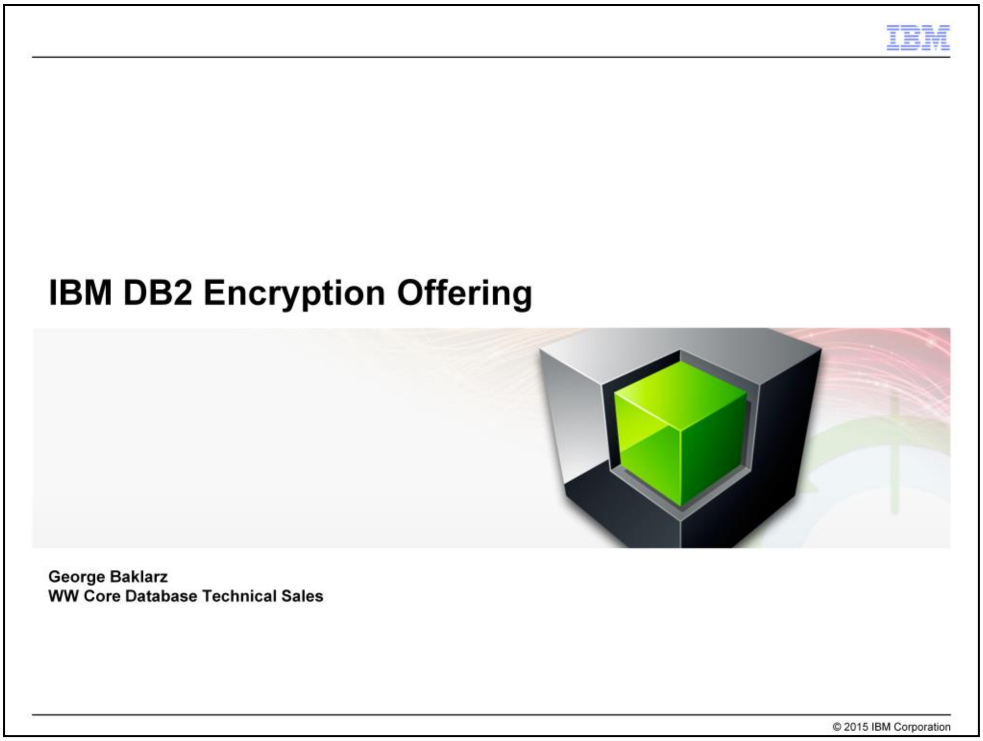


IBM DB2 Encryption Offering

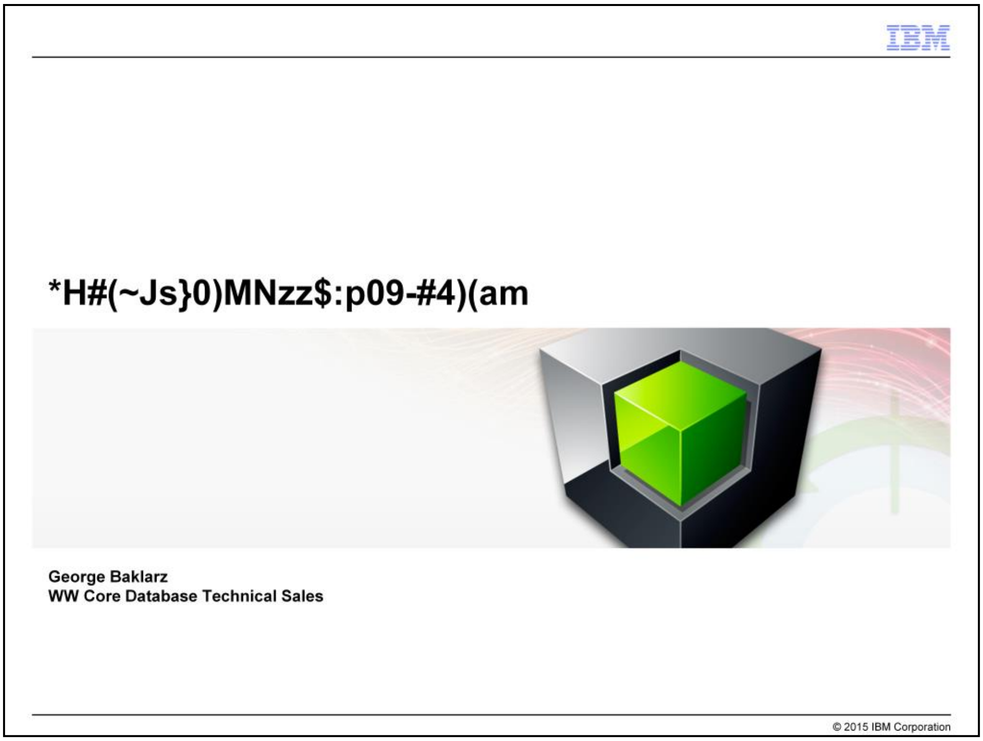
George Baklarz
WW Core Database Technical Sales

This presentation contains information on the new Encryption Technology (DB2 Native Encryption) that was delivered in DB2 10.5 FP5. The material within this presentation was derived from the DB2 10.5 FP5 Skills Transfer session that was developed by:

- Mihai Iacob
- Geoffrey Ng
- Hamdi Roumani
- Greg Stager

Some of the speaker notes were derived from DB2 documentation and blog contents from Walid Rjaibi (CTO, Guardium Data Security).

For questions or feedback regarding this presentation, please contact George Baklarz (baklarz@ca.ibm.com) so that any corrections or changes can be made to the original charts.



*H#(~Js}0)MNzz$:p09-#4)(am

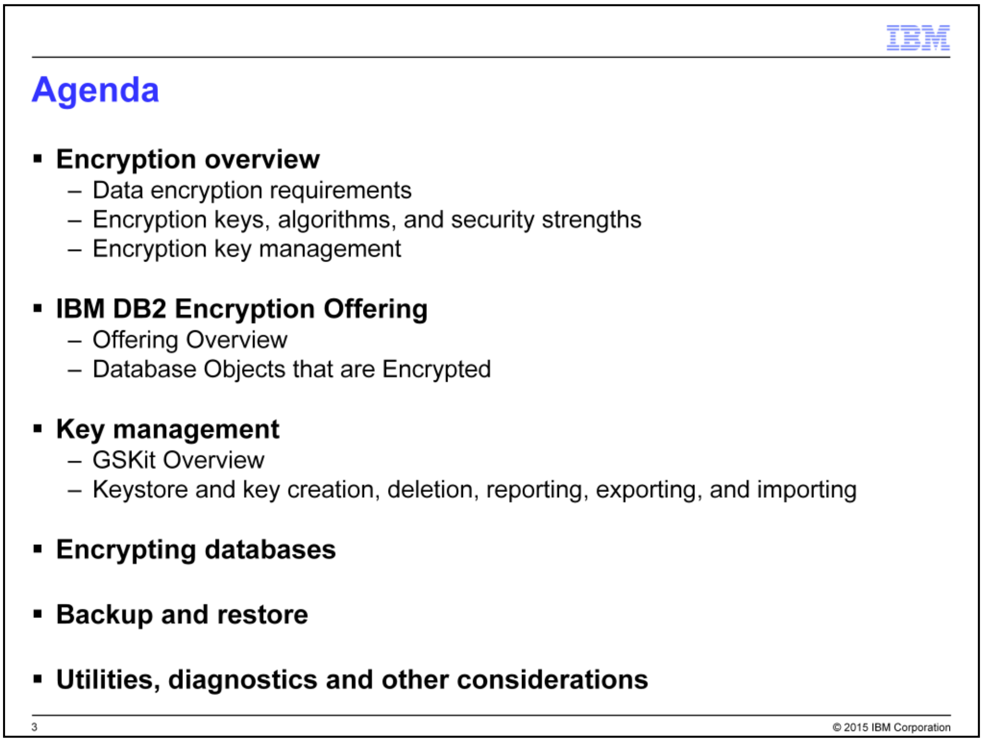
George Baklarz
WW Core Database Technical Sales

This presentation contains information on the new Encryption Technology (DB2 Native Encryption) that was delivered in DB2 10.5 FP5. The material within this presentation was derived from the DB2 10.5 FP5 Skills Transfer session that was developed by:

- Mihai Iacob
- Geoffrey Ng
- Hamdi Roumani
- Greg Stager

Some of the speaker notes were derived from DB2 documentation and blog contents from Walid Rjaibi (CTO, Guardium Data Security).

For questions or feedback regarding this presentation, please contact George Baklarz (baklarz@ca.ibm.com) so that any corrections or changes can be made to the original charts.



Agenda

- **Encryption overview**
  - Data encryption requirements
  - Encryption keys, algorithms, and security strengths
  - Encryption key management
- **IBM DB2 Encryption Offering**
  - Offering Overview
  - Database Objects that are Encrypted
- **Key management**
  - GSKit Overview
  - Keystore and key creation, deletion, reporting, exporting, and importing
- **Encrypting databases**
- **Backup and restore**
- **Utilities, diagnostics and other considerations**

3

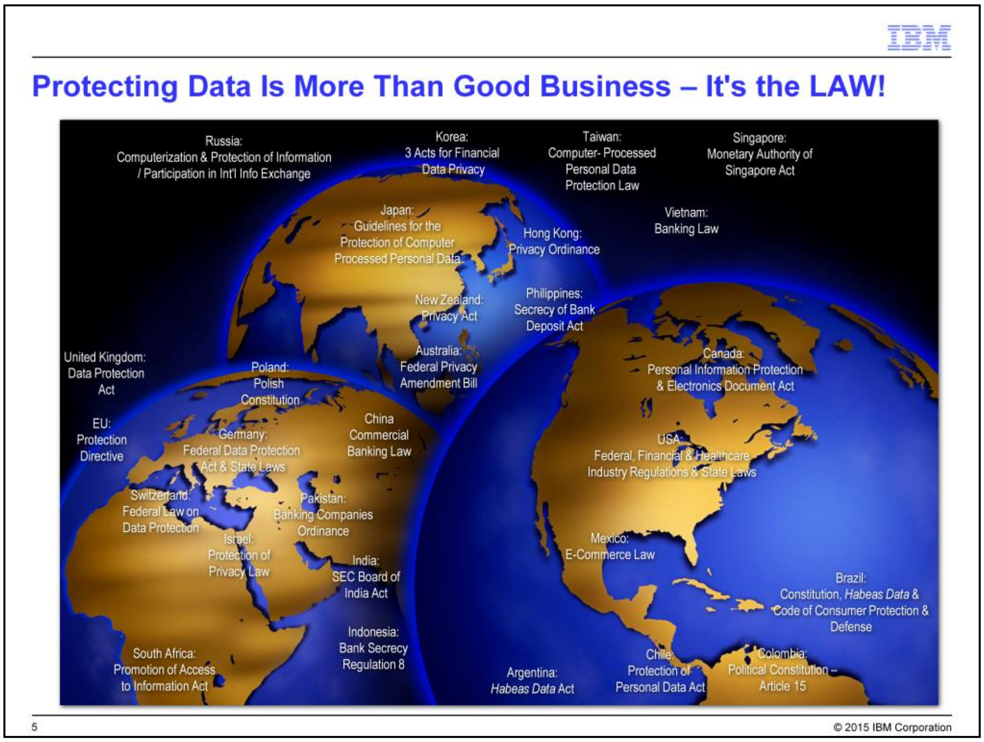
This presentation will cover four sections:

1)An overview of Encryption - Why the need for encryption (or why you really need encryption!)
2)The IBM DB2 Encryption Offering details
3)Key Management – How keystores are created and managed
4)Encrypting databases in DB2
5)Backup and Restore process with encrypted databases
6)Utilities, Diagnostics, and other considerations when using encryption

Encryption Overview

© 2015 IBM Corporation

This section will explore the reasons why encryption are important and what types of encryption technology there are in the marketplace. Some terminology will also be introduced in this section so that the DB2 commands in the next section will be easier to understand.



Protecting Data Is More Than Good Business – It's the LAW!

Russia:
Computerization & Protection of Information / Participation in Int'l Info Exchange
Korea:
3 Acts for Financial Data Privacy
Taiwan:
Computer- Processed Personal Data Protection Law
Singapore:
Monetary Authority of Singapore Act
Japan:
Guidelines for the Protection of Computer Processed Personal Data
Hong Kong:
Privacy Ordinance
Vietnam:
Banking Law
New Zealand:
Privacy Act
Philippines:
Secrecy of Bank Deposit Act
United Kingdom:
Data Protection Act
Poland:
Polish Constitution
Australia:
Federal Privacy Amendment Bill
Canada:
Personal Information Protection & Electronics Document Act
EU:
Protection Directive
Germany:
Federal Data Protection Act & State Laws
China
Commercial Banking Law
USA:
Federal, Financial & Healthcare Industry Regulations & State Laws
Switzerland:
Federal Law on Data Protection
Pakistan:
Banking Companies Ordinance
Israel:
Protection of Privacy Law
India:
SEC Board of India Act
Mexico:
E-Commerce Law
Brazil:
Constitution, Habeas Data & Code of Consumer Protection & Defense
Indonesia:
Bank Secrecy Regulation 8
South Africa:
Promotion of Access to Information Act
Argentina:
Habeas Data Act
Chile:
Protection of Personal Data Act
Colombia:
Political Constitution – Article 15

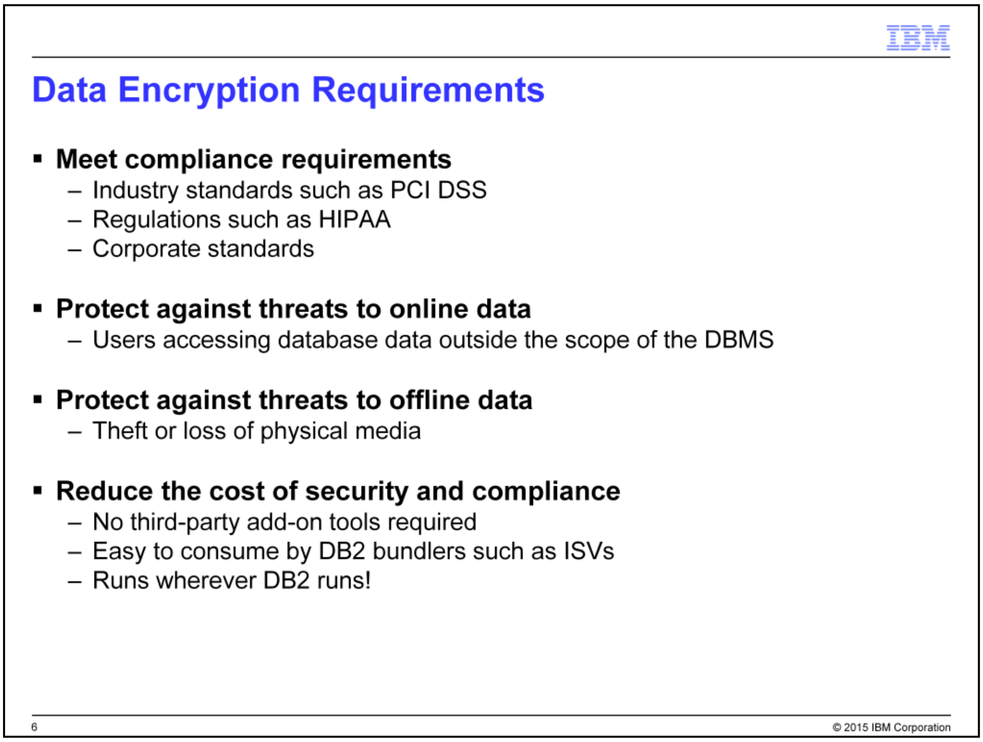
5

© 2015 IBM Corporation

Most people will agree that securing your data from theft and espionage are very important. However, there are worldwide regulations in place that concern data security and it's the law in some countries that you must adequately protect your data.

Companies span the globe and work with partners and colleagues around the world. Companies need to be aware of global regulations. We can expect an increase in regulations going forward.

There are also voluntary compliance requirements such as PCI which are growing in importance.



Data Encryption Requirements

- **Meet compliance requirements**
  - Industry standards such as PCI DSS
  - Regulations such as HIPAA
  - Corporate standards
- **Protect against threats to online data**
  - Users accessing database data outside the scope of the DBMS
- **Protect against threats to offline data**
  - Theft or loss of physical media
- **Reduce the cost of security and compliance**
  - No third-party add-on tools required
  - Easy to consume by DB2 bundlers such as ISVs
  - Runs wherever DB2 runs!

6 © 2015 IBM Corporation

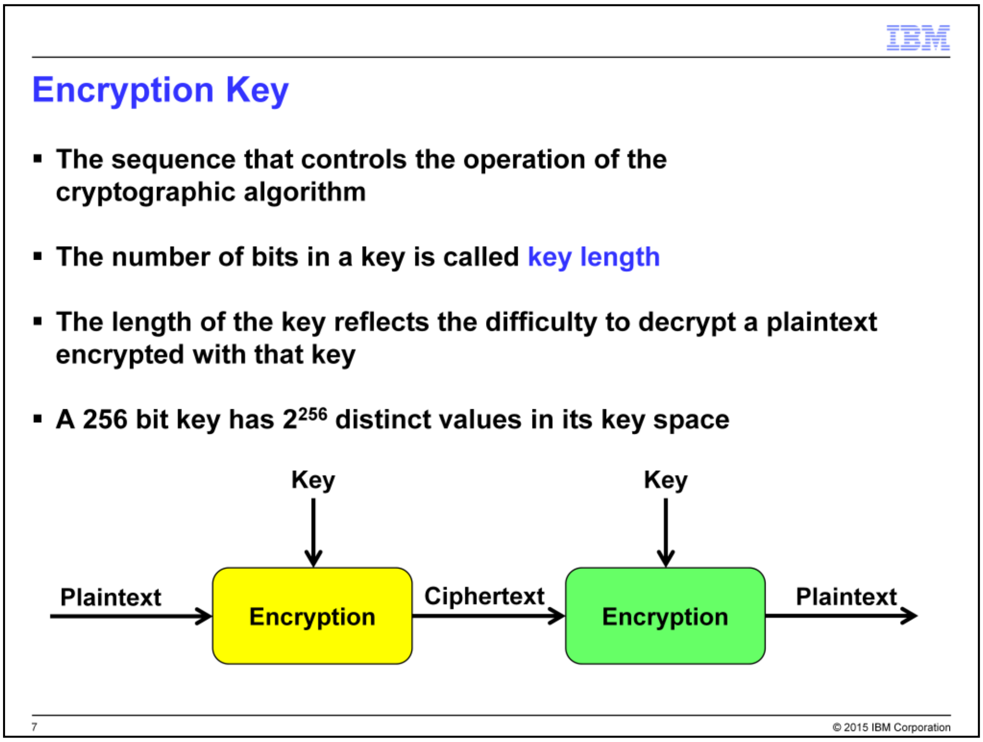
Increasingly, businesses desire or are mandated to encrypt sensitive data to meet organizational or regulatory requirements.

Some of the major governance regulations are

- PCI, Sarbanes-Oxley (SOX), HIPAA
- Data Breach Disclosure Laws, Gramm-Leach-Bliley, Basel II

The DB2 Encryption Offering assists organizations to meet those requirements by providing, natively within the IBM DB2 database engine itself, encryption capabilities that encrypt data at rest for the entire database, including backup images. DB2's native encryption ensures that sensitive data is encrypted and secured at all times. Deployment of DB2's native encryption is straightforward to enable, transparent to the applications accessing the data, and is applied to backups as well. DB2's native encryption meets the requirements of NIST SP 800-131 compliant cryptographic algorithms and utilizes FIPS 140-2 certified cryptographic libraries.

This offering is available on IBM DB2 Enterprise Server Edition, IBM DB2 Workgroup Server Edition, and IBM DB2 Express® Server Edition for an additional charge. For those customer that already have DB2 Advanced Enterprise Server Edition, and DB2 Advanced Workgroup Server Edition, this feature is included at no additional charge. This feature is also included in the free DB2 Express Community Edition (Express-C) so that developers can prototype the technology for future production use.



Encryption Key

- The sequence that controls the operation of the cryptographic algorithm
- The number of bits in a key is called **key length**
- The length of the key reflects the difficulty to decrypt a plaintext encrypted with that key
- A 256 bit key has 2^{256} distinct values in its key space

Key

Key

Plaintext

Encryption

Ciphertext

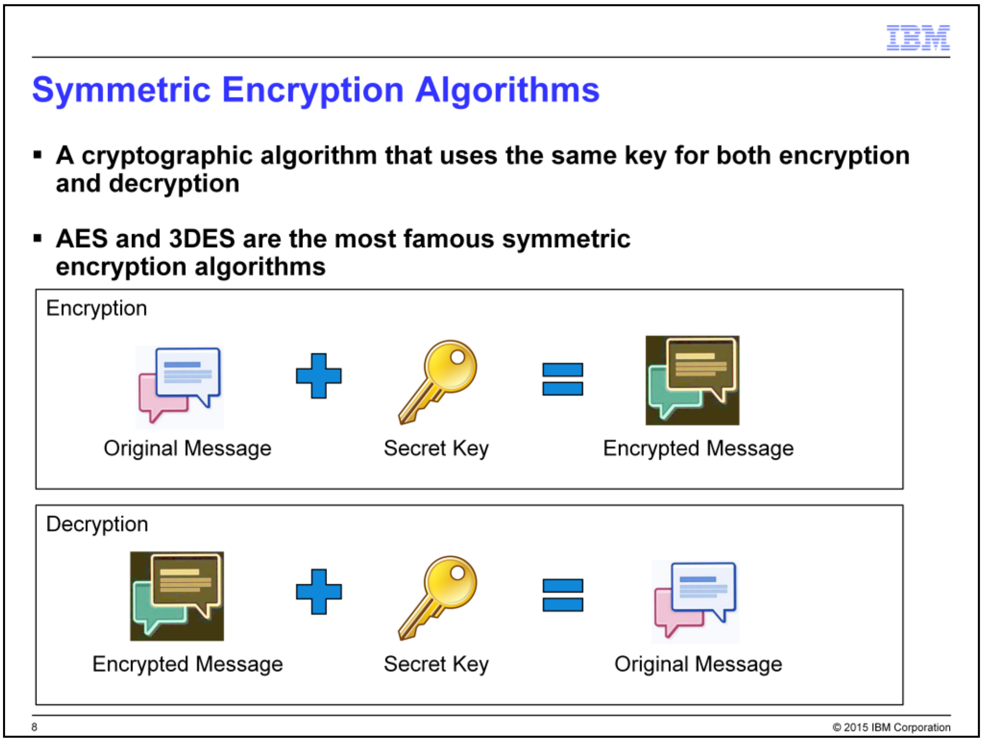
Encryption

Plaintext

The Encryption key is used to control the cryptographic algorithm that is used to "encrypt" and obfuscate the contents of the data. Encryption keys have various lengths which go from 128 bits up to 256 bits. The number of bits that are used are called the key length, and generally speaking, the larger the number of bits, the stronger the encryption.

Encryption strength refers to the amount of effort or complexity required in order to "crack" the code. A 256 bit key would require a huge number of computations in order to "brute force" find the key. It is estimated that even a 128-bit key would take at least 30 years of compute power to guess.

The process of encrypting data is relatively simple – the data is encrypted with the key and then decrypted using the same key. Unless the key is known, there is no way to decrypt the data. In fact, losing the key would result in the data being lost because there is no method that can be used to recover the key.



Symmetric Encryption Algorithms

- A cryptographic algorithm that uses the same key for both encryption and decryption
- AES and 3DES are the most famous symmetric encryption algorithms

Encryption

Original Message + Secret Key = Encrypted Message

Decryption

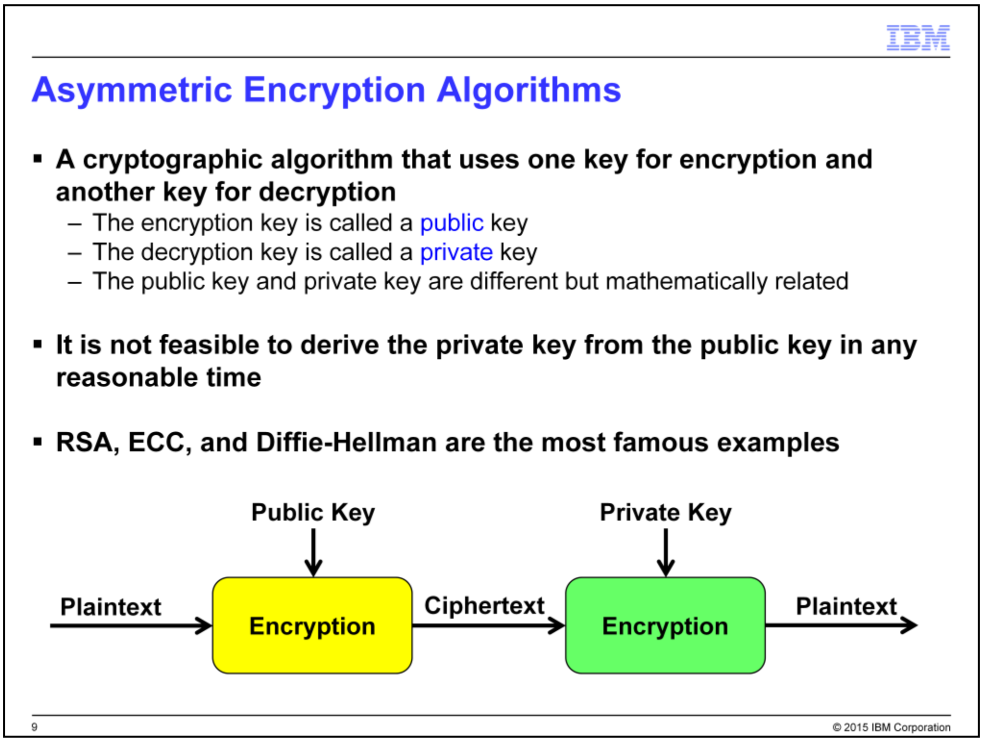
Encrypted Message + Secret Key = Original Message

8

Symmetric key algorithms use the same key for encrypting and decrypting the data. The most common versions of symmetric encryption algorithms includes AES (Advanced Encryption Standard) and 3DES (Triple DES, or Data Encryption Standard).

In cryptography, Triple DES (3DES) is the common name for the Triple Data Encryption Algorithm (TDEA or Triple DEA) symmetric-key block cipher, which applies the Data Encryption Standard (DES) cipher algorithm three times to each data block.

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is the successor of DES as standard symmetric encryption algorithm for US federal organizations. AES uses keys of 128, 192 or 256 bits, although, 128 bit keys provide sufficient strength today.



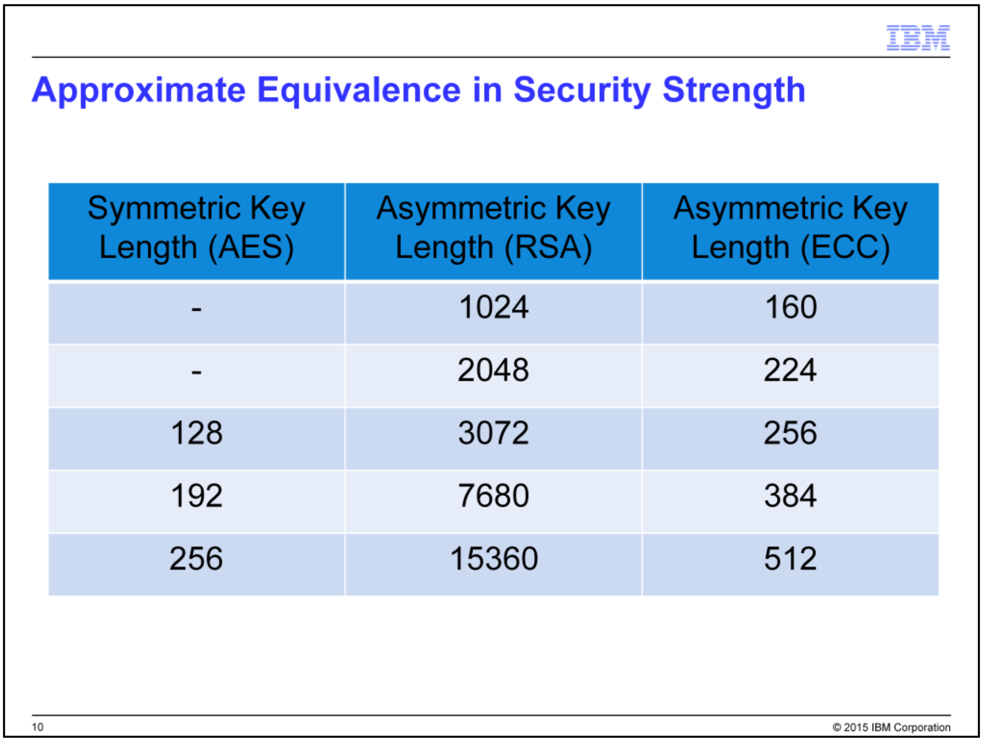
Asymmetric Encryption Algorithms

- **A cryptographic algorithm that uses one key for encryption and another key for decryption**
  - The encryption key is called a public key
  - The decryption key is called a private key
  - The public key and private key are different but mathematically related
- **It is not feasible to derive the private key from the public key in any reasonable time**
- **RSA, ECC, and Diffie-Hellman are the most famous examples**

9

Asymmetric Encryption Algorithms require a public/private key combination. The text is encrypted using a public key, but the data can only be decrypted using a private key. The keys are mathematically related, but it is virtually impossible to derive the private key because of the complexity involved.

The common forms of Asymmetric algorithms includes RSA, ECC, and Diffie-Hellman. Note that Asymmetric Encryption is typically used for communication of emails, transactions, etc… on the internet. It tends to require more processing power than Symmetric algorithms, so data at rest (i.e. databases) applications use Symmetric algorithms.

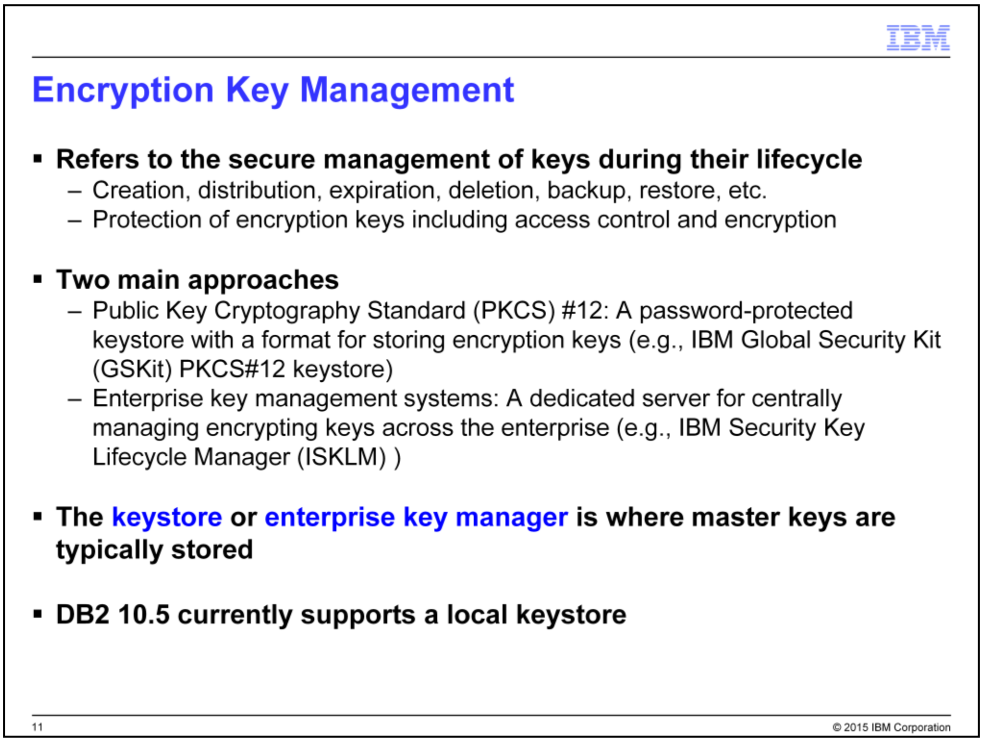


Approximate Equivalence in Security Strength

Symmetric Key Length (AES)	Asymmetric Key Length (RSA)	Asymmetric Key Length (ECC)
-	1024	160
-	2048	224
128	3072	256
192	7680	384
256	15360	512

10

Customer may ask about the strength of the encryption and the length of an encryption key. It is important to know that the number of bits used is different for Symmetric versus Asymmetric algorithms. If someone states that they require at least 3000 bits of key length then they are referring to an Asymmetric algorithm (RSA) not AES. This charts compares the various encryption formats and their bit lengths. A 128-bit AES key is extremely secure and may be sufficient for most customers. In any case, other key bit lengths are available, up to 256 bits long. Note that 3DES is not shown on this slide. 3DES uses 168-bit keys for encryption.



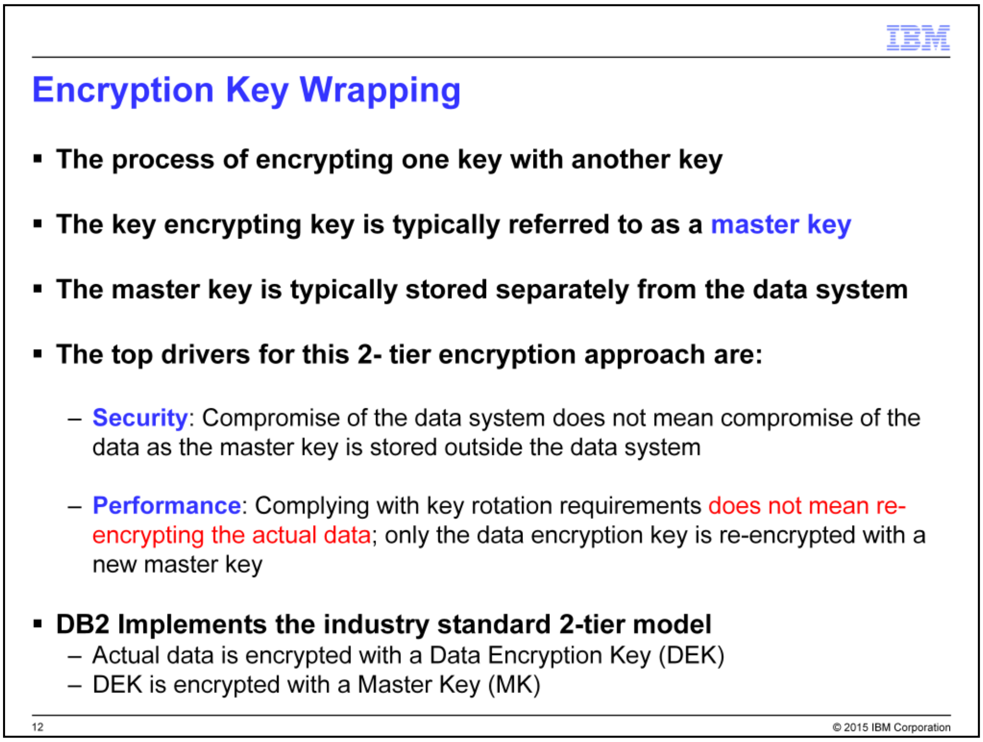
Encryption Key Management

- **Refers to the secure management of keys during their lifecycle**
  - Creation, distribution, expiration, deletion, backup, restore, etc.
  - Protection of encryption keys including access control and encryption
- **Two main approaches**
  - Public Key Cryptography Standard (PKCS) #12: A password-protected keystore with a format for storing encryption keys (e.g., IBM Global Security Kit (GSKit) PKCS#12 keystore)
  - Enterprise key management systems: A dedicated server for centrally managing encrypting keys across the enterprise (e.g., IBM Security Key Lifecycle Manager (ISKLM))
- **The keystore or enterprise key manager is where master keys are typically stored**
- **DB2 10.5 currently supports a local keystore**

11

Encryption key management refers to the secure management of keys during their lifecycle. The key is a critical part of any encryption environment and needs to be protected like any other object within the operating environment.

Keys can be kept either locally (in a keystore), or in an enterprise key manager. For this release of DB2, the keys are stored in a local keystore and the use of an enterprise key manager is not supported. The requirement to place keys in an enterprise key manager is high on the priority list. However, this is not a guarantee that this feature will be released at any time in the future.



Encryption Key Wrapping

- **The process of encrypting one key with another key**
- **The key encrypting key is typically referred to as a master key**
- **The master key is typically stored separately from the data system**
- **The top drivers for this 2- tier encryption approach are:**
  - **Security**: Compromise of the data system does not mean compromise of the data as the master key is stored outside the data system
  - **Performance**: Complying with key rotation requirements does not mean re-encrypting the actual data; only the data encryption key is re-encrypted with a new master key
- **DB2 Implements the industry standard 2-tier model**
  - Actual data is encrypted with a Data Encryption Key (DEK)
  - DEK is encrypted with a Master Key (MK)

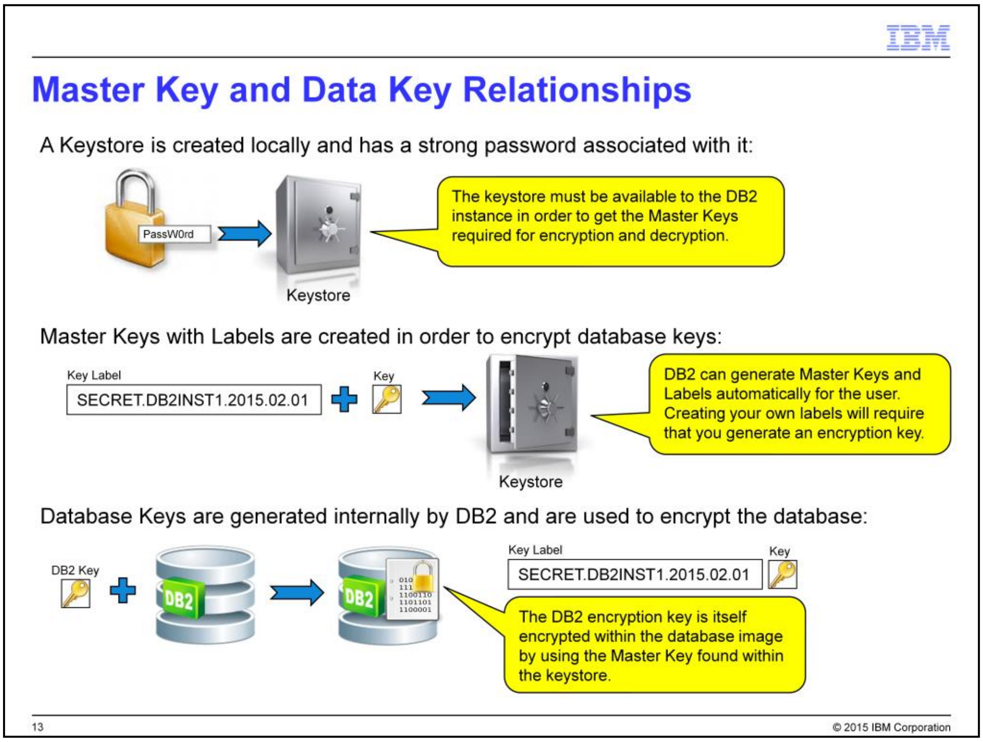
Encryption key wrapping refers to the encrypting of one key with another key. The keystore contains a "master" key to the database. This master key is used to encrypt the key that is found within the database (database encryption key). The database encryption key is what is used to encrypt the data within the database. The reason for having this two-tiered approach is to improve the security of the system and improve the performance of key rotation (which is described in a later slide).

DB2 makes use of the IBM Global Security Kit (GSKit) libraries to create, store, and manage the keys required for encrypting databases. Key data (i.e. keys, certificates, and related information) are stored in a keystore, a file stored in the operational environment. The initial release of DB2 Encryption implements the keystore locally. The keys are managed used an industry standard 2-tier model:

•Actual data is encrypted with a Data Encryption Key (DEK)

•DEK is encrypted with a Master Key (MK)

•DEK is managed within the database while the MK is managed externally



Master Key and Data Key Relationships

A Keystore is created locally and has a strong password associated with it:

The keystore must be available to the DB2 instance in order to get the Master Keys required for encryption and decryption.

Master Keys with Labels are created in order to encrypt database keys:

DB2 can generate Master Keys and Labels automatically for the user. Creating your own labels will require that you generate an encryption key.

Database Keys are generated internally by DB2 and are used to encrypt the database:

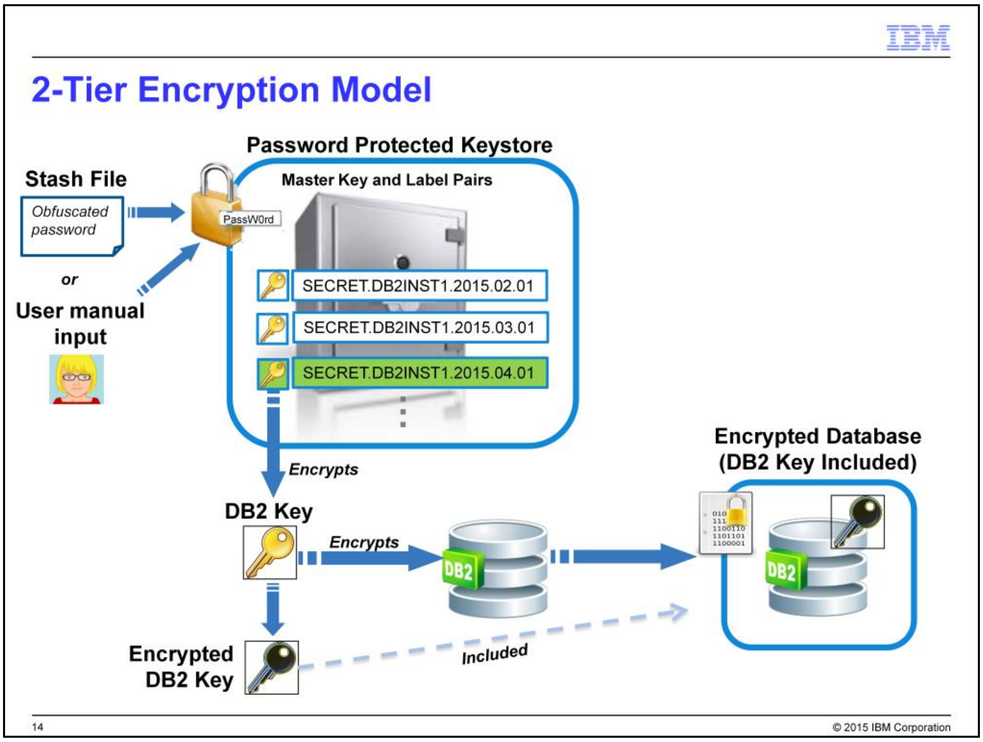
The DB2 encryption key is itself encrypted within the database image by using the Master Key found within the keystore.

13

A keystore must be created locally (or accessible) so that the DB2 instance is able to access it. This keystore will contain Master Keys. These master keys contain labels so that they can be easily referenced during create database commands and other maintenance procedures.

You can create a master key manually, or have DB2 create them for you automatically. These Master Keys are used to encrypt the Database key within DB2, not the actual data within the database.

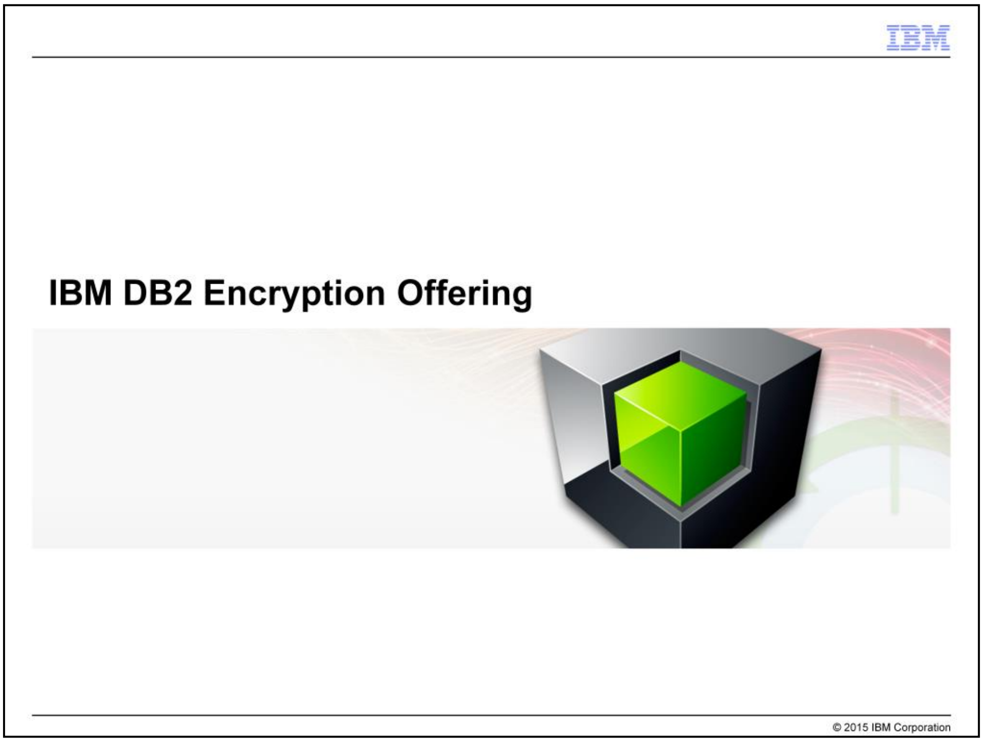
When an encrypted database is created, DB2 will generate a random key internally. This key will be used to encrypt the contents of the database. This key that was generated by DB2 is then itself encrypted with the Master Key and stored within the database image itself. This means that if the database is removed from the system (i.e. the keystore is not available), the database cannot be decrypted because it needs the master key to decrypt the key that was used to encrypt the database. Someone would need access to the keystore and the database in order to be able to decrypt it.



2-Tier Encryption Model

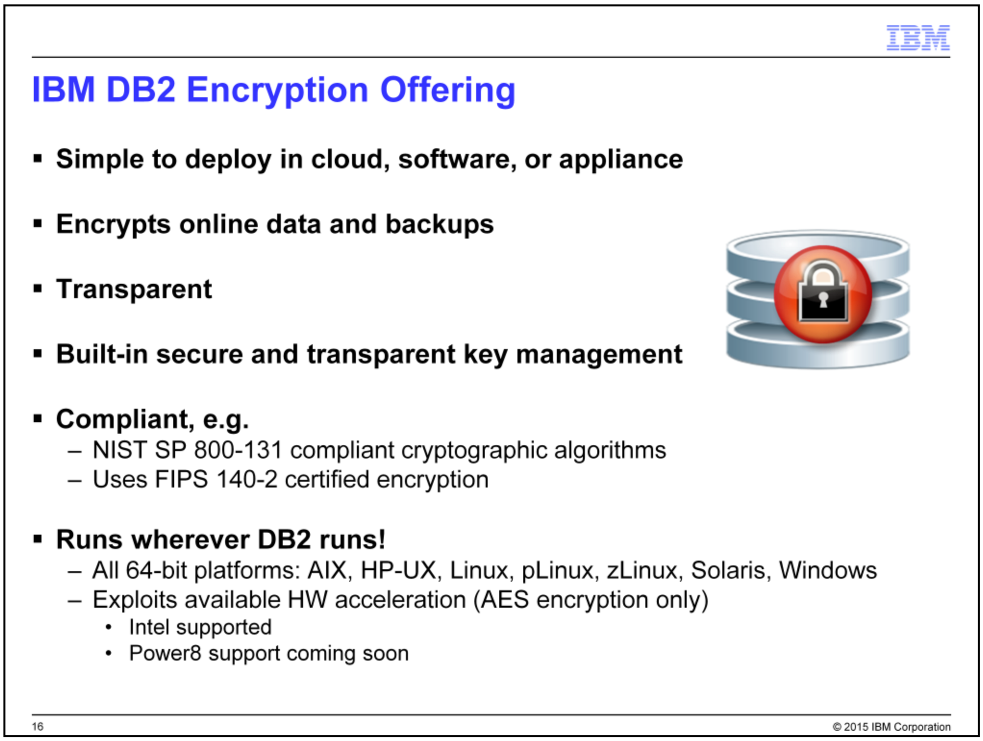
IBM
Password Protected Keystore
Stash File
Obfuscated password
or
User manual input
PassW0rd
Master Key and Label Pairs
SECRET.DB2INST1.2015.02.01
SECRET.DB2INST1.2015.03.01
SECRET.DB2INST1.2015.04.01
Encrypts
DB2 Key
Encrypts
DB2
Encrypted Database (DB2 Key Included)
010
111
1100110
1101101
1100001
DB2
Included
Encrypted DB2 Key
14
© 2015 IBM Corporation

This slide (created by Jennifer Chen) nicely summarizes the relationship between the keys used in DB2 Encryption.



IBM DB2 Encryption Offering

IBM DB2 Encryption Offering is the official name of the encryption product. This section will discuss the product and how a customer can implement encryption at a database level.



IBM

IBM DB2 Encryption Offering

- **Simple to deploy in cloud, software, or appliance**
- **Encrypts online data and backups**
- **Transparent**
- **Built-in secure and transparent key management**
- **Compliant, e.g.**
  - NIST SP 800-131 compliant cryptographic algorithms
  - Uses FIPS 140-2 certified encryption
- **Runs wherever DB2 runs!**
  - All 64-bit platforms: AIX, HP-UX, Linux, pLinux, zLinux, Solaris, Windows
  - Exploits available HW acceleration (AES encryption only)
    - Intel supported
    - Power8 support coming soon

16 © 2015 IBM Corporation

DB2 native encryption uses a symmetric encryption scheme for both the database and the backup images. Two symmetric encryption algorithms are supported: AES and 3DES. Encryption is supported for all DB2 configurations including DPF and pureScale. Similarly, encryption is supported for all types of tables including column organized tables (BLU Acceleration). The sections below cover the top 5 things you need to know about DB2 native encryption.

•DB2 native encryption is transparent to your applications and schemas

•Key Management is secure and transparent

•DB2 native encryption encrypts both your online data and your backup imagesDB2 native encryption employs certified and compliant cryptography, and exploits hardware acceleration for cryptographic operations.

•DB2 native encryption supports encrypting your existing DB2 databases

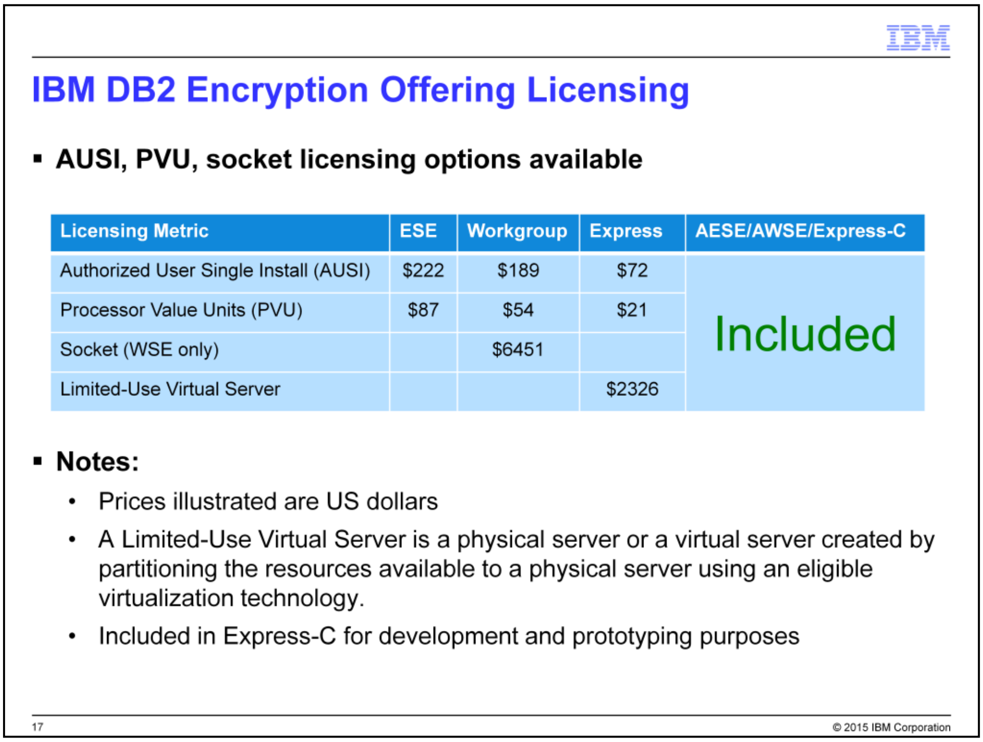
The Encryption feature is available on the following supported hardware platforms:

•AIX®, HP-UX, Linux 64 bit, Linux Power® System Big Endian

•Linux System z®, Inspur K-UX

•Solaris 64-bit SPARC, Solaris 64-bit Intel™ or AMD, Microsoft™ Windows 64 bit

The encryption feature is not available on any 32-bit platform.



IBM DB2 Encryption Offering Licensing

- **AUSI, PVU, socket licensing options available**

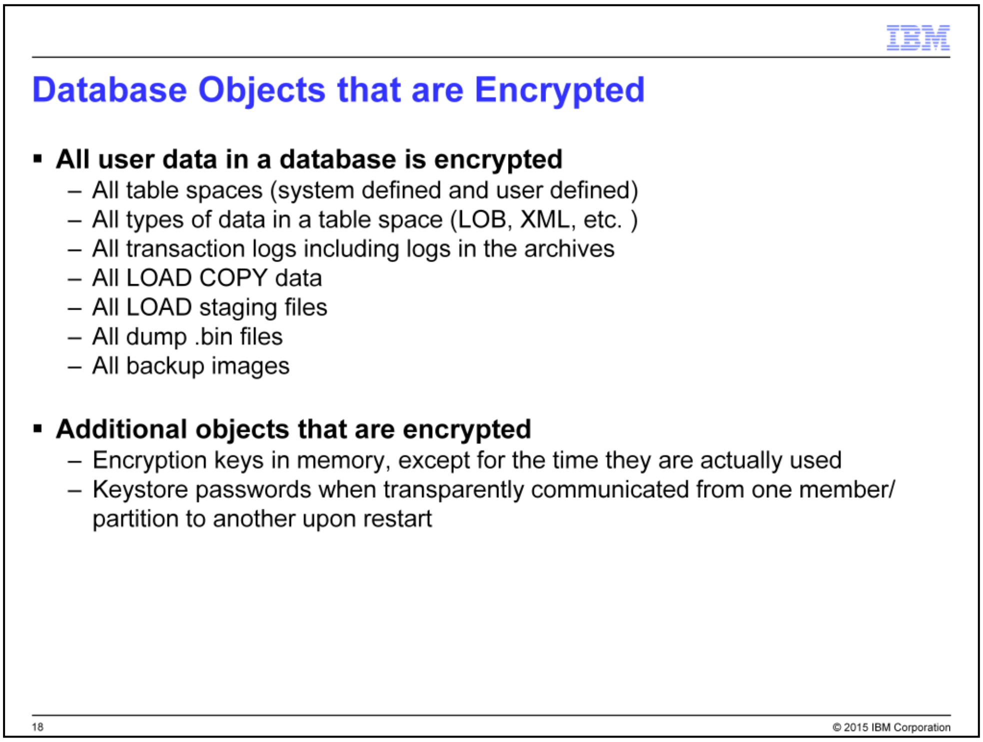
Licensing Metric	ESE	Workgroup	Express	AESE/AWSE/Express-C
Authorized User Single Install (AUSI)	$222	$189	$72	Included
Processor Value Units (PVU)	$87	$54	$21	
Socket (WSE only)		$6451		
Limited-Use Virtual Server			$2326	

- **Notes:**
  - Prices illustrated are US dollars
  - A Limited-Use Virtual Server is a physical server or a virtual server created by partitioning the resources available to a physical server using an eligible virtualization technology.
  - Included in Express-C for development and prototyping purposes

This offering is available on IBM DB2 Enterprise Server Edition, IBM DB2 Workgroup Server Edition, and IBM DB2 Express® Server Edition for an additional charge. For those customer that already have DB2 Advanced Enterprise Server Edition, and DB2 Advanced Workgroup Server Edition, this feature is included at no additional charge. This feature is also included in the free DB2 Express Community Edition (Express-C) so that developers can prototype the technology for future production use.

Note: Prices illustrated are US dollars only. Prices will vary by country and the feature may not be available in all countries.

A LU Virtual Server is a physical server or a virtual server created by partitioning the resources available to a physical server using an eligible virtualization technology.



Database Objects that are Encrypted

- **All user data in a database is encrypted**
  - All table spaces (system defined and user defined)
  - All types of data in a table space (LOB, XML, etc.)
  - All transaction logs including logs in the archives
  - All LOAD COPY data
  - All LOAD staging files
  - All dump .bin files
  - All backup images

- **Additional objects that are encrypted**
  - Encryption keys in memory, except for the time they are actually used
  - Keystore passwords when transparently communicated from one member/ partition to another upon restart

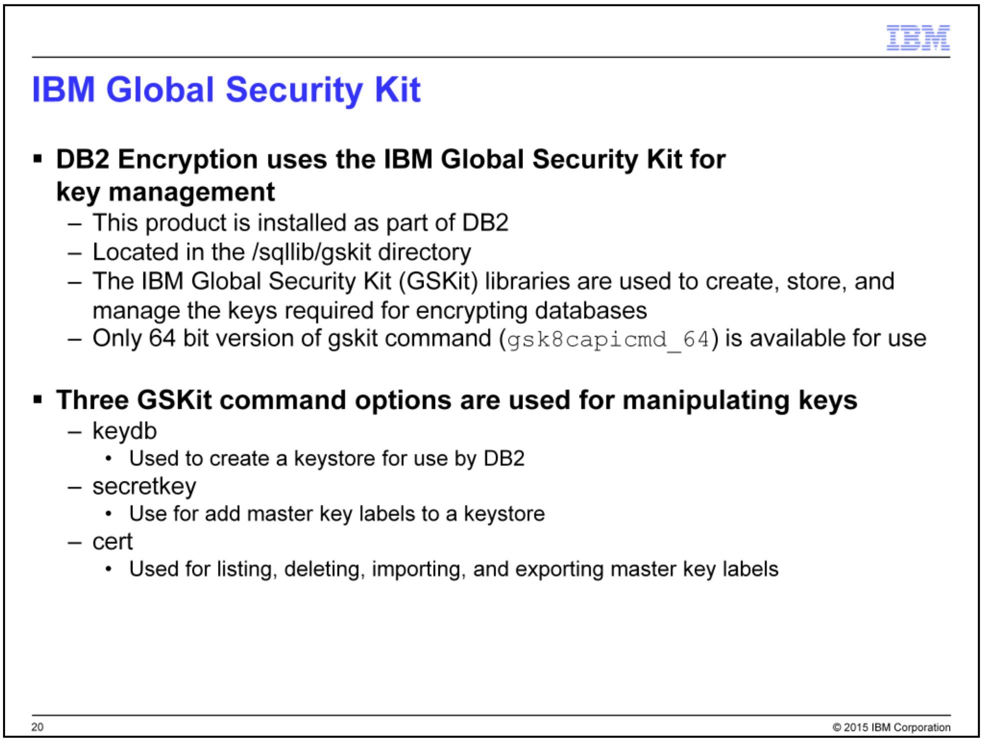
All data that DB2 creates that could have sensitive data in it will be encrypted. Note that in the event of a database failure, some of the data may need to be decrypted for diagnostic purposes. If a customer loses their keystore database (or password), IBM has no facility to retrieve the lost keys.

Note: Some control information within the database is not encrypted, but no user data is exposed. Any user data will be encrypted by DB2.

Master Key Management

© 2015 IBM Corporation

IBM DB2 Encryption Offering is the official name of the encryption product. This section will discuss the product and how a customer can implement encryption at a database level.



IBM Global Security Kit

- **DB2 Encryption uses the IBM Global Security Kit for key management**
  - This product is installed as part of DB2
  - Located in the /sqllib/gskit directory
  - The IBM Global Security Kit (GSKit) libraries are used to create, store, and manage the keys required for encrypting databases
  - Only 64 bit version of gskit command (`gsk8capicmd_64`) is available for use
- **Three GSKit command options are used for manipulating keys**
  - keydb
    - Used to create a keystore for use by DB2
  - secretkey
    - Use for add master key labels to a keystore
  - cert
    - Used for listing, deleting, importing, and exporting master key labels

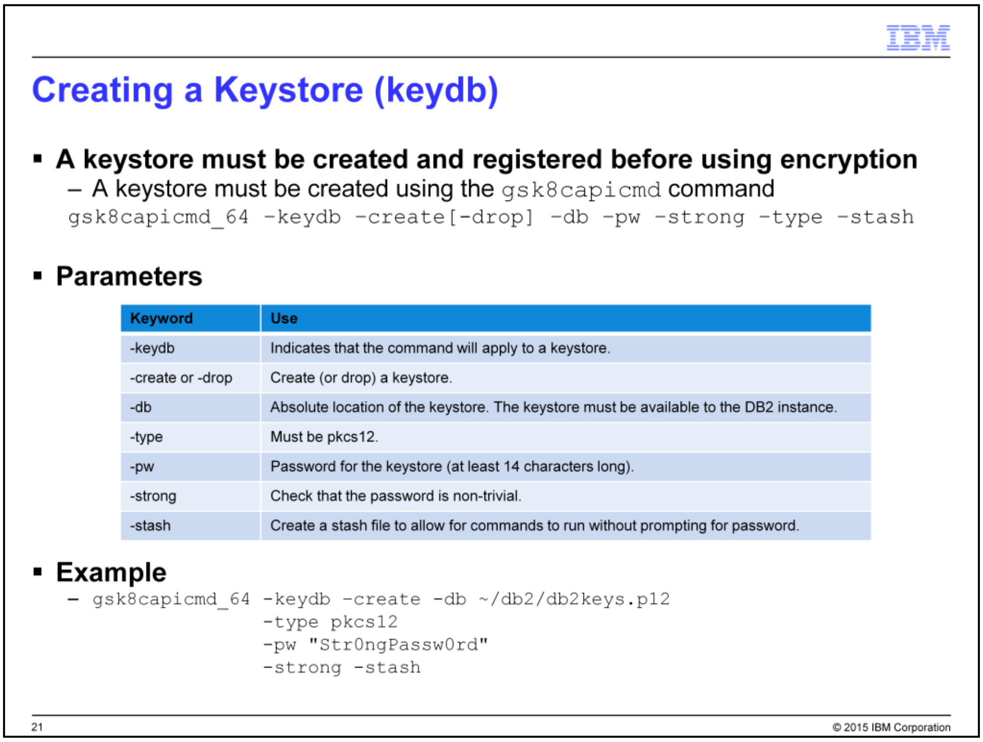
20

© 2015 IBM Corporation

DB2 makes use of the IBM Global Security Kit (GSKit) libraries to create, store, and manage the keys required for encrypting databases.

GSKit is a set of tools and C/C++ programming interfaces that can be used to add secure channels using the TLS protocol to TCP/IP applications (products). It also provides the cryptographic functions, the protocol implementation, and key generation and management functionality for encryption. GSKit ships only compiled object code and header files. GSKit is only for IBM internal use and is not offered for sale as a standalone product, i.e., it is designed for the use in other IBM products only.

Note: The routines that are used with the GSKit have been certified to conform to the FIPS security specifications.



Creating a Keystore (keydb)

- **A keystore must be created and registered before using encryption**
  - A keystore must be created using the `gsk8capicmd` command

```
gsk8capicmd_64 -keydb -create[-drop] -db -pw -strong -type -stash
```

- **Parameters**

Keyword	Use
-keydb	Indicates that the command will apply to a keystore.
-create or -drop	Create (or drop) a keystore.
-db	Absolute location of the keystore. The keystore must be available to the DB2 instance.
-type	Must be pkcs12.
-pw	Password for the keystore (at least 14 characters long).
-strong	Check that the password is non-trivial.
-stash	Create a stash file to allow for commands to run without prompting for password.

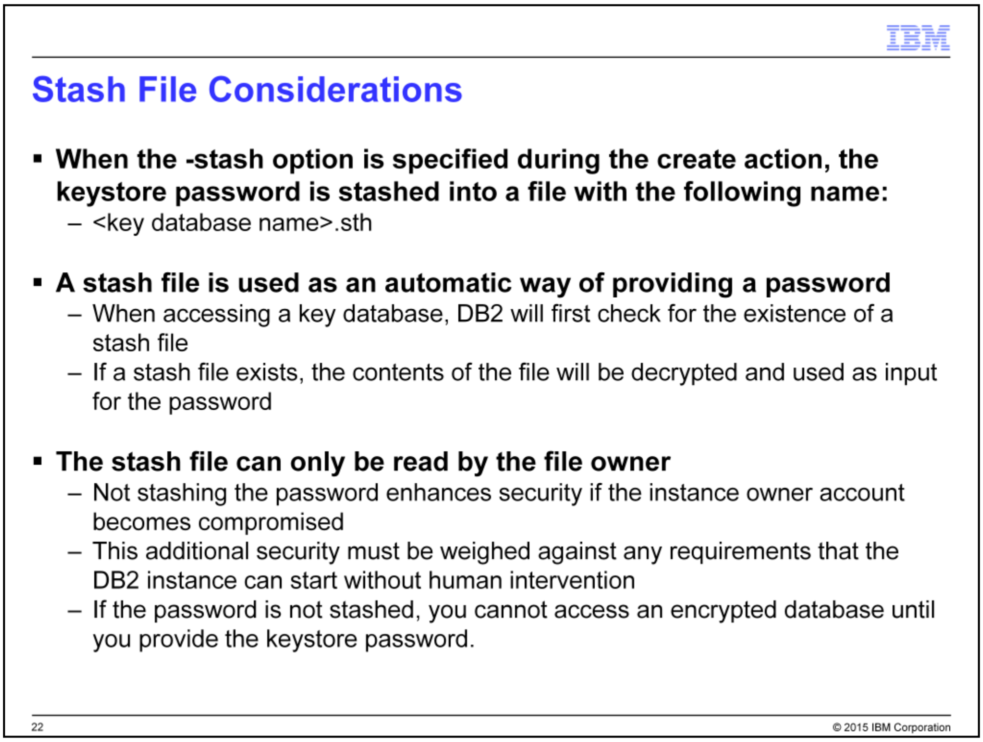
- **Example**

```
– gsk8capicmd_64 -keydb -create -db ~/db2/db2keys.p12
                 -type pkcs12
                 -pw "Str0ngPassw0rd"
                 -strong -stash
```

The GSKIt commands are used to create a PKCS#12-compliant keystore (a storage object for encryption keys). Then the DB2 instance is updated with the location of the keystore and the keystore type and location.

The gsk8capicmd_64 command must be run prior to creating any DB2 databases that use encryption. The keystore that is created must be accessible by the DB2 instance, otherwise none of the encrypted databases can be opened for use.

The parameters listed in the slide are all required, except for the –stash keyword, which is discussed in the next slide.



IBM

Stash File Considerations

- **When the -stash option is specified during the create action, the keystore password is stashed into a file with the following name:**
  - <key database name>.sth
- **A stash file is used as an automatic way of providing a password**
  - When accessing a key database, DB2 will first check for the existence of a stash file
  - If a stash file exists, the contents of the file will be decrypted and used as input for the password
- **The stash file can only be read by the file owner**
  - Not stashing the password enhances security if the instance owner account becomes compromised
  - This additional security must be weighed against any requirements that the DB2 instance can start without human intervention
  - If the password is not stashed, you cannot access an encrypted database until you provide the keystore password.

22 © 2015 IBM Corporation

The GSKit can also create a stash file during keystore creation. A stash file is used as an automatic way of providing a password. When accessing a key database the system will first check for the existence of a stash file. If one exists the contents of the file will be decrypted and used as input for the password. Otherwise, the password will need to be supplied on db2 startup.

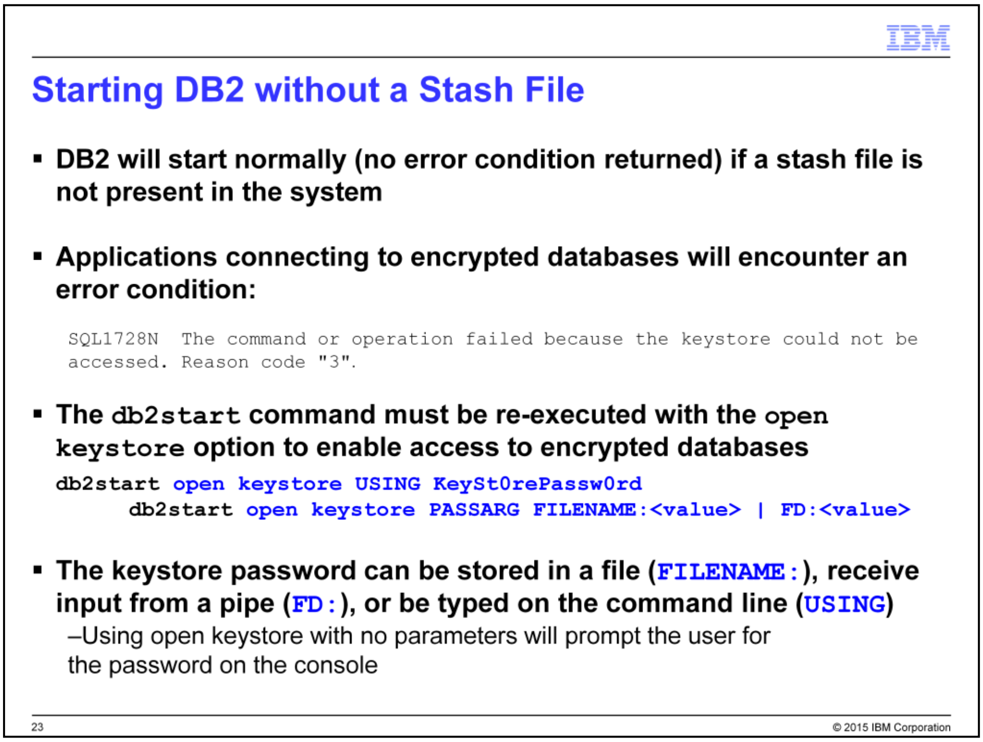
Some customers may want to modify db2start scripts to supply passwords rather than stash the password (i.e. to prevent disk theft or loss)

PASSARG has two forms of supplying the password:

•fd:<file descriptor> where password has been written to open pipe

•filename:<filename> where the first line of the file contains the password

If you start a database without the open keystore parameter, any client wanting to connect to an encrypted database will get a connection failure. You will need to issue the db2start command again with the proper password for the keystore in order to have access opened up to the database.



Starting DB2 without a Stash File

- **DB2 will start normally (no error condition returned) if a stash file is not present in the system**
- **Applications connecting to encrypted databases will encounter an error condition:**

```
SQL1728N  The command or operation failed because the keystore could not be
accessed. Reason code "3".
```

- **The `db2start` command must be re-executed with the `open keystore` option to enable access to encrypted databases**

```
db2start open keystore USING KeySt0rePassw0rd
      db2start open keystore PASSARG FILENAME:<value> | FD:<value>
```

- **The keystore password can be stored in a file (`FILENAME:`), receive input from a pipe (`FD:`), or be typed on the command line (`USING`)**
  - –Using open keystore with no parameters will prompt the user for the password on the console

23

Some customers may want to modify db2start scripts to supply passwords rather than stash the password. If a stash file is not available during DB2 start processing (and no passwords were provided for the keystore), DB2 will start normally, but the encrypted databases cannot be accessed.

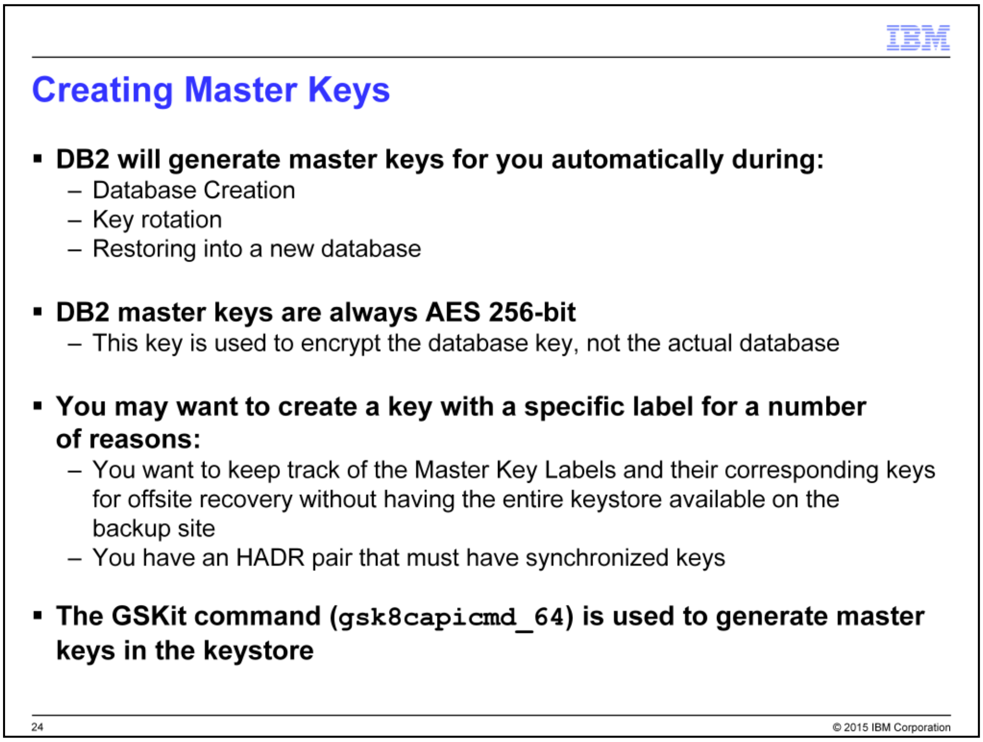
In order to access these encrypted databases, the db2start command must be issued again, but with the password for the stash file supplied using the open keystore USING… option.

PASSARG has two forms of supplying the password:

•fd:<file descriptor> where password has been written to open pipe

•filename:<filename> where the first line of the file contains the password

The customer will need to decide whether or not it is worth the risk of having a stashed keystore password, or delaying the startup of a database due to system maintenance or a restart.



IBM

Creating Master Keys

- **DB2 will generate master keys for you automatically during:**
  - Database Creation
  - Key rotation
  - Restoring into a new database
- **DB2 master keys are always AES 256-bit**
  - This key is used to encrypt the database key, not the actual database
- **You may want to create a key with a specific label for a number of reasons:**
  - You want to keep track of the Master Key Labels and their corresponding keys for offsite recovery without having the entire keystore available on the backup site
  - You have an HADR pair that must have synchronized keys
- **The GSKit command (`gsk8capicmd_64`) is used to generate master keys in the keystore**

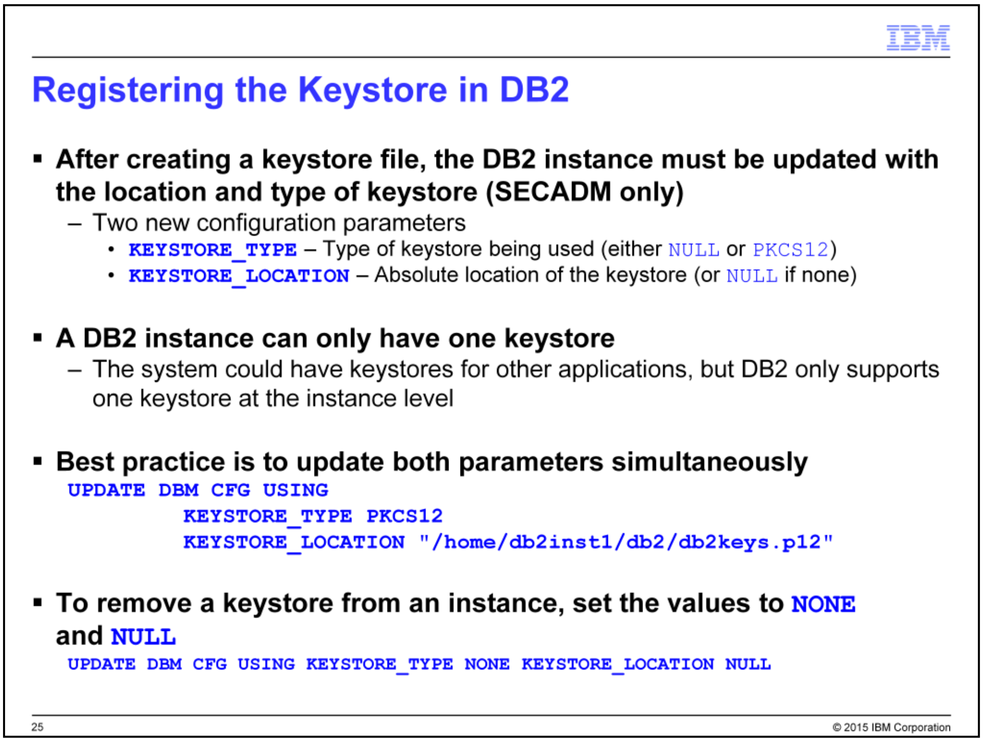
DB2 can do all of the keystore management automatically, including the generation of master key labels. The only customer requirement is to create the initial keystore.

Master keys are used to encrypt the database key. This is the two-level key wrapping that was mentioned earlier in the presentation. DB2 generates a database key automatically (inside the database and invisible to the user). This database key is used to encrypt the contents of the database. The database key is stored within the database itself, but it must be encrypted first by the master key. And it is the master key that is stored in the keystore.

If you do decide to create a master key label, you must create it using the gsk8capicmd_64 and generate the appropriate encryption key (more on this later). There are a couple of situations that creating your own keys is necessary:

•Using different keys for offsite recovery (where the keystore may not be available)

•Using HADR and having a different keystore at the secondary site



Registering the Keystore in DB2

- After creating a keystore file, the DB2 instance must be updated with the location and type of keystore (SECADM only)
  - Two new configuration parameters
    - `KEYSTORE_TYPE` – Type of keystore being used (either `NULL` or `PKCS12`)
    - `KEYSTORE_LOCATION` – Absolute location of the keystore (or `NULL` if none)
- A DB2 instance can only have one keystore
  - The system could have keystores for other applications, but DB2 only supports one keystore at the instance level
- Best practice is to update both parameters simultaneously

```
UPDATE DBM CFG USING
          KEYSTORE_TYPE PKCS12
          KEYSTORE_LOCATION "/home/db2inst1/db2/db2keys.p12"
```

- To remove a keystore from an instance, set the values to `NONE` and `NULL`

```
UPDATE DBM CFG USING KEYSTORE_TYPE NONE KEYSTORE_LOCATION NULL
```

DB2 needs to be made aware of the location and type of keystore that is being used for encryption. There are two new instance parameters that will need to be updated before encryption can be used.

•keystore_type

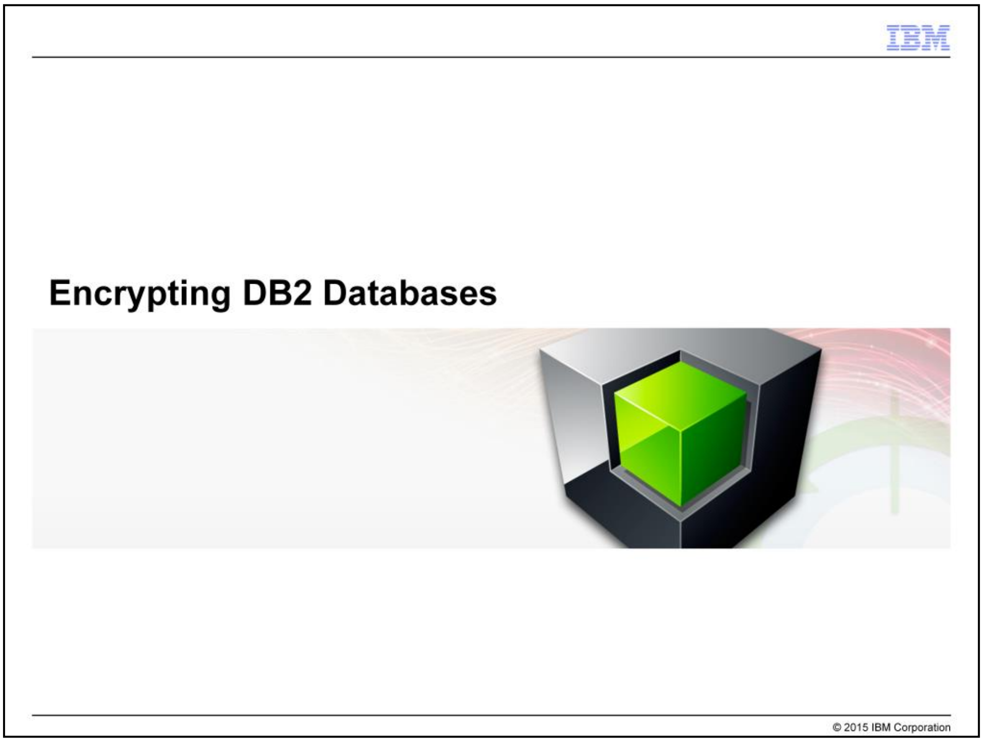
The keystore type can be either NULL or PKCS12. NULL means that there is no keystore defined for this instance, and no databases under this instance are encrypted.

PKCS12 specifies that the keystore type is PKCS #12. The value of the keystore_location configuration parameter is used to configure the location of the keystore. You cannot set keystore_type to PKCS12 unless the keystore_location database manager configuration parameter is set to a non-NULL file name.

•keystore_location

When this value is NULL it means that there is no keystore defined for this instance, and no databases under this instance are encrypted. You can't set keystore_location to NULL unless the keystore_type database manager configuration parameter is set to NULL.

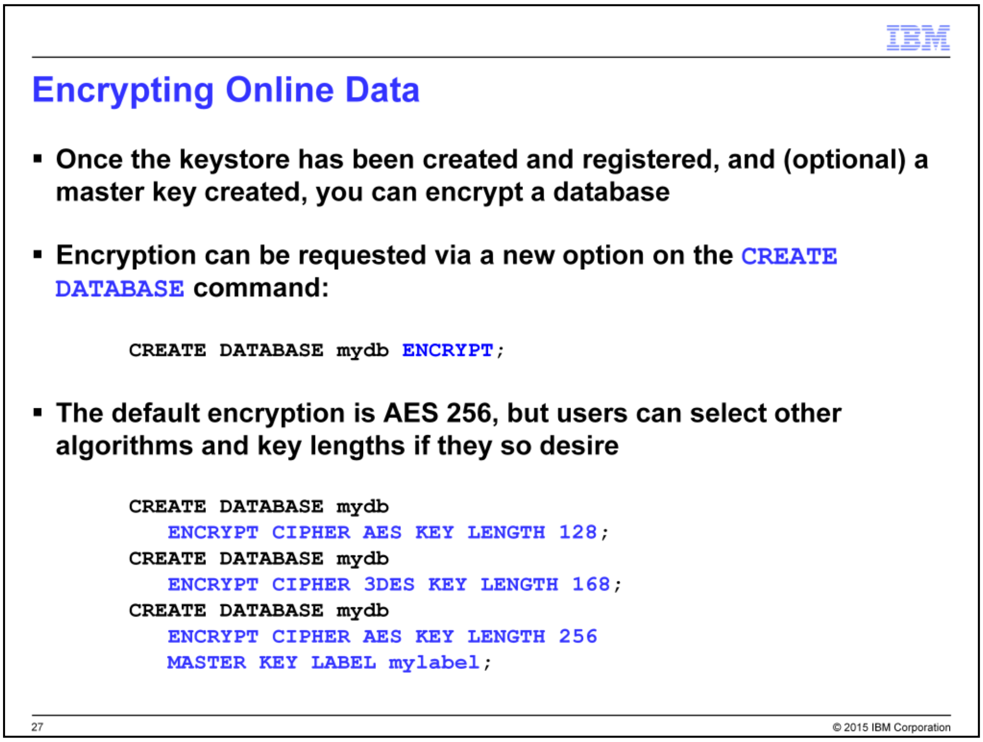
Best practice is to set both parameters at the same time to avoid any errors.



Encrypting DB2 Databases

© 2015 IBM Corporation

This section will explain how DB2 databases can be encrypted and the options that are available to a user.



Encrypting Online Data

- Once the keystore has been created and registered, and (optional) a master key created, you can encrypt a database
- Encryption can be requested via a new option on the CREATE DATABASE command:

```
CREATE DATABASE mydb ENCRYPT;
```

- The default encryption is AES 256, but users can select other algorithms and key lengths if they so desire

```
CREATE DATABASE mydb
   ENCRYPT CIPHER AES KEY LENGTH 128;
CREATE DATABASE mydb
   ENCRYPT CIPHER 3DES KEY LENGTH 168;
CREATE DATABASE mydb
   ENCRYPT CIPHER AES KEY LENGTH 256
   MASTER KEY LABEL mylabel;
```

27

The CREATE DATABASE command has a number of options related to the level of encryption. Two types of encryption ciphers are supported (AES, 3DES), along with a variety of key lengths. The full syntax is:

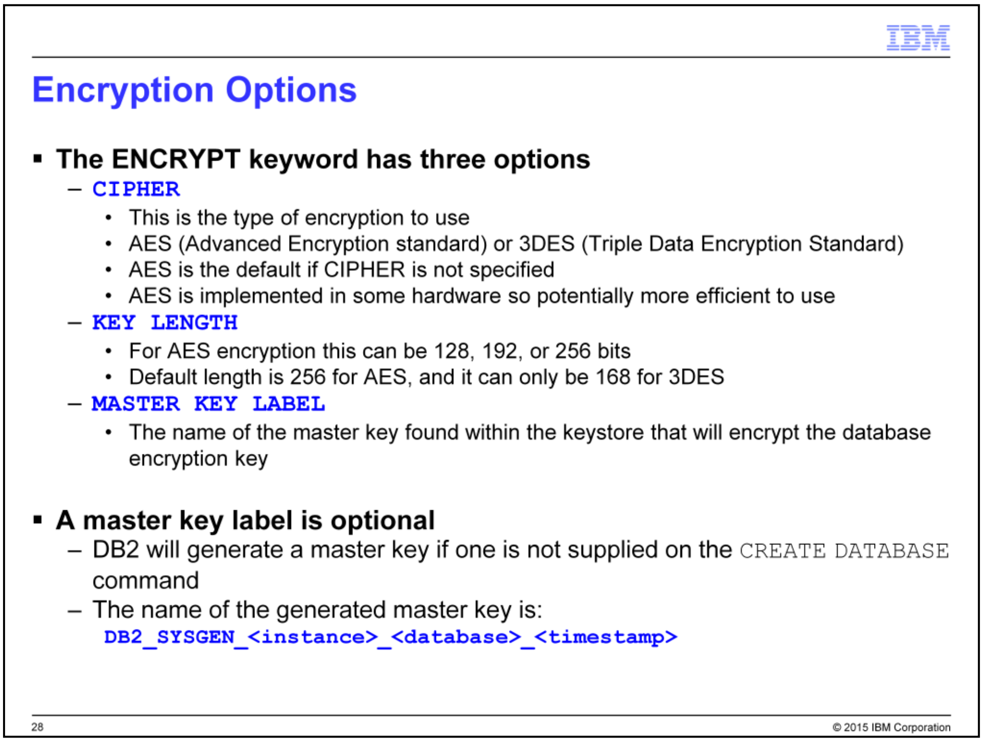
```
CREATE DATABASE <name> ENCRYPT
     CIPHER [ AES | 3DES ]
     KEY LENGTH [ 128 | 168 | 192 | 256 ]
     MASTER KEY LABEL [label]
```

Here is the simplest version of the CREATE DATABASE command that will generate an encrypted database.

```
CREATE DATABASE SECRET ENCRYPT
```

There are no special requirements for users or applications to provide any keys to access the database. All of the normal security features within DB2 would be used to restrict user access to tables and the types of commands administers can issue.

If you supply a master key in the CREATE DATABASE command, the key must exist in the keystore; otherwise DB2 will issue an error message and not create the database.



Encryption Options

- **The ENCRYPT keyword has three options**
  - `CIPHER`
    - This is the type of encryption to use
    - AES (Advanced Encryption standard) or 3DES (Triple Data Encryption Standard)
    - AES is the default if CIPHER is not specified
    - AES is implemented in some hardware so potentially more efficient to use
  - `KEY LENGTH`
    - For AES encryption this can be 128, 192, or 256 bits
    - Default length is 256 for AES, and it can only be 168 for 3DES
  - `MASTER KEY LABEL`
    - The name of the master key found within the keystore that will encrypt the database encryption key
- **A master key label is optional**
  - DB2 will generate a master key if one is not supplied on the `CREATE DATABASE` command
  - The name of the generated master key is:
    `DB2_SYSGEN_<instance>_<database>_<timestamp>`

Cipher

Cipher refers to the type of algorithm that will be used to encrypt the data in the database. DB2 supports two encryption algorithms, AES and 3DES. Both forms of encryption use a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

Key Length

The key length is dependent on which cipher you chose to use for the encryption. The options are:

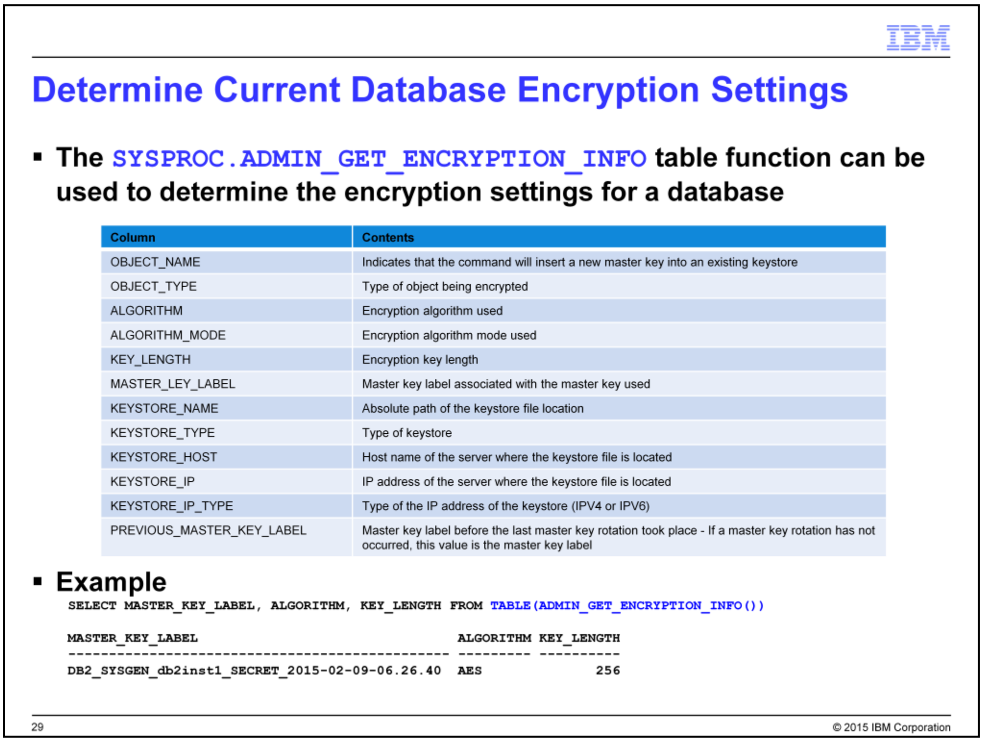
•AES – 128, 192, 256

•3DES – 168

If you do not specify a key length, the default for AES is 256 and the key length can only be 168 for 3DES.

Master Key Label

When you create a database without a Master Key Label, DB2 will automatically generate one for you. The encryption algorithm that is used for encrypting with the master key is always AES. If the master key is automatically generated by the DB2 data server, it is always a 256-bit key. If a master key label is not specified, the database manager automatically generates a master key and inserts it into the keystore file.



Determine Current Database Encryption Settings

- The SYSPROC.ADMIN_GET_ENCRYPTION_INFO table function can be used to determine the encryption settings for a database

Column	Contents
OBJECT_NAME	Indicates that the command will insert a new master key into an existing keystore
OBJECT_TYPE	Type of object being encrypted
ALGORITHM	Encryption algorithm used
ALGORITHM_MODE	Encryption algorithm mode used
KEY_LENGTH	Encryption key length
MASTER_LEY_LABEL	Master key label associated with the master key used
KEYSTORE_NAME	Absolute path of the keystore file location
KEYSTORE_TYPE	Type of keystore
KEYSTORE_HOST	Host name of the server where the keystore file is located
KEYSTORE_IP	IP address of the server where the keystore file is located
KEYSTORE_IP_TYPE	Type of the IP address of the keystore (IPV4 or IPV6)
PREVIOUS_MASTER_KEY_LABEL	Master key label before the last master key rotation took place - If a master key rotation has not occurred, this value is the master key label

- Example

```
SELECT MASTER_KEY_LABEL, ALGORITHM, KEY_LENGTH FROM TABLE(ADMIN_GET_ENCRYPTION_INFO())

MASTER_KEY_LABEL                                 ALGORITHM KEY_LENGTH
------------------------------------------------ --------- ----------
DB2_SYSGEN_db2inst1_SECRET_2015-02-09-06.26.40   AES              256
```

29 © 2015 IBM Corporation

The built in table function SYSPROC.ADMIN_GET_ENCRYPTION_INFO() returns information regarding the current encryption settings in the database, including the master key label.

The ADMIN_GET_ENCRYPTION_INFO() function returns the following information:

•OBJECT_NAME – Name of the object being encrypted.

•OBJECT_TYPE – Type of object being encrypted.

•ALGORITHM – Encryption algorithm used.

•ALGORITHM_MODE – Encryption algorithm mode used.

•KEY_LENGTH – Encryption key length.

•MASTER_KEY_LABEL – Master key label associated with the master key used.

•KEYSTORE_NAME – Absolute path of the keystore file location.

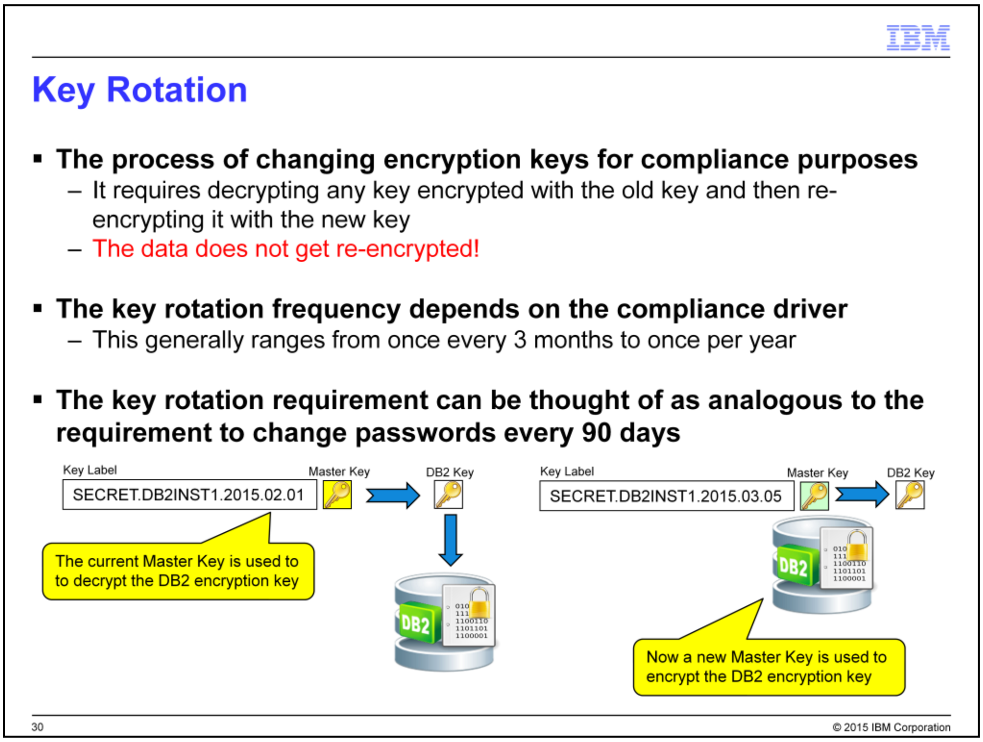
•KEYSTORE_TYPE – Type of keystore.

•KEYSTORE_HOST – Host name of the server where the keystore file is located.

•KEYSTORE_IP – IP address of the server where the keystore file is located.

•KEYSTORE_IP_TYPE – Type of the IP address of the keystore (IPV4 or IPV6).

•PREVIOUS_MASTER_KEY_LABEL – Master key label before the last master key rotation took place. If a master key rotation has not occurred, this value is the master key label



Key Rotation

- **The process of changing encryption keys for compliance purposes**
  - It requires decrypting any key encrypted with the old key and then re-encrypting it with the new key
  - The data does not get re-encrypted!
- **The key rotation frequency depends on the compliance driver**
  - This generally ranges from once every 3 months to once per year
- **The key rotation requirement can be thought of as analogous to the requirement to change passwords every 90 days**

The current Master Key is used to to decrypt the DB2 encryption key

Now a new Master Key is used to encrypt the DB2 encryption key

Key rotation refers to the changing of encryption keys for compliance purposes. This is very similar to changing a password every 90 days.

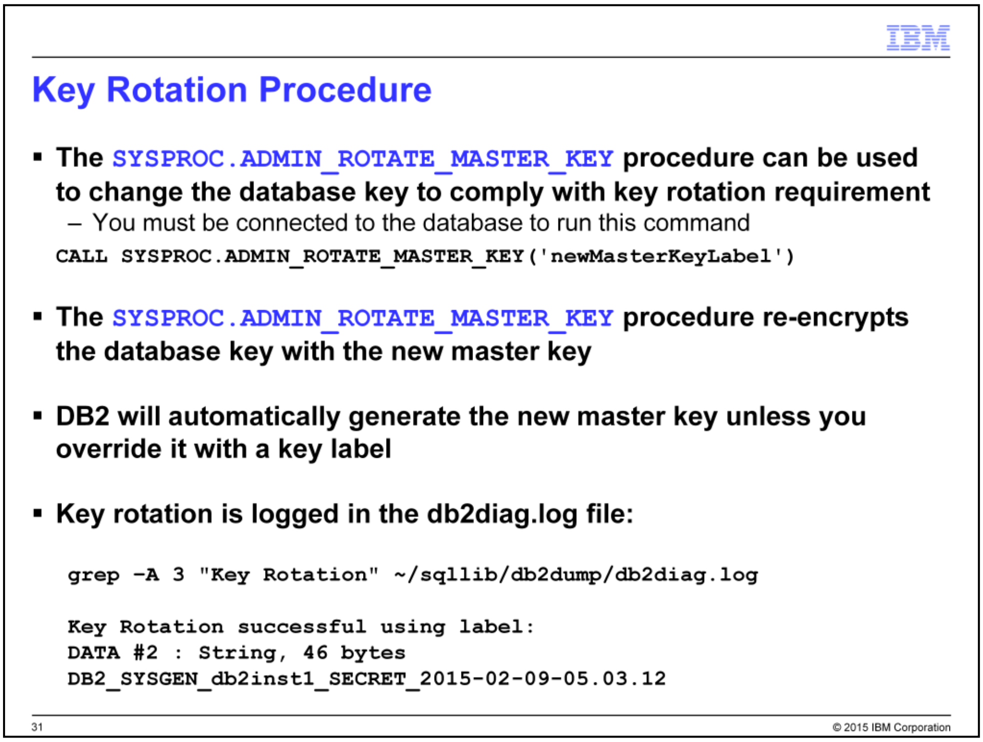
Key rotation is done WITHIN the database. The process is as follows.

•The master key for the database is used to decrypt the database key (this is done when the database is started)

•A new master key is generated for the database

•This new master key is used to encrypt the database key

The database encryption key is never changed. If you did change the database encryption key, you would have to re-encrypt the entire database. Instead, only the key that encrypts the database is re-encrypted. This avoids having to re-encrypt the entire database.



Key Rotation Procedure

- **The SYSPROC.ADMIN_ROTATE_MASTER_KEY procedure can be used to change the database key to comply with key rotation requirement**
  - You must be connected to the database to run this command

```
CALL SYSPROC.ADMIN_ROTATE_MASTER_KEY('newMasterKeyLabel')
```

- **The SYSPROC.ADMIN_ROTATE_MASTER_KEY procedure re-encrypts the database key with the new master key**

- **DB2 will automatically generate the new master key unless you override it with a key label**

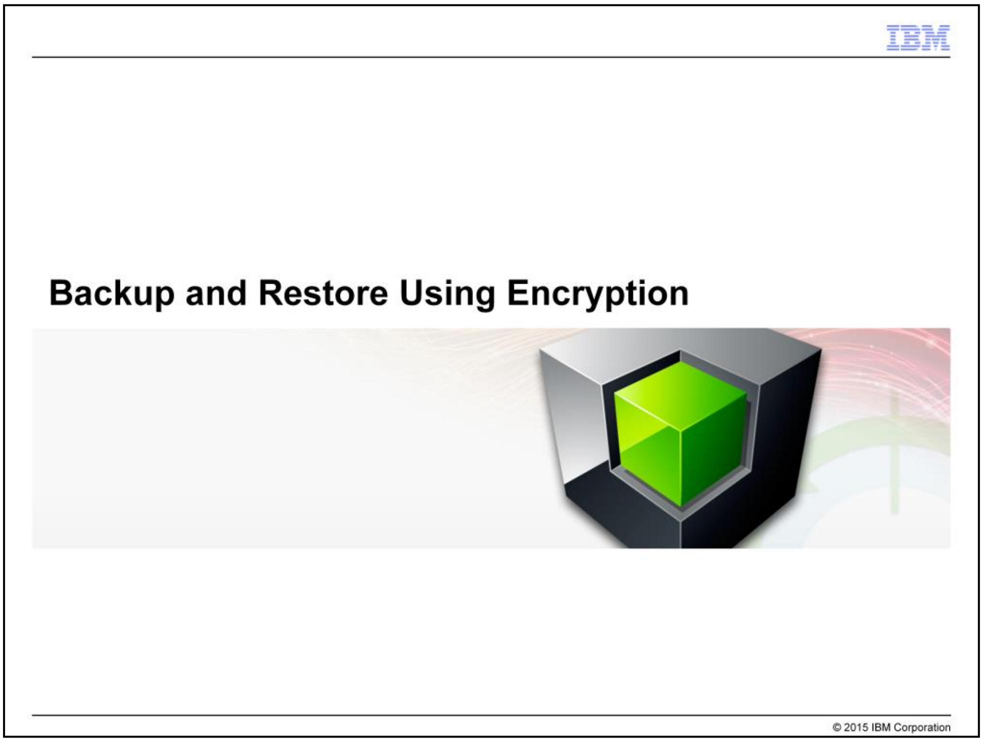
- **Key rotation is logged in the db2diag.log file:**

```
grep –A 3 "Key Rotation" ~/sqllib/db2dump/db2diag.log

Key Rotation successful using label:
DATA #2 : String, 46 bytes
DB2_SYSGEN_db2inst1_SECRET_2015-02-09-05.03.12
```

DB2 native encryption allows you to rotate your database MK to comply with your corporate security policies. You rotate your database MK by calling the new ADMIN_ROTATE_MASTER_KEY procedure.

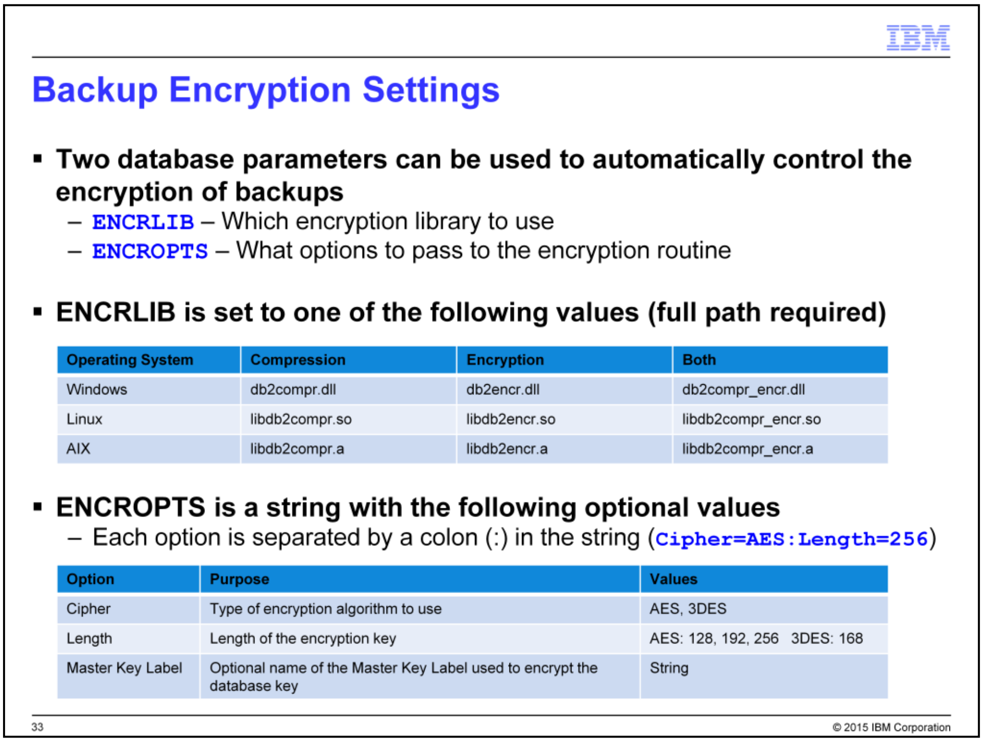
The procedure decrypts your database DEK with the old MK and then re-encrypts it with the new MK. You have 2 options when calling the ADMIN_ROTATE_MASTER_KEY procedure. You can either provide a label for the desired new MK or use the default. When using the default, DB2 automatically generates a new master key and adds it to the keystore on your behalf. Then, it rotates the current database MK to this newly generated MK.

The ADMIN_GET_ENCRYPTION_INFO function can be used to get information about the encryption used for the current database.



Backup and Restore Using Encryption

This section will describe the backup and restore considerations when using encryption.



Backup Encryption Settings

- **Two database parameters can be used to automatically control the encryption of backups**
  - **ENCRLIB** – Which encryption library to use
  - **ENCROPTS** – What options to pass to the encryption routine
- **ENCRLIB is set to one of the following values (full path required)**

Operating System	Compression	Encryption	Both
Windows	db2compr.dll	db2encr.dll	db2compr_encr.dll
Linux	libdb2compr.so	libdb2encr.so	libdb2compr_encr.so
AIX	libdb2compr.a	libdb2encr.a	libdb2compr_encr.a

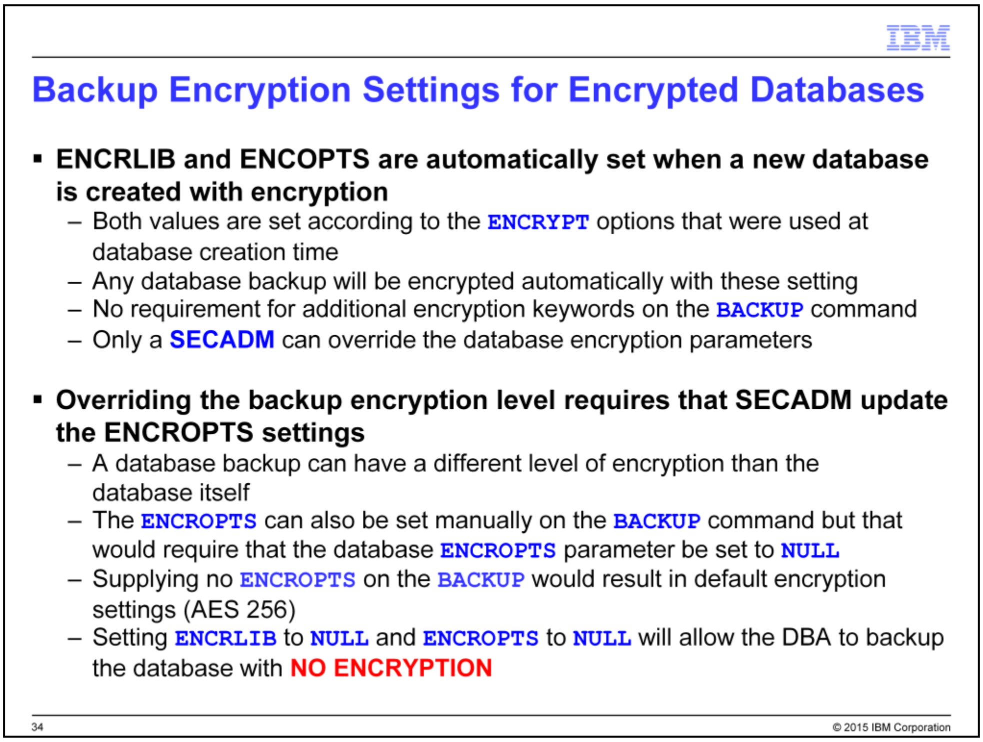
- **ENCROPTS is a string with the following optional values**
  - Each option is separated by a colon (:) in the string (`Cipher=AES:Length=256`)

Option	Purpose	Values
Cipher	Type of encryption algorithm to use	AES, 3DES
Length	Length of the encryption key	AES: 128, 192, 256 3DES: 168
Master Key Label	Optional name of the Master Key Label used to encrypt the database key	String

The encryption for backup images is independent of online database encryption. That is, you can choose to encrypt your backup images even if your online database is not encrypted. You can request an encrypted backup image by explicitly specifying the ENCRYPT option of the BACKUP DATABASE command.

Alternatively, you can enforce and automate backup images encryption by configuring the new ENCRLIB and ENCROPTS database configuration parameters.

This chart lists the names of the libraries that are used compression, encryption, and a combination of encryption and compression. If you are setting the ENCRLIB database parameter, you must specify the absolute file name of the encryption library (/home/db2inst1/sqllib/lib/libdb2encr.so) rather than just the library name. On the backup command, you normally only need to supply the library name (libdb2encr.so).



IBM

Backup Encryption Settings for Encrypted Databases

- **ENCRLIB and ENCOPTS are automatically set when a new database is created with encryption**
  - Both values are set according to the **ENCRYPT** options that were used at database creation time
  - Any database backup will be encrypted automatically with these setting
  - No requirement for additional encryption keywords on the **BACKUP** command
  - Only a **SECADM** can override the database encryption parameters
- **Overriding the backup encryption level requires that SECADM update the ENCROPTS settings**
  - A database backup can have a different level of encryption than the database itself
  - The **ENCROPTS** can also be set manually on the **BACKUP** command but that would require that the database **ENCROPTS** parameter be set to **NULL**
  - Supplying no **ENCROPTS** on the **BACKUP** would result in default encryption settings (AES 256)
  - Setting **ENCRLIB** to **NULL** and **ENCROPTS** to **NULL** will allow the DBA to backup the database with **NO ENCRYPTION**

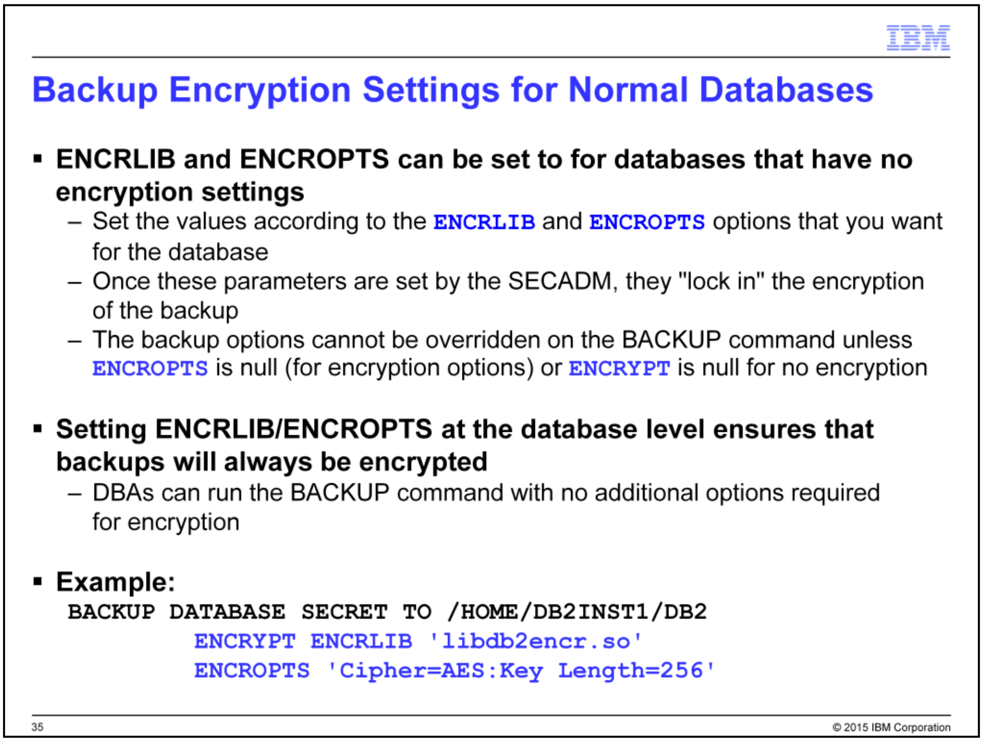
34

When you create an encrypted database, the encrlib and encropts database configuration parameters are set such that subsequent database backup operations use the native DB2 encryption library with options that were specified at database creation time.

The encryption for backup images is independent of online database encryption. That is, you can choose to encrypt your backup images even if your online database is not encrypted. You can request an encrypted backup image by explicitly specifying the ENCRYPT option of the BACKUP DATABASE command. Alternatively, you can enforce and automate backup images encryption by configuring the new ENCRLIB and ENCROPTS database configuration parameters.

Notes:

If you do not want to backup the database with encryption, you need to remove the ENCRLIB and ENCROPTS settings. Only a SECADM is authorized to update the ENCRLIB/ENCROPTS settings on a database.



IBM

Backup Encryption Settings for Normal Databases

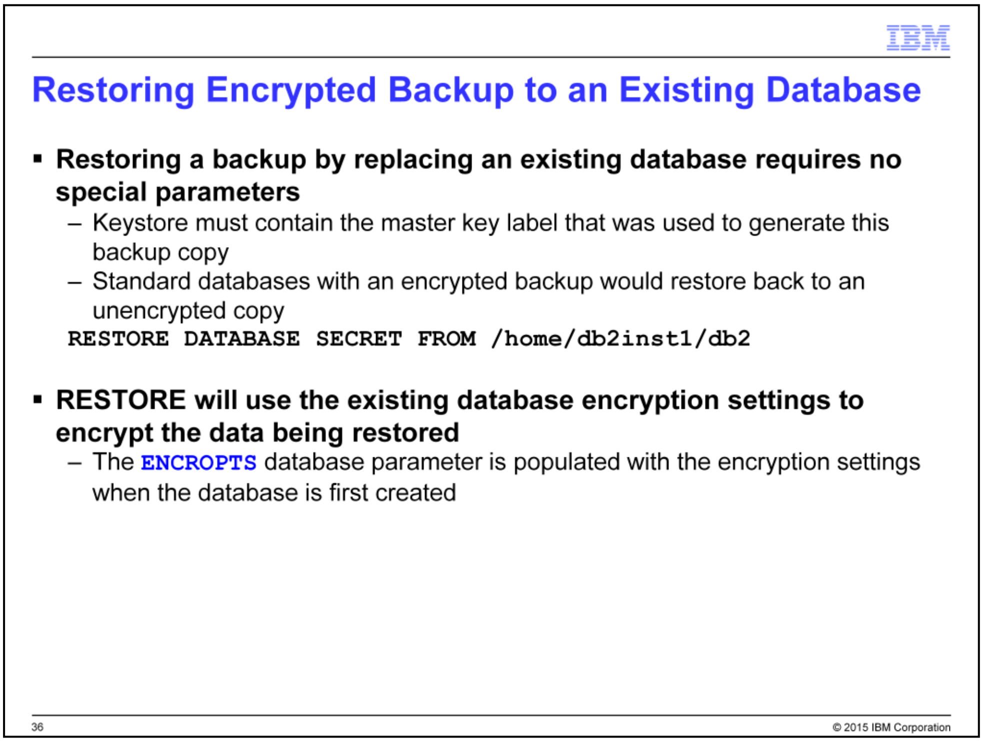
- **ENCRLIB and ENCROPTS can be set to for databases that have no encryption settings**
  - Set the values according to the `ENCRLIB` and `ENCROPTS` options that you want for the database
  - Once these parameters are set by the SECADM, they "lock in" the encryption of the backup
  - The backup options cannot be overridden on the BACKUP command unless `ENCROPTS` is null (for encryption options) or `ENCRYPT` is null for no encryption
- **Setting ENCRLIB/ENCROPTS at the database level ensures that backups will always be encrypted**
  - DBAs can run the BACKUP command with no additional options required for encryption
- **Example:**

```
BACKUP DATABASE SECRET TO /HOME/DB2INST1/DB2
         ENCRYPT ENCRLIB 'libdb2encr.so'
         ENCROPTS 'Cipher=AES:Key Length=256'
```

35

For normal databases (with no encryption set), you can make use of the ENCRLIB and ENCROPTS parameters on the backup command. This gives you the option of having the backups encrypted for offsite storage. When you create a standard database, both of these parameters are set to null.

You can request an encrypted backup image by explicitly specifying the ENCRYPT/ENCRLIB option of the BACKUP DATABASE command.



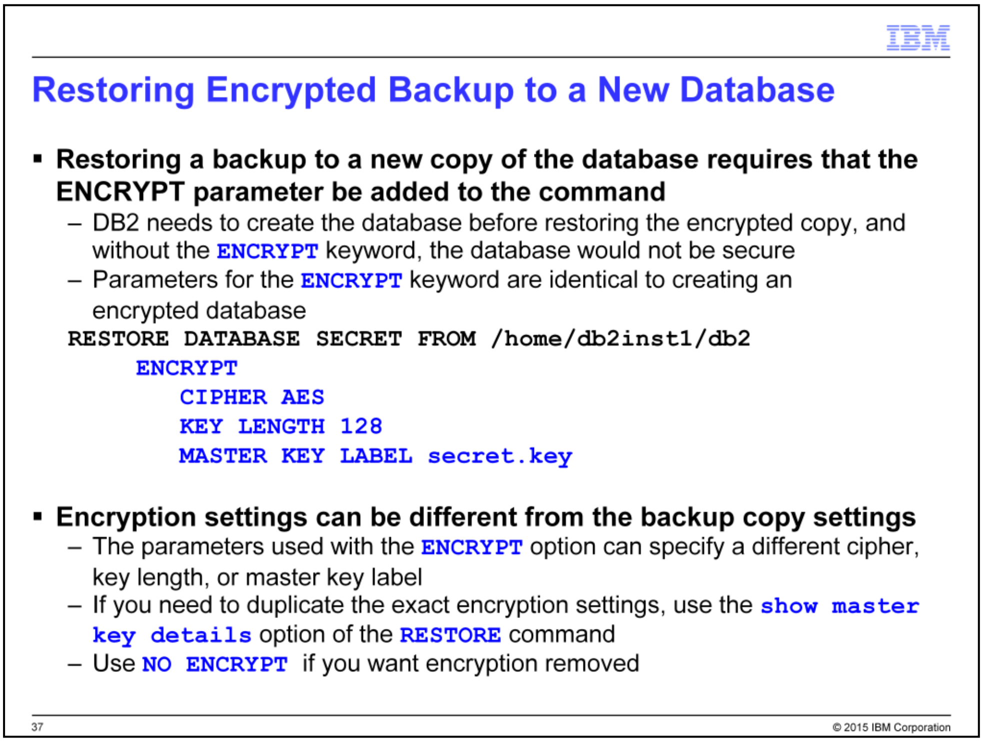
Restoring Encrypted Backup to an Existing Database

- **Restoring a backup by replacing an existing database requires no special parameters**
  - Keystore must contain the master key label that was used to generate this backup copy
  - Standard databases with an encrypted backup would restore back to an unencrypted copy

```
RESTORE DATABASE SECRET FROM /home/db2inst1/db2
```

- **RESTORE will use the existing database encryption settings to encrypt the data being restored**
  - The `ENCROPTS` database parameter is populated with the encryption settings when the database is first created

Restoring an encrypted backup on top of an existing database requires no additional parameters. The existing encryption settings of the database will be used as the data is being restored.



Restoring Encrypted Backup to a New Database

- **Restoring a backup to a new copy of the database requires that the ENCRYPT parameter be added to the command**
  - DB2 needs to create the database before restoring the encrypted copy, and without the `ENCRYPT` keyword, the database would not be secure
  - Parameters for the `ENCRYPT` keyword are identical to creating an encrypted database

```
RESTORE DATABASE SECRET FROM /home/db2inst1/db2
     ENCRYPT
        CIPHER AES
        KEY LENGTH 128
        MASTER KEY LABEL secret.key
```

- **Encryption settings can be different from the backup copy settings**
  - The parameters used with the `ENCRYPT` option can specify a different cipher, key length, or master key label
  - If you need to duplicate the exact encryption settings, use the `show master key details` option of the `RESTORE` command
  - Use `NO ENCRYPT` if you want encryption removed

What happens during the RESTORE process is that the database is created first, and then the contents of the backup are placed into the new database. Unless you specify the encryption options, the new database is created without appropriate encryption and the restore of the backup image will fail.

To fix this, we need to add the encryption options to the command. These are exactly the same parameters as found on the CREATE DATABASE command:

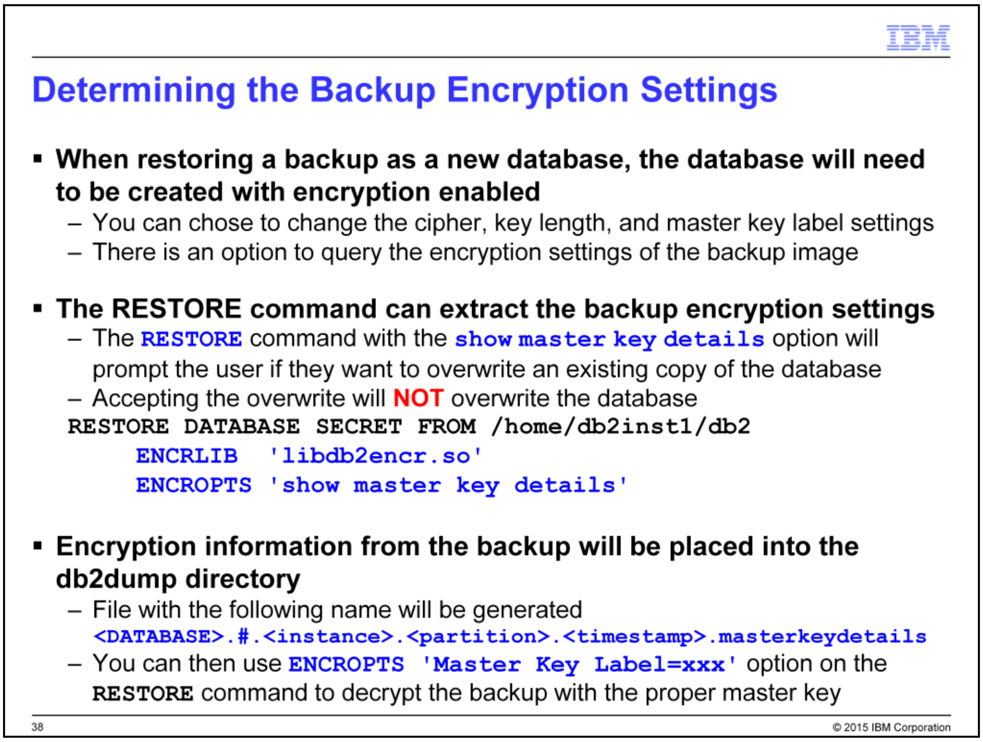
•Encrypt – Required keyword

•Cipher - AES or 3DES

•Key Length - 128, 168, 192, 256

•Master Key Label

Adding these options to the RESTORE command will allow the recovery to proceed. The encryption options on restore can also be different than what the database was originally backed up with.



Determining the Backup Encryption Settings

- **When restoring a backup as a new database, the database will need to be created with encryption enabled**
  - You can chose to change the cipher, key length, and master key label settings
  - There is an option to query the encryption settings of the backup image
- **The RESTORE command can extract the backup encryption settings**
  - The `RESTORE` command with the `show master key details` option will prompt the user if they want to overwrite an existing copy of the database
  - Accepting the overwrite will **NOT** overwrite the database

```
RESTORE DATABASE SECRET FROM /home/db2inst1/db2
     ENCRLIB  'libdb2encr.so'
     ENCROPTS 'show master key details'
```

- **Encryption information from the backup will be placed into the db2dump directory**
  - File with the following name will be generated `<DATABASE>.#.<instance>.<partition>.<timestamp>.masterkeydetails`
  - You can then use `ENCROPTS 'Master Key Label=xxx'` option on the `RESTORE` command to decrypt the backup with the proper master key

38

© 2015 IBM Corporation

If you are unsure of what settings were used when a backup was created, you can use the 'show master key details' option in the ENCRLIB setting to dump the information into a file.

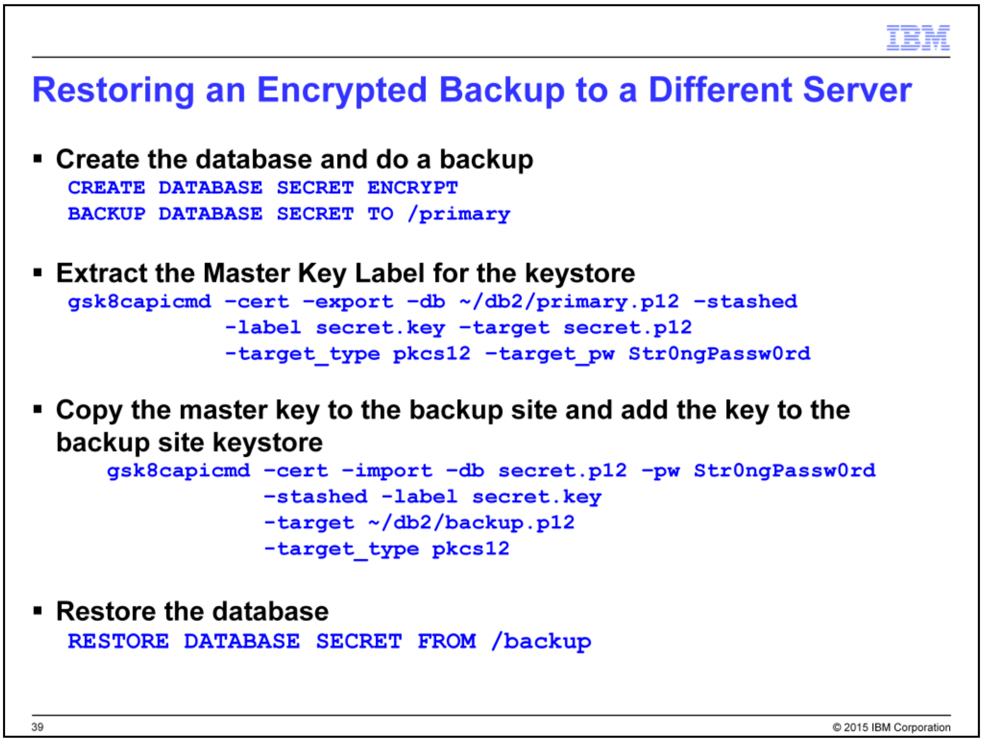
Then the restore command runs it will prompt the user on whether or not the existing database should be overwritten. This does NOT happen during the restore process. So while the message seems to imply that the database will be replaced, it is not.

A file will be generated in the db2dump directory with the format:

```
<DATABASE>.#.<instance>.<partition>.<timestamp>.masterkeydetails
```

The contents of the file will contain detailed information on the encryption settings of the backup.

```
KeyStore Type: PKCS12
KeyStore Location: /home/db2inst1/db2/db2keys.p12
KeyStore Host Name: localhost.localdomain
KeyStore IP Address: 127.0.0.1
KeyStore IP Address Type: IPV4
Encryption Algorithm: AES
Encryption Algorithm Mode: CBC
Encryption Key Length: 256
Master Key Label: DB2_SYSGEN_db2inst1_SECRET_2015-02-09-04.28.34[d
```



Restoring an Encrypted Backup to a Different Server

- **Create the database and do a backup**

```
CREATE DATABASE SECRET ENCRYPT
BACKUP DATABASE SECRET TO /primary
```

- **Extract the Master Key Label for the keystore**

```
gsk8capicmd -cert -export -db ~/db2/primary.p12 -stashed
            -label secret.key -target secret.p12
            -target_type pkcs12 -target_pw Str0ngPassw0rd
```

- **Copy the master key to the backup site and add the key to the backup site keystore**

```
   gsk8capicmd -cert -import -db secret.p12 -pw Str0ngPassw0rd
               -stashed -label secret.key
               -target ~/db2/backup.p12
               -target_type pkcs12
```

- **Restore the database**

```
RESTORE DATABASE SECRET FROM /backup
```

39

DB2 relies on the keystore file and stash file to get access to the key required to decrypt the database. When you move a backup copy to another server, you may have a different keystore with different master keys.

You have a couple of options on how a backup image can be restored. One is to export the master key from the original keystore (with the gsk8capicmd_64 -cert -export command). The key that is exported can then be imported into the keystore on the second system. You may also have the keyfile that you created when generating the master key. If so, you could just take that file and recreate the key at the second site.

The other option is to make the entire keystore available on the second server, but this would mean replacing the contents of the file at the second site. This would not be practical if there were production systems running there.

Finally, you can use the following technique to move a backup to another server:

•Create a new master key label to be used for the backup: label4systemb

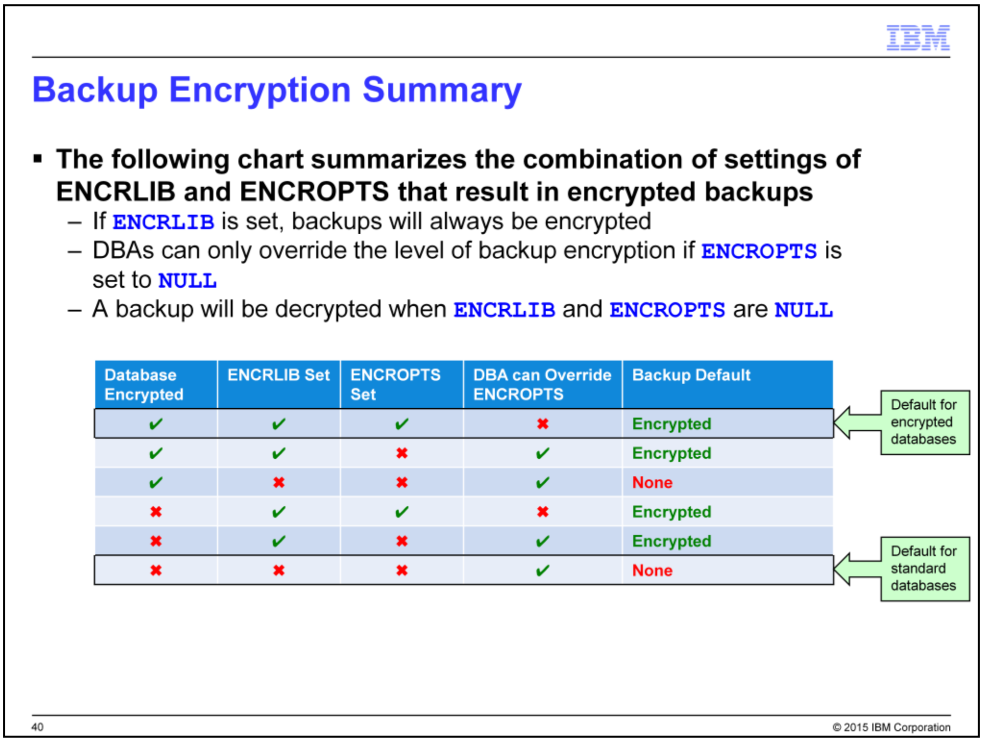
•Extract the key from the keystore on system A, securely copy it to system B, import it into the keystore on system B.

•Take the encrypted backup and specify 'master key label=label4systemb' via encropts

•Copy the encrypted backup to system B

•Restore the encrypted backup on system B

You will note the importance of the keystore file. It is critical that the keystore be frequently backed up and stored in a safe place. In addition, this file needs to be accessed only by the instance owner, and no one else.



IBM

Backup Encryption Summary

- **The following chart summarizes the combination of settings of ENCRLIB and ENCROPTS that result in encrypted backups**
  - If `ENCRLIB` is set, backups will always be encrypted
  - DBAs can only override the level of backup encryption if `ENCROPTS` is set to `NULL`
  - A backup will be decrypted when `ENCRLIB` and `ENCROPTS` are `NULL`

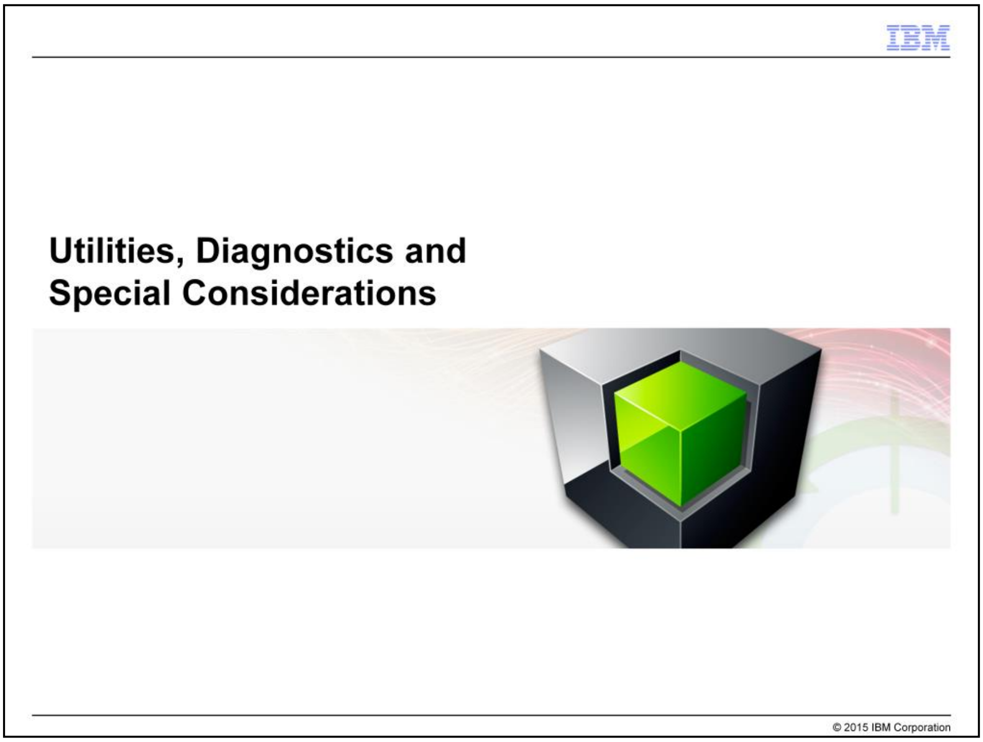
Database Encrypted	ENCRLIB Set	ENCROPTS Set	DBA can Override ENCROPTS	Backup Default
✔	✔	✔	✖	Encrypted
✔	✔	✖	✔	Encrypted
✔	✖	✖	✔	None
✖	✔	✔	✖	Encrypted
✖	✔	✖	✔	Encrypted
✖	✖	✖	✔	None

Default for encrypted databases

Default for standard databases

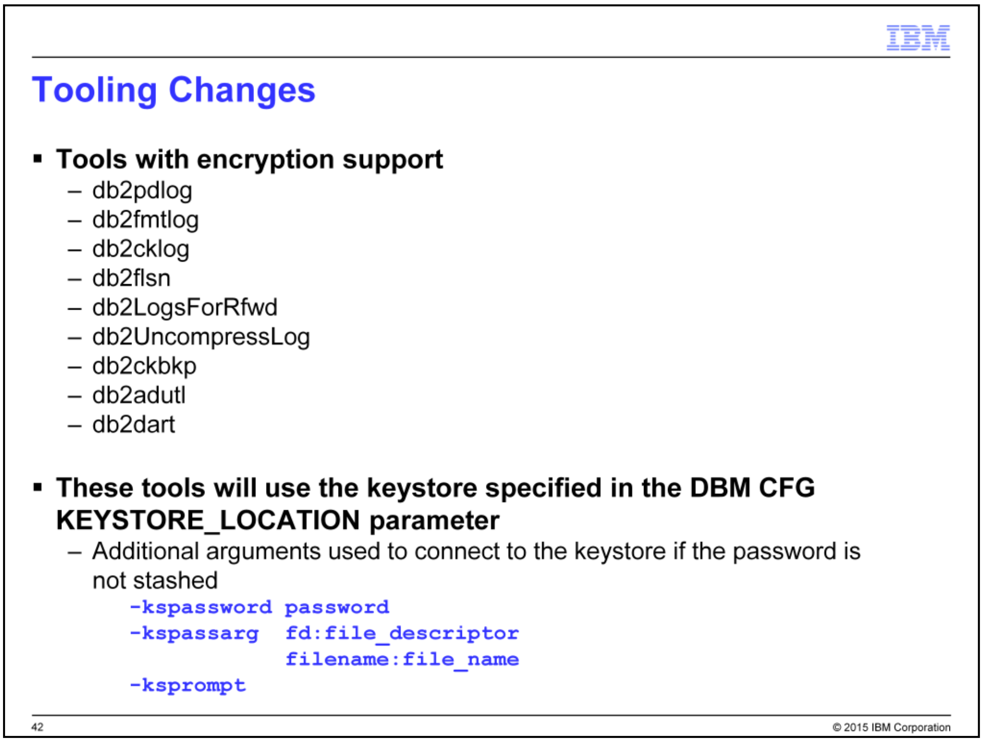
This slide summarizes the types of backup you can perform on encrypted and normal databases. Note that if ENCRLIB and ENCROPTS are set, there is no way for a DBA to override the settings.

If ENCROPTS is not set, the backup will always be encrypted, but the DBA can specify the encryption level, the cipher used, and the master key label.



Utilities, Diagnostics and Special Considerations

This section discusses some of the miscellaneous items associated with encryption.



Tooling Changes

- **Tools with encryption support**
  - db2pdlog
  - db2fmtlog
  - db2cklog
  - db2flsn
  - db2LogsForRfwd
  - db2UncompressLog
  - db2ckbkp
  - db2adutl
  - db2dart
- **These tools will use the keystore specified in the DBM CFG KEYSTORE_LOCATION parameter**
  - Additional arguments used to connect to the keystore if the password is not stashed

```
-kspassword password
-kspassarg  fd:file_descriptor
            filename:file_name
-ksprompt
```

42

A number of DB2 tools have been modified to support encrypted databases. The utilities will always look for the keystore based on the KEYSTORE_LOCATION parameter that was set at the instance level. There is no other way to specify the keystore location.

The additional arguments that are added to these utilities includes the ability to supply the keystore password in either a file, a named pipe, or by prompting on the command line.

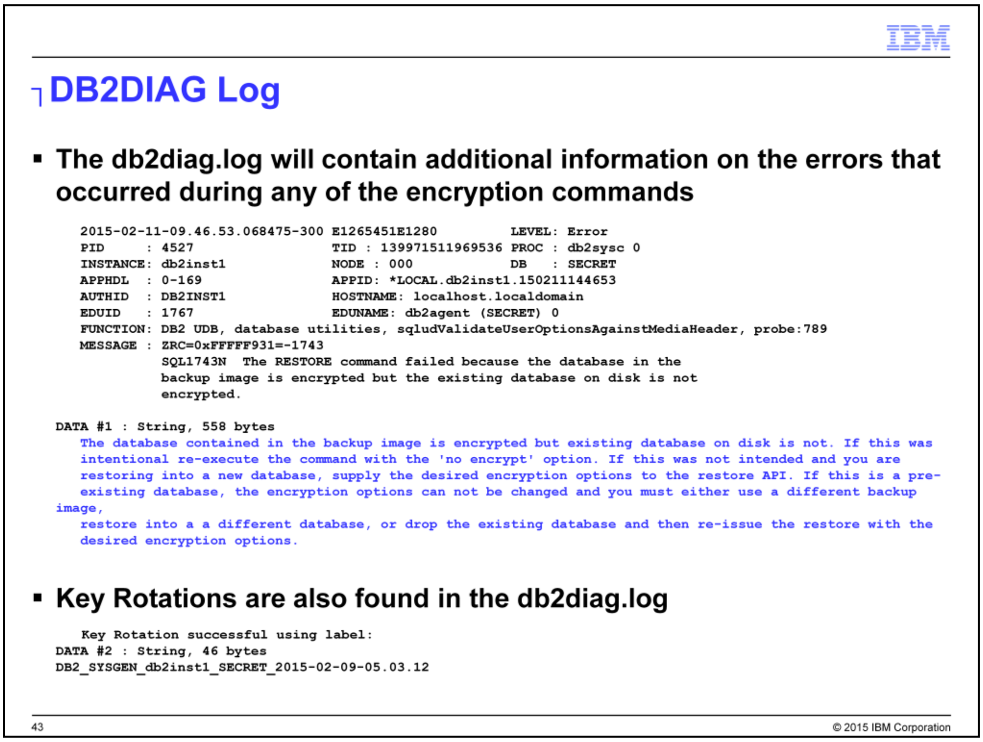
There are three utilities which do not support encryption.

- db2pxlog
- db2PatchLog
- db2logscan

The following products do not support encryption at this time

- Recovery Expert
- Shadow tables [encryption is supported, but the data that is staged will be decrypted]

The recovery expert does support encryption yet.



DB2DIAG Log

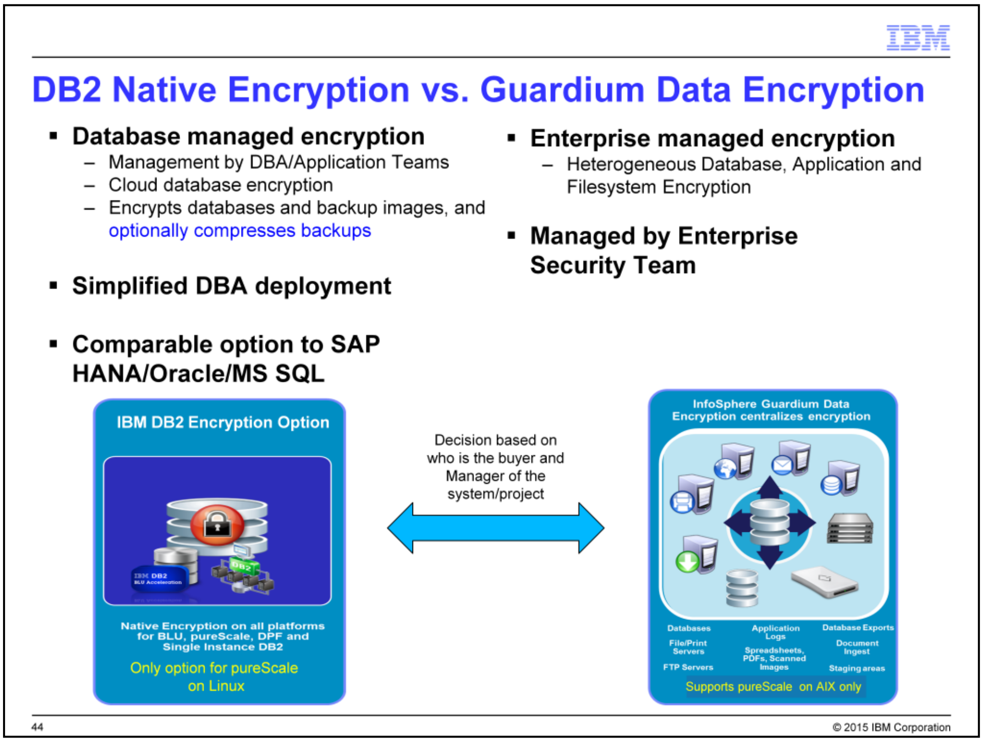
- **The db2diag.log will contain additional information on the errors that occurred during any of the encryption commands**

```
   2015-02-11-09.46.53.068475-300 E1265451E1280         LEVEL: Error
   PID     : 4527                 TID : 139971511969536 PROC : db2sysc 0
   INSTANCE: db2inst1             NODE : 000            DB   : SECRET
   APPHDL  : 0-169                APPID: *LOCAL.db2inst1.150211144653
   AUTHID  : DB2INST1             HOSTNAME: localhost.localdomain
   EDUID   : 1767                 EDUNAME: db2agent (SECRET) 0
   FUNCTION: DB2 UDB, database utilities, sqludValidateUserOptionsAgainstMediaHeader, probe:789
   MESSAGE : ZRC=0xFFFFF931=-1743
             SQL1743N  The RESTORE command failed because the database in the
             backup image is encrypted but the existing database on disk is not
             encrypted.

DATA #1 : String, 558 bytes
   The database contained in the backup image is encrypted but existing database on disk is not. If this was
   intentional re-execute the command with the 'no encrypt' option. If this was not intended and you are
   restoring into a new database, supply the desired encryption options to the restore API. If this is a pre-
   existing database, the encryption options can not be changed and you must either use a different backup
image,
   restore into a a different database, or drop the existing database and then re-issue the restore with the
   desired encryption options.
```

- **Key Rotations are also found in the db2diag.log**

```
   Key Rotation successful using label:
DATA #2 : String, 46 bytes
DB2_SYSGEN_db2inst1_SECRET_2015-02-09-05.03.12
```

The db2diag log is a source of information on errors that might occur when using encryption. Aside from figuring out what errors you have with the encryption, you can also get information on key rotations that took place within the database.



DB2 Native Encryption vs. Guardium Data Encryption

- **Database managed encryption**
  - Management by DBA/Application Teams
  - Cloud database encryption
  - Encrypts databases and backup images, and optionally compresses backups
- **Simplified DBA deployment**
- **Comparable option to SAP HANA/Oracle/MS SQL**

- **Enterprise managed encryption**
  - Heterogeneous Database, Application and Filesystem Encryption
- **Managed by Enterprise Security Team**

IBM DB2 Encryption Option

Native Encryption on all platforms for BLU, pureScale, DPF and Single Instance DB2

Only option for pureScale on Linux

Decision based on who is the buyer and Manager of the system/project

InfoSphere Guardium Data Encryption centralizes encryption

Supports pureScale on AIX only

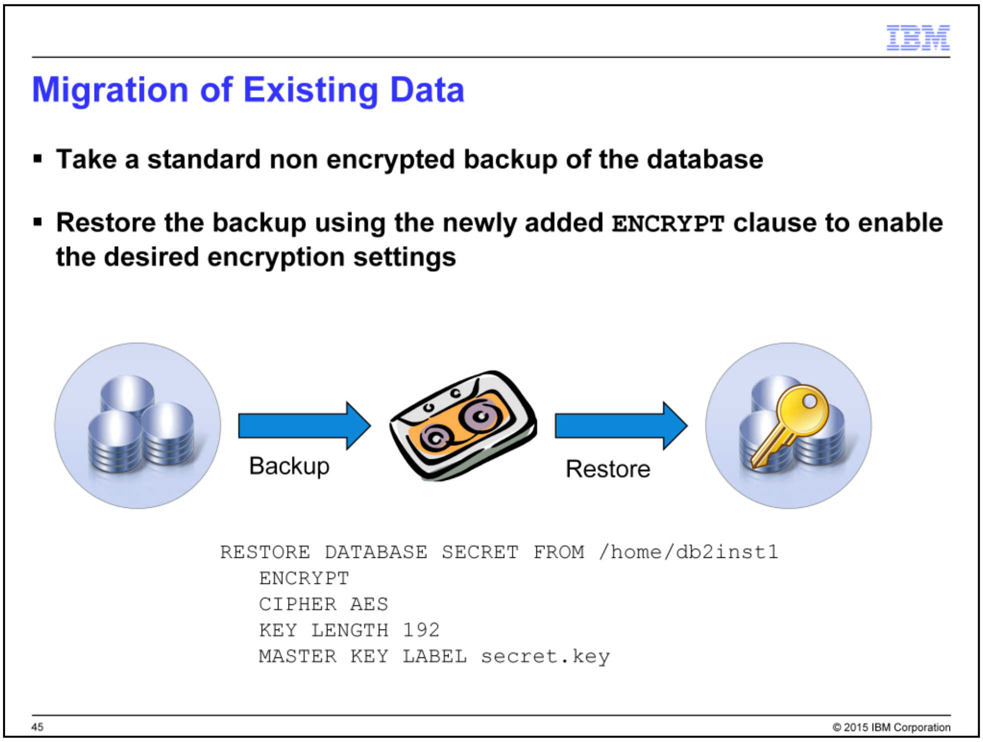
44

In addition to the DB2 Native Encryption option, customers can also purchase the InfoSphere Guardium Data Encryption product. This product encompasses more than just the database files as illustrated in the following diagram.

InfoSphere Guardium Data Encryption appeals to Enterprise Security teams who want to manage heterogeneous databases along with application and file system files.

The DB2 Native Encryption option is targeted at DBA and Application teams who want to encrypt databases and backup images. The technology is easy to implement and can be deployed at the application layer, rather than requiring everything to be encrypted. This technology is also well suited for Cloud deployments.

From a technology perspective, InfoSphere Guardium supports all DB2 platforms, except for the DB2 pureScale feature, which can only be encrypted on the AIX platform. The DB2 Native Encryption supports pureScale on Linux as well as AIX. In addition, the IBM DB2 Encryption option supports encryption and compression on backup, while the IGDE product does not support compression on an encrypted backup.



Migration of Existing Data

- Take a standard non encrypted backup of the database
- Restore the backup using the newly added `ENCRYPT` clause to enable the desired encryption settings

Backup
Restore

```
RESTORE DATABASE SECRET FROM /home/db2inst1
   ENCRYPT
   CIPHER AES
   KEY LENGTH 192
   MASTER KEY LABEL secret.key
```

45

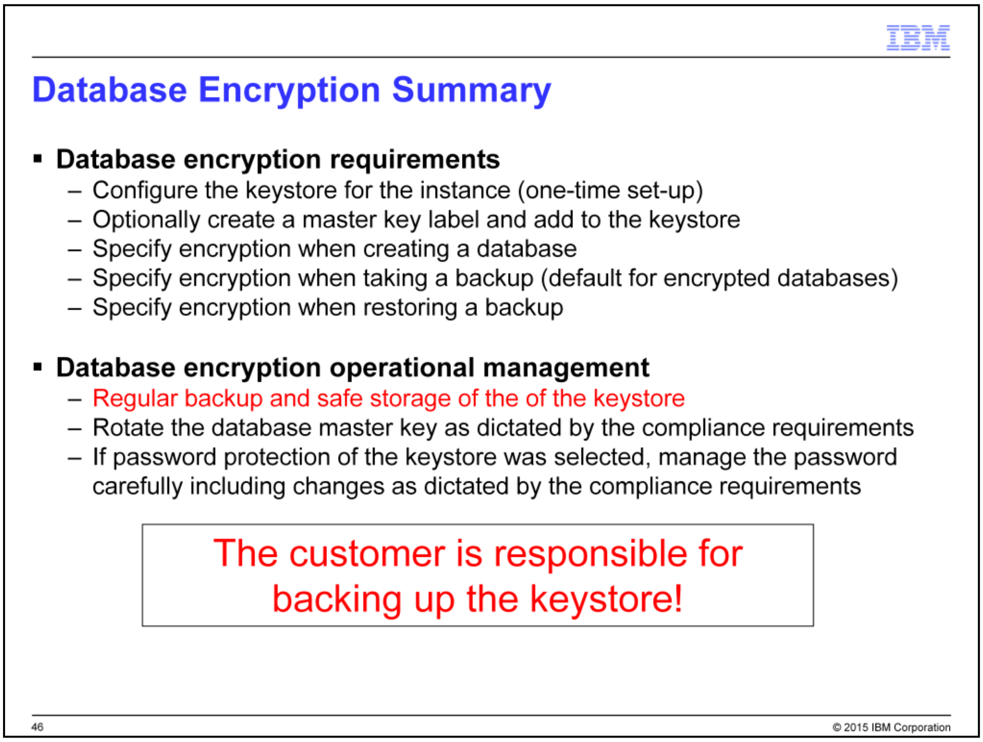
© 2015 IBM Corporation

It is possible to convert an existing unencrypted database into an encrypted database. The approach is as follows:

•First, you take a backup of your existing database using the BACKUP DATABASE command.

•Then, restore that backup image into a new database using the RESTORE DATABASE command. When invoking the RESTORE DATABASE command, you specify the new ENCRYPT option. This new option mirrors exactly the ENCRYPT option of the CREATE DATABASE command. That is, the default is that your new database will be encrypted using AES 256, but you can choose different algorithms and key sizes if so desired.

There is no utility that will allow you to encrypt a database in place. For this reason, a customer will have to do some planning to allow for a sufficient amount of time to do a full database restore.



Database Encryption Summary

- **Database encryption requirements**
  - Configure the keystore for the instance (one-time set-up)
  - Optionally create a master key label and add to the keystore
  - Specify encryption when creating a database
  - Specify encryption when taking a backup (default for encrypted databases)
  - Specify encryption when restoring a backup
- **Database encryption operational management**
  - Regular backup and safe storage of the of the keystore
  - Rotate the database master key as dictated by the compliance requirements
  - If password protection of the keystore was selected, manage the password carefully including changes as dictated by the compliance requirements

The customer is responsible for backing up the keystore!

This slide explains the steps that are required to implement encryption and the responsibilities that a customer has to manage their keystore database.

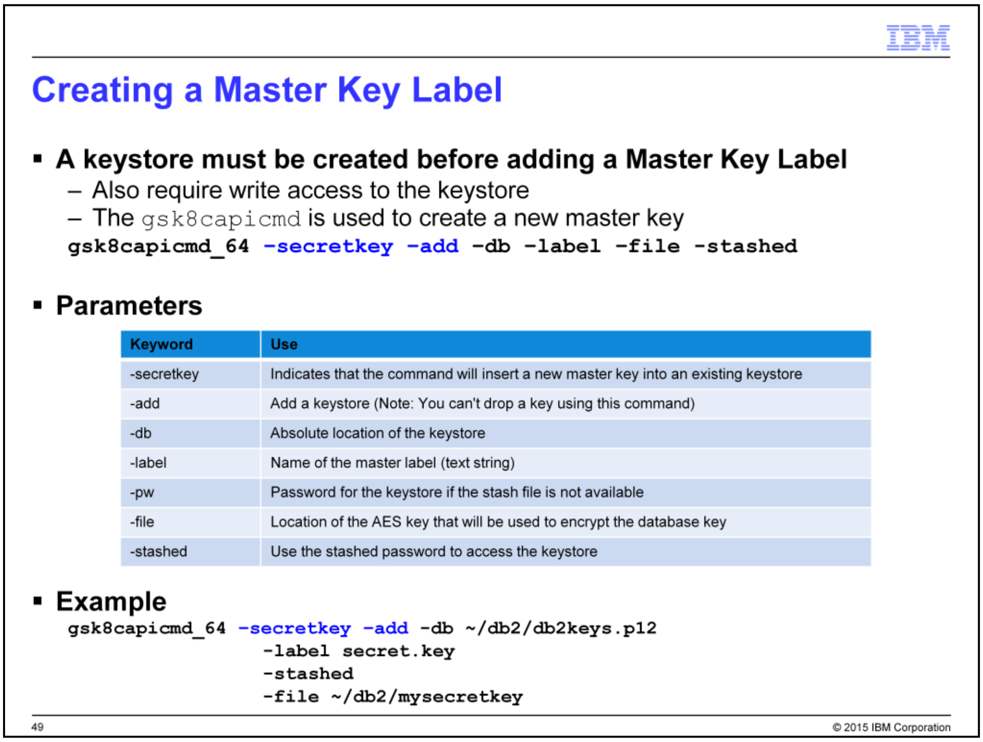
The most important point is that the customer is responsible for backing up the keystore! If you lose the keystore you loose access to your databases!

IBM DB2 Encryption Offering
Additional Slides

Thank-you!

Master Key Management Additional Slides

IBM DB2 Encryption Offering is the official name of the encryption product. This section will discuss the product and how a customer can implement encryption at a database level.



Creating a Master Key Label

- **A keystore must be created before adding a Master Key Label**
  - Also require write access to the keystore
  - The `gsk8capicmd` is used to create a new master key

```
gsk8capicmd_64 -secretkey -add -db -label -file -stashed
```

- **Parameters**

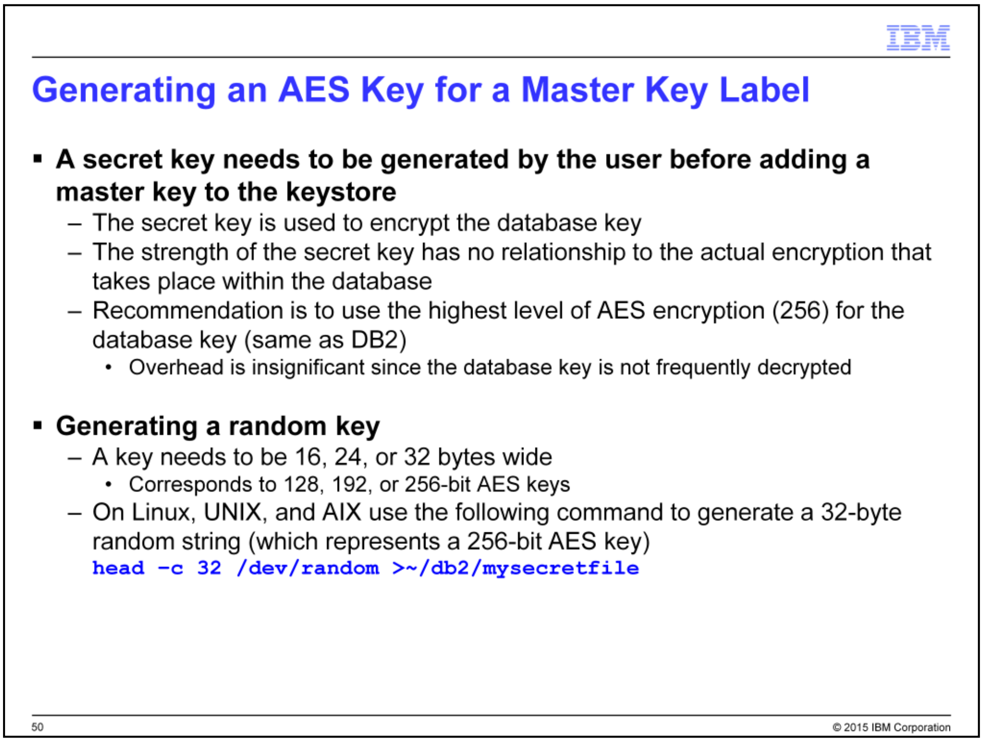
Keyword	Use
-secretkey	Indicates that the command will insert a new master key into an existing keystore
-add	Add a keystore (Note: You can't drop a key using this command)
-db	Absolute location of the keystore
-label	Name of the master label (text string)
-pw	Password for the keystore if the stash file is not available
-file	Location of the AES key that will be used to encrypt the database key
-stashed	Use the stashed password to access the keystore

- **Example**

```
gsk8capicmd_64 -secretkey -add -db ~/db2/db2keys.p12
                -label secret.key
                -stashed
                -file ~/db2/mysecretkey
```

Creating a master key label involves the use of the gsk8capicmd_64 and the –secretkey –add option. The command needs to know the location of the keystore for the database, the password (if it is not stashed), the name (label) of your secret key, and a file that contains the secret key.

The next slide explains the contents of the file.



Generating an AES Key for a Master Key Label

- **A secret key needs to be generated by the user before adding a master key to the keystore**
  - The secret key is used to encrypt the database key
  - The strength of the secret key has no relationship to the actual encryption that takes place within the database
  - Recommendation is to use the highest level of AES encryption (256) for the database key (same as DB2)
    - Overhead is insignificant since the database key is not frequently decrypted
- **Generating a random key**
  - A key needs to be 16, 24, or 32 bytes wide
    - Corresponds to 128, 192, or 256-bit AES keys
  - On Linux, UNIX, and AIX use the following command to generate a 32-byte random string (which represents a 256-bit AES key)

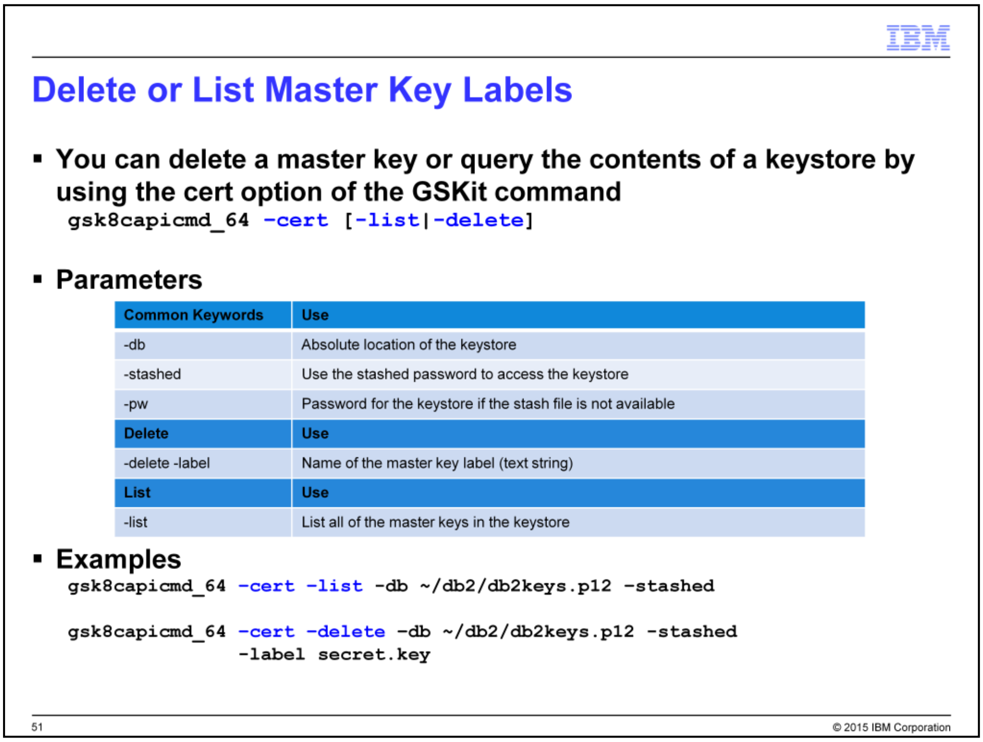
```
head -c 32 /dev/random >~/db2/mysecretfile
```

50

The secret key that you generate for your master key label must be either 16, 24, or 32 bytes long. This key is used to encrypt the database key (which subsequently encrypts the actual database contents).

You could edit a file directly an place exactly 32 characters in it (16=128 bits, 24=192 bits, 32=256 bits). Rather than relying on readable characters, the head –c 32 /dev/random command in Linux could be used to generate a random string of characters to use as input for the key.

The key file that you generate is then used with the –secretkey –add option to be placed into the keystore.



Delete or List Master Key Labels

- **You can delete a master key or query the contents of a keystore by using the cert option of the GSKit command**

```
gsk8capicmd_64 -cert [-list|-delete]
```

- **Parameters**

Common Keywords	Use
-db	Absolute location of the keystore
-stashed	Use the stashed password to access the keystore
-pw	Password for the keystore if the stash file is not available
Delete	**Use**
-delete -label	Name of the master key label (text string)
List	**Use**
-list	List all of the master keys in the keystore

- **Examples**

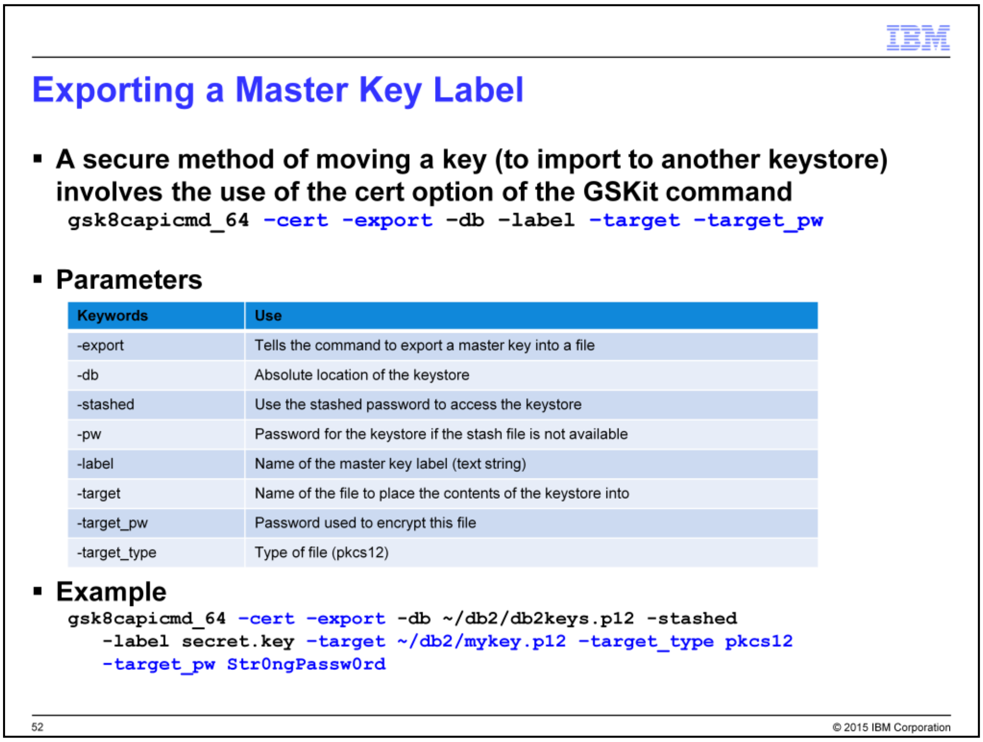
```
gsk8capicmd_64 -cert -list -db ~/db2/db2keys.p12 -stashed

gsk8capicmd_64 -cert -delete -db ~/db2/db2keys.p12 -stashed
               -label secret.key
```

51

While not recommended, you could delete a master key label by using the –cert –delete option of the gsk8capicmd_64 command. One of the reasons you want to keep master keys around is that you may have done a backup some time in the past, and unless you have kept the specific master key for that backup, you will not be able to restore it.

The –list command is used to extract the names of the certificates (master key labels) that are within the keystore.

Note that the user must have access to the stash file or the password for the keystore in order to issue these commands.



Exporting a Master Key Label

- A secure method of moving a key (to import to another keystore) involves the use of the cert option of the GSKit command

```
gsk8capicmd_64 -cert -export -db -label -target -target_pw
```

- Parameters

Keywords	Use
-export	Tells the command to export a master key into a file
-db	Absolute location of the keystore
-stashed	Use the stashed password to access the keystore
-pw	Password for the keystore if the stash file is not available
-label	Name of the master key label (text string)
-target	Name of the file to place the contents of the keystore into
-target_pw	Password used to encrypt this file
-target_type	Type of file (pkcs12)

- Example

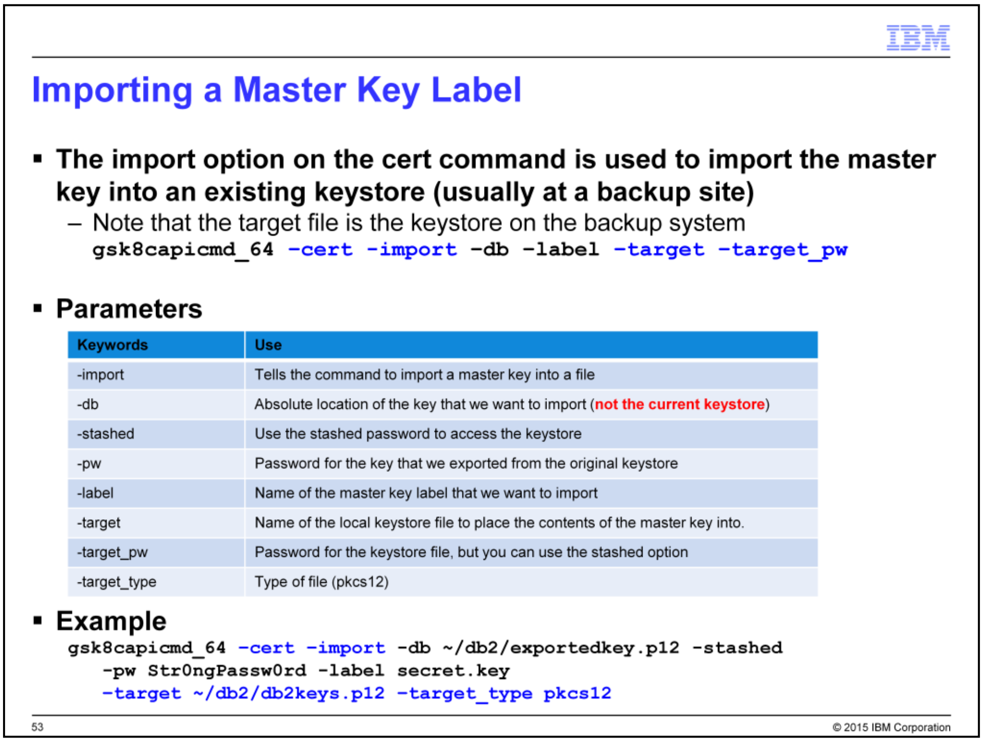
```
gsk8capicmd_64 -cert -export -db ~/db2/db2keys.p12 -stashed
   -label secret.key -target ~/db2/mykey.p12 -target_type pkcs12
   -target_pw Str0ngPassw0rd
```

There are situations where you may want to export the master key label to another system. For instance, the backups that are generated for an encrypted database are (generally speaking) encrypted as well. If you want to restore this database on a different server, then that server will need access to the master key label.

There are two methods available to move a master key. One method is to use the –extract command which takes the contents of the current encryption key and places it into a file. This file can then be moved to a second system where you would use the –secretkey –add instructions to place it into the local keystore.

One of the drawbacks of using this approach is that the key file that is generated, is not encrypted. If this file were intercepted at any point, you've got a serious security exposure.

The –export option is a more secure way of transporting a master key. The –export takes the master key that you specify and places it into a file that is encrypted, using a password that you supply. This file can be moved to the backup system and then added to the local keystore. In this case, if the file were to be lost during "transit", the key is not exposed because it is encrypted with a password.



Importing a Master Key Label

- **The import option on the cert command is used to import the master key into an existing keystore (usually at a backup site)**
  – Note that the target file is the keystore on the backup system

```
gsk8capicmd_64 -cert -import -db -label -target -target_pw
```

- **Parameters**

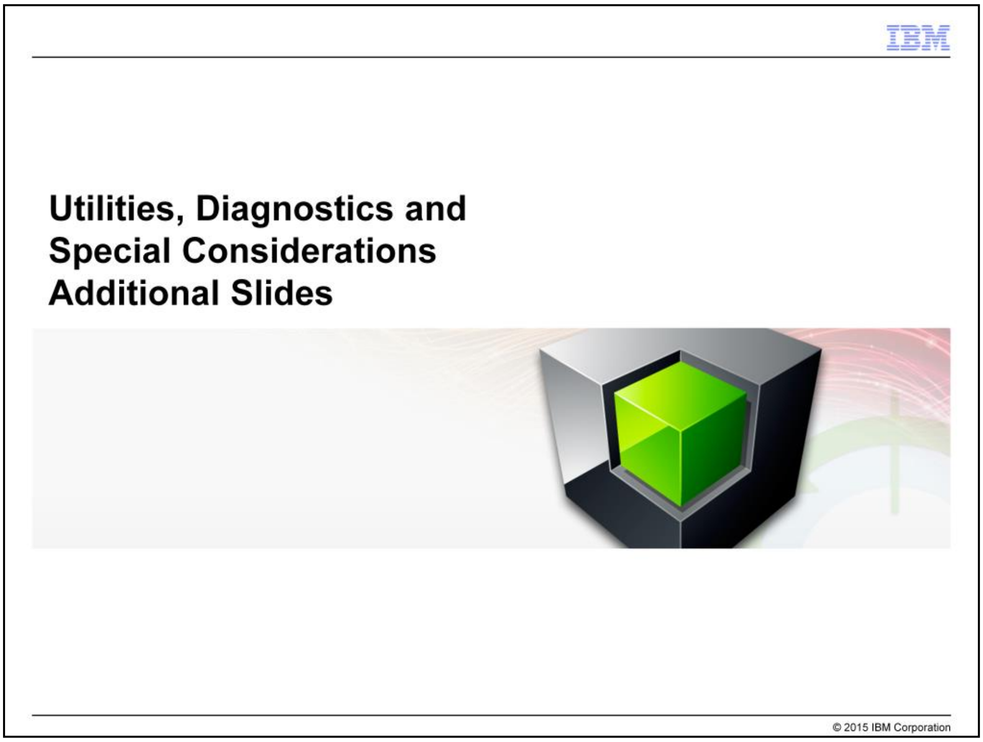
Keywords	Use
-import	Tells the command to import a master key into a file
-db	Absolute location of the key that we want to import (**not the current keystore**)
-stashed	Use the stashed password to access the keystore
-pw	Password for the key that we exported from the original keystore
-label	Name of the master key label that we want to import
-target	Name of the local keystore file to place the contents of the master key into.
-target_pw	Password for the keystore file, but you can use the stashed option
-target_type	Type of file (pkcs12)

- **Example**

```
gsk8capicmd_64 -cert -import -db ~/db2/exportedkey.p12 -stashed
   -pw Str0ngPassw0rd -label secret.key
   -target ~/db2/db2keys.p12 -target_type pkcs12
```

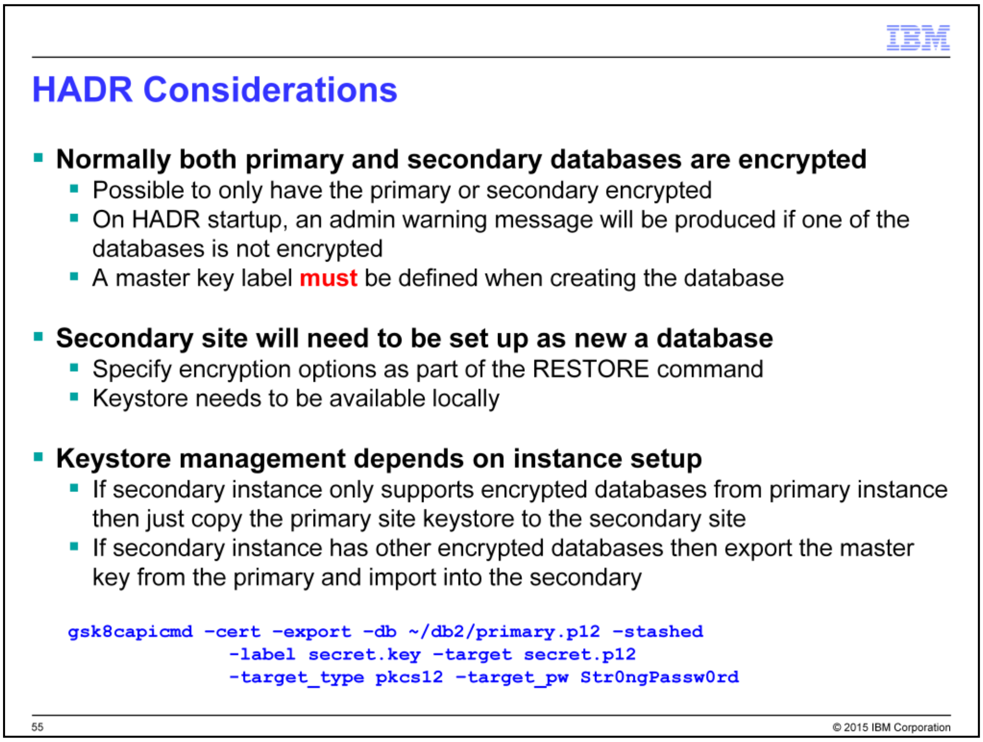
Once a master key label file has been moved to another system, you can use the –cert –import option to place it into the local keystore. You must have the password that was used to encrypt the key in order for it to be decrypted. The keystore that the key is being loaded in to will also require a password. In this case, you can use the –stashed option so that the command takes the password from a stash file. If the stash file does not exist then you will need to supply the password for the local keystore as well.

Note that the –db option refers to the master key label file that you generated on the primary system, and -target refers to the local keystore.



Utilities, Diagnostics and Special Considerations Additional Slides

This section discusses some of the miscellaneous items associated with encryption.



HADR Considerations

- **Normally both primary and secondary databases are encrypted**
  - Possible to only have the primary or secondary encrypted
  - On HADR startup, an admin warning message will be produced if one of the databases is not encrypted
  - A master key label **must** be defined when creating the database
- **Secondary site will need to be set up as new a database**
  - Specify encryption options as part of the RESTORE command
  - Keystore needs to be available locally
- **Keystore management depends on instance setup**
  - If secondary instance only supports encrypted databases from primary instance then just copy the primary site keystore to the secondary site
  - If secondary instance has other encrypted databases then export the master key from the primary and import into the secondary

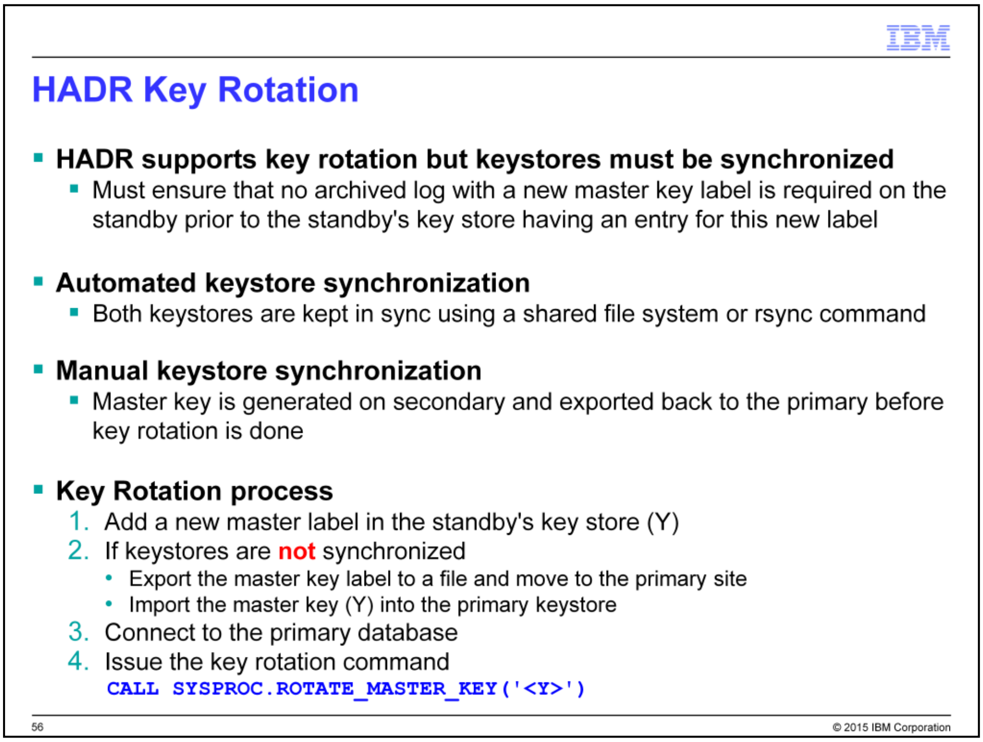
```
gsk8capicmd -cert -export -db ~/db2/primary.p12 -stashed
              -label secret.key -target secret.p12
              -target_type pkcs12 -target_pw Str0ngPassw0rd
```

HADR is supported with encryption. This slide illustrates the steps required in order to have encryption working on both sites.

One of the interesting things about encryption and HADR is that one or both of the sites can be encrypted. You can choose to have the secondary site encrypted and not the primary (or vice versa). This is fully supported, but a warning message will be issued to remind you that they are out of sync.

The same master key must be available on both sites. The best way to do this is to have the same keystore on both sites. If there is an existing keystore at the secondary site, then you should export the key from the primary and import it into the secondary site.

Keystores on primary and secondary can be kept in sync automatically by using some file level synchronization mechanism such as rsync or similar or by physically sharing the keystore. If doing the master key synchronization manually, just add keys to both keystores but make sure it is done to the standby first.



HADR Key Rotation

- **HADR supports key rotation but keystores must be synchronized**
  - Must ensure that no archived log with a new master key label is required on the standby prior to the standby's key store having an entry for this new label
- **Automated keystore synchronization**
  - Both keystores are kept in sync using a shared file system or rsync command
- **Manual keystore synchronization**
  - Master key is generated on secondary and exported back to the primary before key rotation is done
- **Key Rotation process**
  1. Add a new master label in the standby's key store (Y)
  2. If keystores are **not** synchronized
     - Export the master key label to a file and move to the primary site
     - Import the master key (Y) into the primary keystore
  3. Connect to the primary database
  4. Issue the key rotation command

```
CALL SYSPROC.ROTATE_MASTER_KEY('<Y>')
```

56

© 2015 IBM Corporation

The same master key must be available on both sites. The best way to do this is to have the same keystore on both sites. If there is an existing keystore at the secondary site, then you should export the key from the primary and import it into the secondary site.

HADR also supports key rotation but you must supply a master key label for the key rotation command. Best practice is to:

STANDBY:

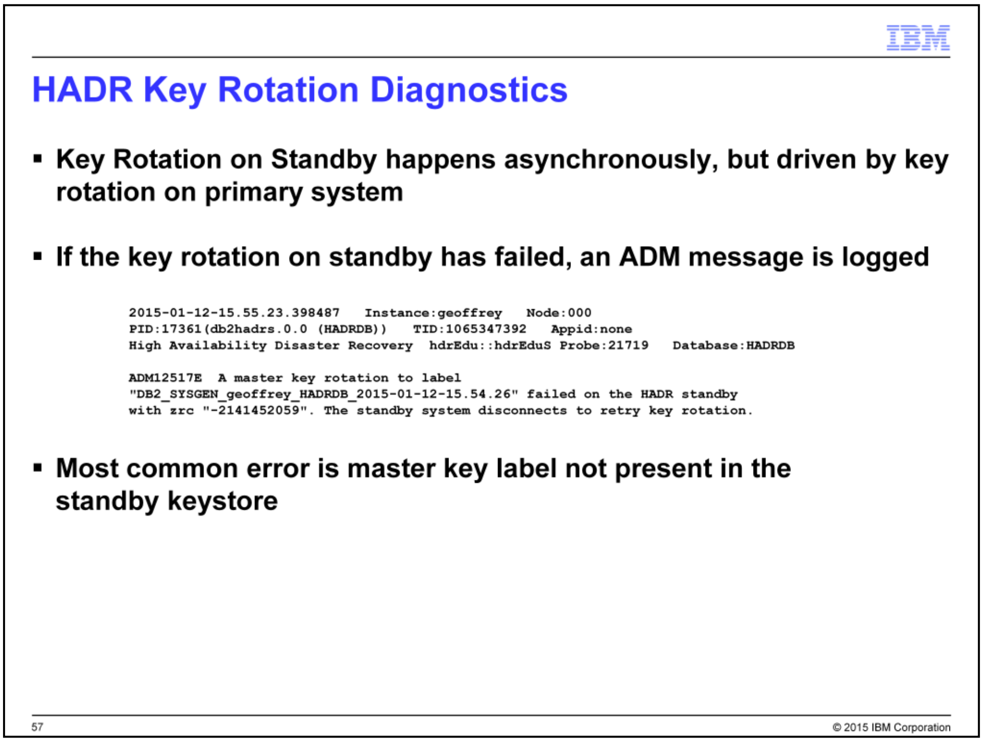
•Add new master key label <Y> to the key store on the standby database. Extract the master key and move it to the primary site.

PRIMARY:

•Import the master key (same one as used on secondary) into the keystore

•CALL SYSPROC.ROTATE_MASTER_KEY('<Y>');

Keystores on primary and secondary can be kept in sync automatically by using some file level synchronization mechanism such as rsync or similar or by physically sharing the keystore. If doing the master key synchronization manually, just add keys to both keystores but make sure it is done to the standby first.



HADR Key Rotation Diagnostics

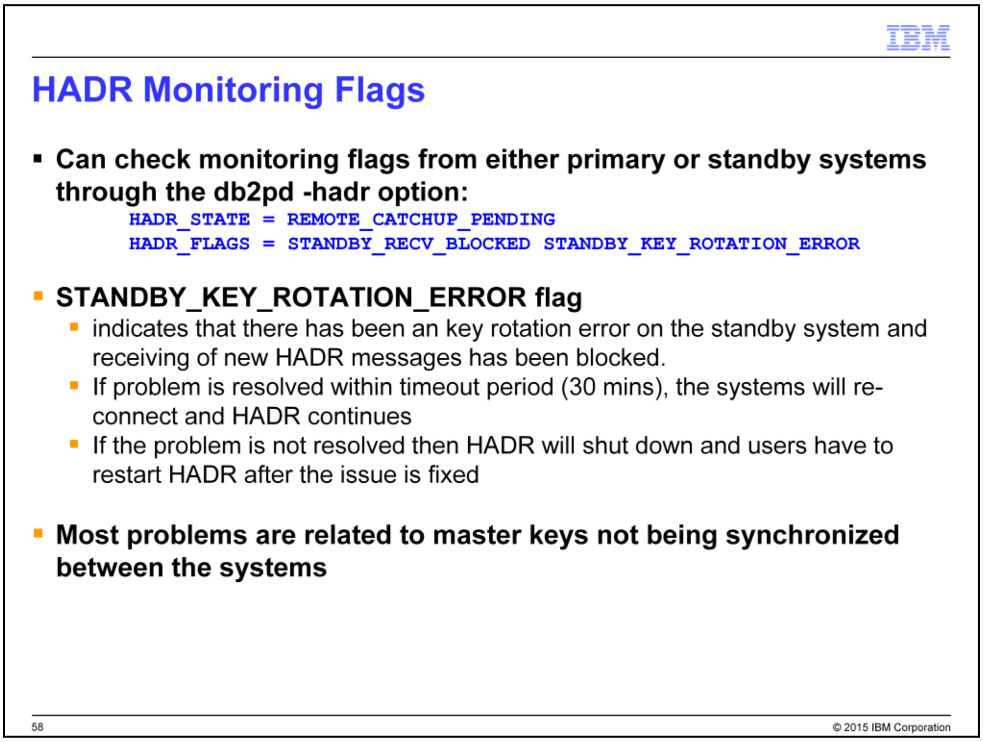
- **Key Rotation on Standby happens asynchronously, but driven by key rotation on primary system**
- **If the key rotation on standby has failed, an ADM message is logged**

```
2015-01-12-15.55.23.398487   Instance:geoffrey   Node:000
PID:17361(db2hadrs.0.0 (HADRDB))   TID:1065347392   Appid:none
High Availability Disaster Recovery  hdrEdu::hdrEduS Probe:21719   Database:HADRDB

ADM12517E  A master key rotation to label
"DB2_SYSGEN_geoffrey_HADRDB_2015-01-12-15.54.26" failed on the HADR standby
with zrc "-2141452059". The standby system disconnects to retry key rotation.
```

- **Most common error is master key label not present in the standby keystore**

57

HADR key rotation is logged in the db2diag file, along with any errors that are caused during rotation. View these messages to see cause of key rotation failure. Most common is master key label not present in standby keystore.



HADR Monitoring Flags

- Can check monitoring flags from either primary or standby systems through the db2pd -hadr option:

```
HADR_STATE = REMOTE_CATCHUP_PENDING
HADR_FLAGS = STANDBY_RECV_BLOCKED STANDBY_KEY_ROTATION_ERROR
```

- **STANDBY_KEY_ROTATION_ERROR flag**
  - indicates that there has been an key rotation error on the standby system and receiving of new HADR messages has been blocked.
  - If problem is resolved within timeout period (30 mins), the systems will re-connect and HADR continues
  - If the problem is not resolved then HADR will shut down and users have to restart HADR after the issue is fixed
- **Most problems are related to master keys not being synchronized between the systems**

58

Most problems are related to master key issues. The following scenarios illustrate the potential problems that can occur.

Customer creates primary DB using automatically generated master key label.

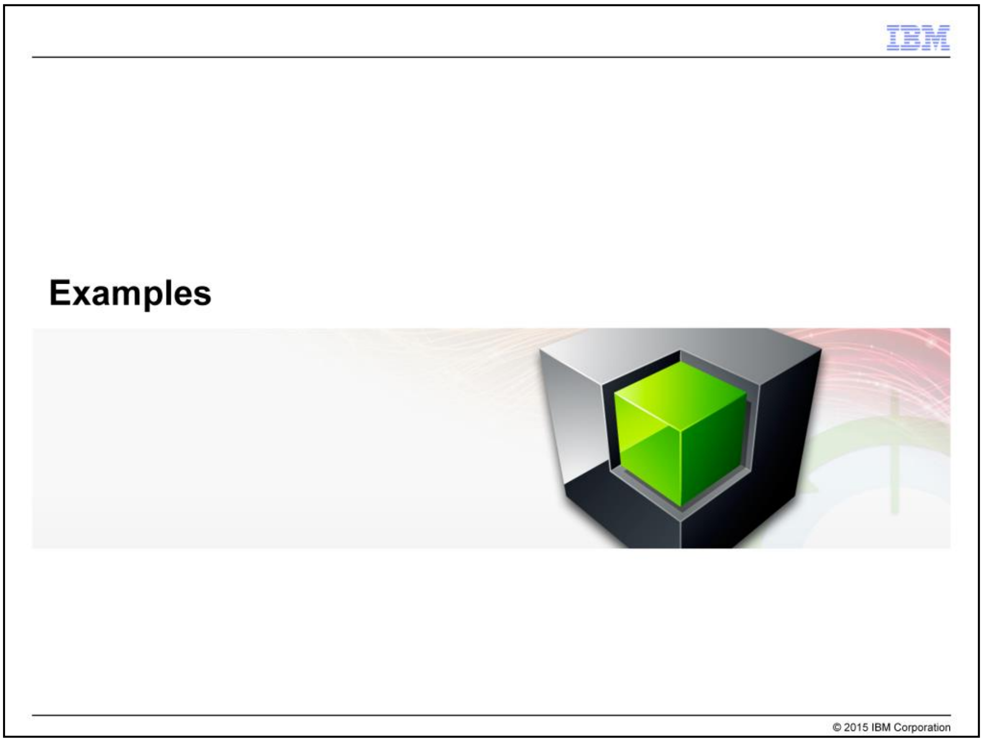
- Determine label using the built-in table function
- Extract master key from primary system's keystore
- Import master key into standby system's keystore
- Use this label when restoring backup to create standby database

Customer creates standby DB using automatically generated master key label

The standby will immediately drive key rotation as it detects primary is using a different label.

This will fail as the label does not exist on the standby site (hadr will retry several times but after bring down the system).

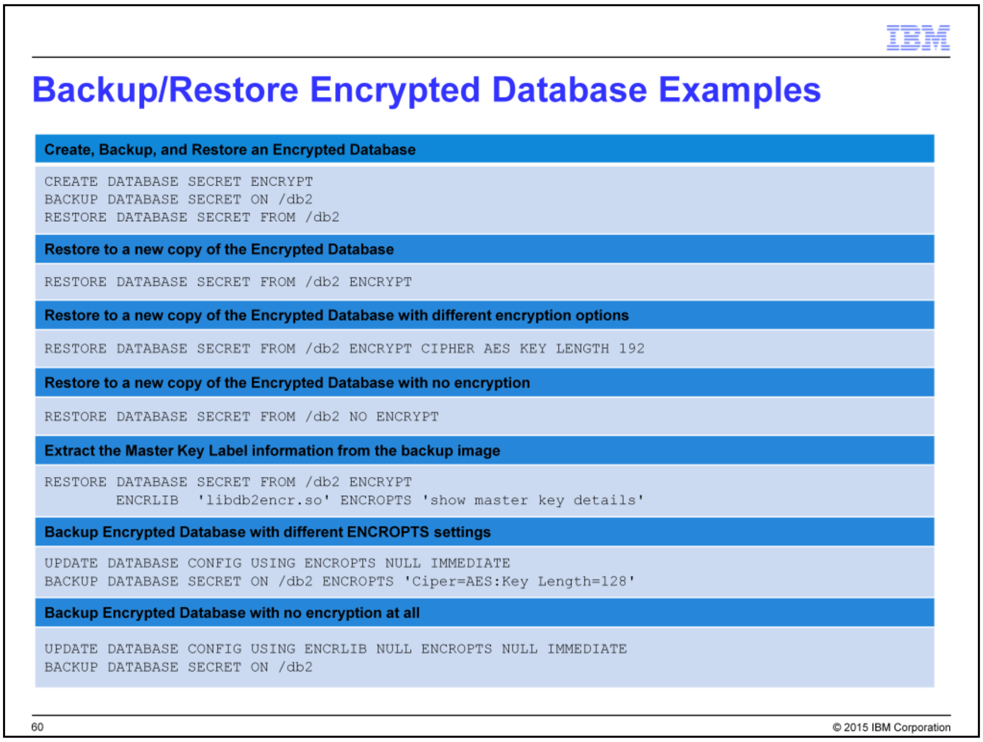
Add label and restart.



Examples

© 2015 IBM Corporation

This section discusses some of the miscellaneous items associated with encryption.



Backup/Restore Encrypted Database Examples

Create, Backup, and Restore an Encrypted Database

```
CREATE DATABASE SECRET ENCRYPT
BACKUP DATABASE SECRET ON /db2
RESTORE DATABASE SECRET FROM /db2
```

Restore to a new copy of the Encrypted Database

```
RESTORE DATABASE SECRET FROM /db2 ENCRYPT
```

Restore to a new copy of the Encrypted Database with different encryption options

```
RESTORE DATABASE SECRET FROM /db2 ENCRYPT CIPHER AES KEY LENGTH 192
```

Restore to a new copy of the Encrypted Database with no encryption

```
RESTORE DATABASE SECRET FROM /db2 NO ENCRYPT
```

Extract the Master Key Label information from the backup image

```
RESTORE DATABASE SECRET FROM /db2 ENCRYPT
        ENCRLIB  'libdb2encr.so' ENCROPTS 'show master key details'
```

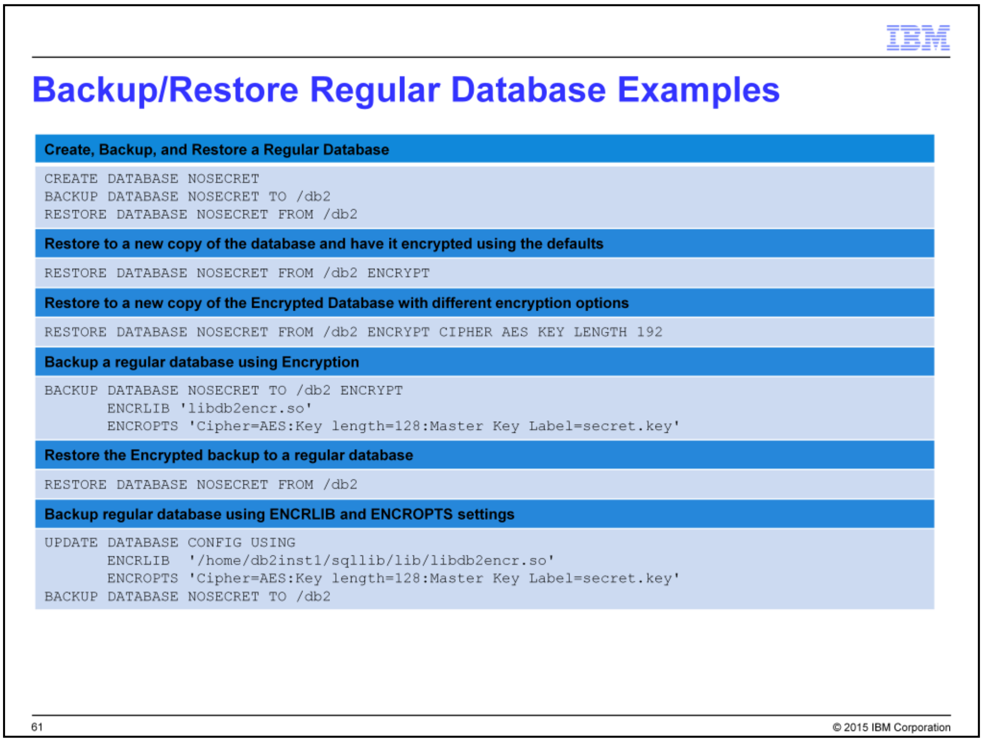
Backup Encrypted Database with different ENCROPTS settings

```
UPDATE DATABASE CONFIG USING ENCROPTS NULL IMMEDIATE
BACKUP DATABASE SECRET ON /db2 ENCROPTS 'Ciper=AES:Key Length=128'
```

Backup Encrypted Database with no encryption at all

```
UPDATE DATABASE CONFIG USING ENCRLIB NULL ENCROPTS NULL IMMEDIATE
BACKUP DATABASE SECRET ON /db2
```

This slide shows some examples of using the backup and restore commands with encrypted databases.



Backup/Restore Regular Database Examples

Create, Backup, and Restore a Regular Database

```
CREATE DATABASE NOSECRET
BACKUP DATABASE NOSECRET TO /db2
RESTORE DATABASE NOSECRET FROM /db2
```

Restore to a new copy of the database and have it encrypted using the defaults

```
RESTORE DATABASE NOSECRET FROM /db2 ENCRYPT
```

Restore to a new copy of the Encrypted Database with different encryption options

```
RESTORE DATABASE NOSECRET FROM /db2 ENCRYPT CIPHER AES KEY LENGTH 192
```

Backup a regular database using Encryption

```
BACKUP DATABASE NOSECRET TO /db2 ENCRYPT
       ENCRLIB 'libdb2encr.so'
       ENCROPTS 'Cipher=AES:Key length=128:Master Key Label=secret.key'
```

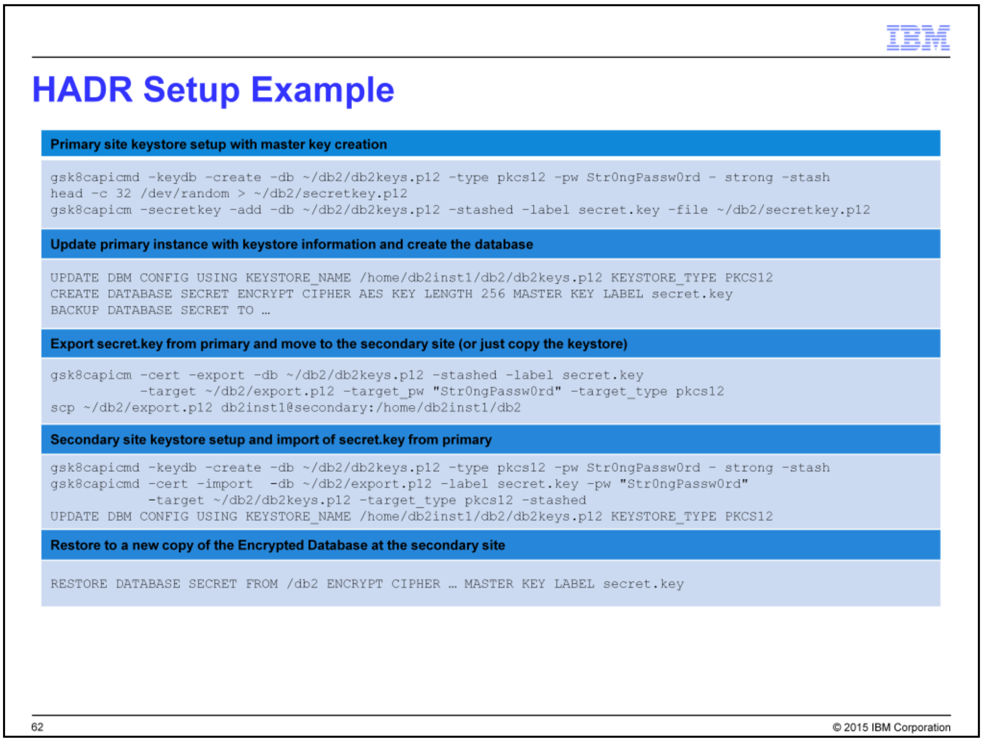
Restore the Encrypted backup to a regular database

```
RESTORE DATABASE NOSECRET FROM /db2
```

Backup regular database using ENCRLIB and ENCROPTS settings

```
UPDATE DATABASE CONFIG USING
       ENCRLIB  '/home/db2inst1/sqllib/lib/libdb2encr.so'
       ENCROPTS 'Cipher=AES:Key length=128:Master Key Label=secret.key'
BACKUP DATABASE NOSECRET TO /db2
```

This slide shows some examples of using the backup and restore commands with normal databases.



HADR Setup Example

Primary site keystore setup with master key creation

```
gsk8capicmd -keydb -create -db ~/db2/db2keys.p12 -type pkcs12 -pw Str0ngPassw0rd - strong -stash
head -c 32 /dev/random > ~/db2/secretkey.p12
gsk8capicm -secretkey -add -db ~/db2/db2keys.p12 -stashed -label secret.key -file ~/db2/secretkey.p12
```

Update primary instance with keystore information and create the database

```
UPDATE DBM CONFIG USING KEYSTORE_NAME /home/db2inst1/db2/db2keys.p12 KEYSTORE_TYPE PKCS12
CREATE DATABASE SECRET ENCRYPT CIPHER AES KEY LENGTH 256 MASTER KEY LABEL secret.key
BACKUP DATABASE SECRET TO …
```

Export secret.key from primary and move to the secondary site (or just copy the keystore)

```
gsk8capicm -cert -export -db ~/db2/db2keys.p12 -stashed -label secret.key
           -target ~/db2/export.p12 -target_pw "Str0ngPassw0rd" -target_type pkcs12
scp ~/db2/export.p12 db2inst1@secondary:/home/db2inst1/db2
```

Secondary site keystore setup and import of secret.key from primary

```
gsk8capicmd -keydb -create -db ~/db2/db2keys.p12 -type pkcs12 -pw Str0ngPassw0rd - strong -stash
gsk8capicmd -cert -import  -db ~/db2/export.p12 -label secret.key -pw "Str0ngPassw0rd"
            -target ~/db2/db2keys.p12 -target_type pkcs12 -stashed
UPDATE DBM CONFIG USING KEYSTORE_NAME /home/db2inst1/db2/db2keys.p12 KEYSTORE_TYPE PKCS12
```

Restore to a new copy of the Encrypted Database at the secondary site

```
RESTORE DATABASE SECRET FROM /db2 ENCRYPT CIPHER … MASTER KEY LABEL secret.key
```

62

HADR is supported with encryption. This slide illustrates the steps required in order to have encryption working on both sites.

One of the interesting things about encryption and HADR is that one or both of the sites can be encrypted. You can choose to have the secondary site encrypted and not the primary (or vice versa). This is fully supported, but a warning message will be issued to remind you that they are out of sync.

The same master key must be available on both sites. The best way to do this is to have the same keystore on both sites. If there is an existing keystore at the secondary site, then you should export the key from the primary and import it into the secondary site.

HADR also supports key rotation but you must supply a master key label for the key rotation command. Best practice is to:

STANDBY:

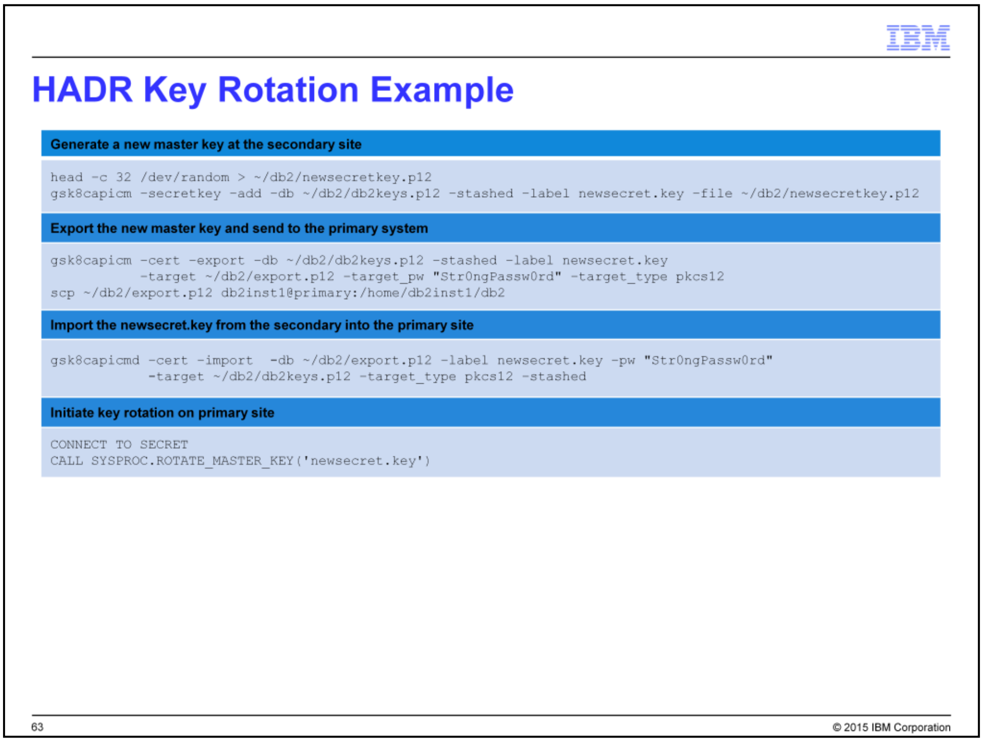
•Add new master key label <Y> to the key store on the standby database. Extract the master key and move it to the primary site.

PRIMARY:

•Import the master key (same one as used on secondary) into the keystore

•CALL SYSPROC.ROTATE_MASTER_KEY('<Y>');

Keystores on primary and secondary can be kept in sync automatically by using some file level synchronization mechanism such as rsync or similar or by physically sharing the keystore. If doing the master key synchronization manually, just add keys to both keystores but make sure it is done to the standby first.



HADR Key Rotation Example

Generate a new master key at the secondary site

```
head -c 32 /dev/random > ~/db2/newsecretkey.p12
gsk8capicm -secretkey -add -db ~/db2/db2keys.p12 -stashed -label newsecret.key -file ~/db2/newsecretkey.p12
```

Export the new master key and send to the primary system

```
gsk8capicm -cert -export -db ~/db2/db2keys.p12 -stashed -label newsecret.key
           -target ~/db2/export.p12 -target_pw "Str0ngPassw0rd" -target_type pkcs12
scp ~/db2/export.p12 db2inst1@primary:/home/db2inst1/db2
```

Import the newsecret.key from the secondary into the primary site

```
gsk8capicmd -cert -import  -db ~/db2/export.p12 -label newsecret.key -pw "Str0ngPassw0rd"
            -target ~/db2/db2keys.p12 -target_type pkcs12 -stashed
```

Initiate key rotation on primary site

```
CONNECT TO SECRET
CALL SYSPROC.ROTATE_MASTER_KEY('newsecret.key')
```

HADR also supports key rotation but you must supply a master key label for the key rotation command. Best practice is to:

STANDBY:

•Add new master key label <Y> to the key store on the standby database. Extract the master key and move it to the primary site.

PRIMARY:

•Import the master key (same one as used on secondary) into the keystore

•CALL SYSPROC.ROTATE_MASTER_KEY('<Y>');

Keystores on primary and secondary can be kept in sync automatically by using some file level synchronization mechanism such as rsync or similar or by physically sharing the keystore. If doing the master key synchronization manually, just add keys to both keystores but make sure it is done to the standby first.