z/TPF V1.1

TPF Users Group Spring 2008

The Evolution of Cryptography

Name: Mark Gambino
Venue: Education Session

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

© 2008 IBM Corporation

Agenda

- **History of different cryptography types**
  - Symmetric key cryptography
  - Digest algorithms
  - Public key cryptography
- **How and when do I use each type of cryptography**
- **History of IBM System z hardware cryptography**
- **History of cryptography on TPF**

© 2008 IBM Corporation

Symmetric Key Cryptography

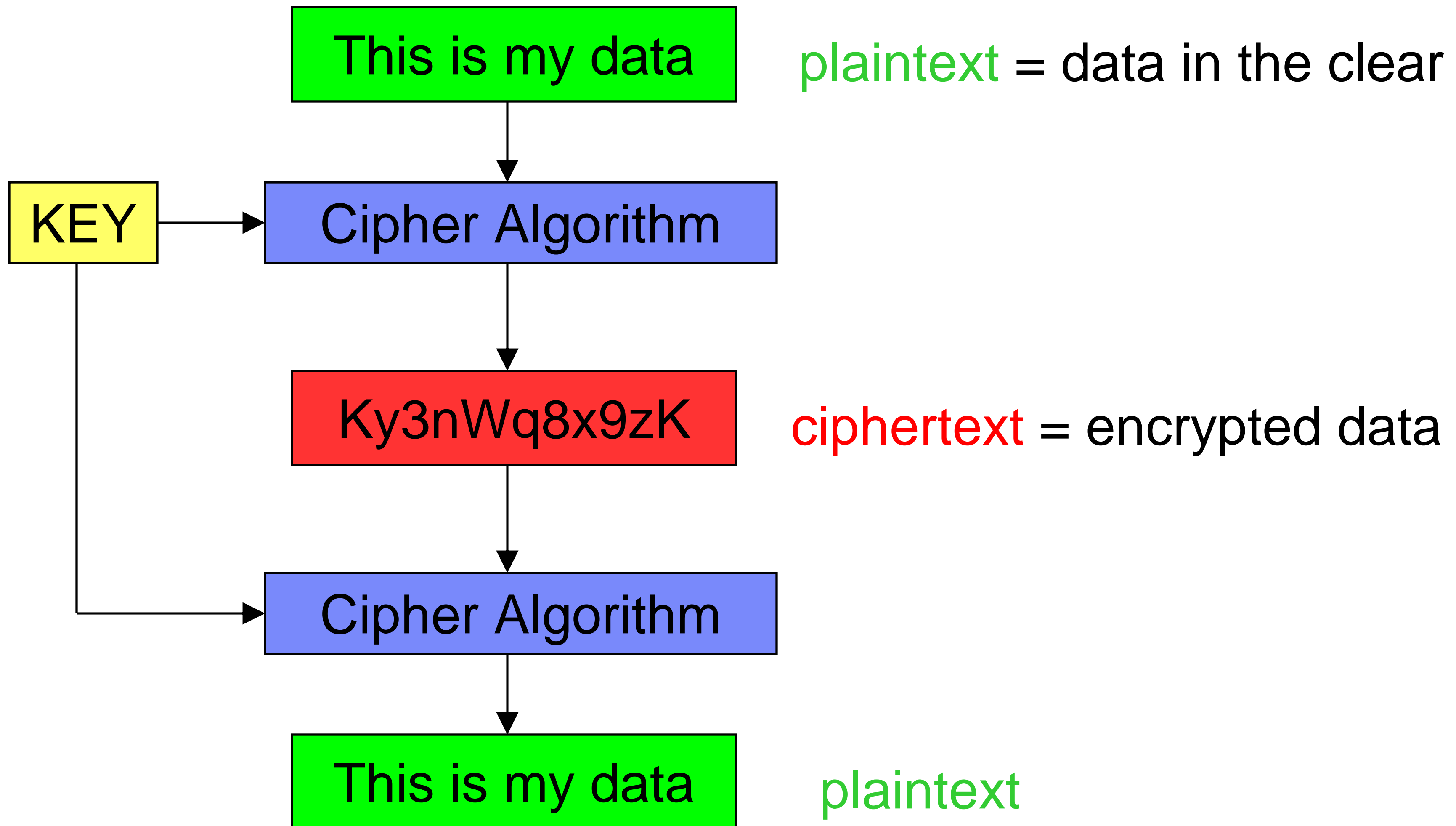
© 2008 IBM Corporation

Symmetric Key Cryptography

- **The same key that encrypts the data is also used to decrypt the data**
- **Stream ciphers**
  - Operate on the data one byte at a time
- **Block ciphers**
  - Operate on the data in groups (blocks) of bytes
  - Cipher Block Chaining (CBC) mode
    - Previous encrypted block of data is used to encrypt the next block
    - More secure mode of operation

© 2008 IBM Corporation

Symmetric Key Cryptography Example



© 2008 IBM Corporation

Data Encryption Standard (DES)

- **Is a symmetric key algorithm**
- **Developed by IBM in the 1970s**
- **Became an approved standard in 1976**
- **Became a Federal Information Processing Standards (FIPS) standard in 1977**
- **Block cipher that operates on 64-bit (8-byte) chunks of data**
- **DES keys are 64 bits**
  - Key strength is only 56 bits because one bit in each of the eight bytes in the key is a parity bit

© 2008 IBM Corporation

DES Attacks

- **First publicly acknowledged way to break DES was in 1997**
  - Researchers without knowledge of the DES key value were able to decrypt a message that was encrypted with that DES key
- **By 1999 the methods to break DES were refined to the point where it could be done in less than 24 hours**
- **Updated FIPS publication in 1999 stated to use Triple-DES and that DES was only allowed to be used by legacy systems from now on**

© 2008 IBM Corporation

Double-DES (DDES) versus Triple-DES (TDES)

- **Double-DES with two different DES keys**
  - Key size is 128 bits (64*2 = 128)
  - You might think the key strength is 112 bits (56*2 = 112)
  - Because of certain plaintext attacks, Double-DES key strength is really only 80 bits
- **Triple-DES with three different DES keys**
  - Key size is 192 bits (64*3 = 192)
  - You might think the key strength is 168 bits (56*3 = 168)
  - Because of “meet in the middle” attacks, Triple-DES key strength is really only 112 bits

© 2008 IBM Corporation

Triple-DES (TDES) Takes Over

- **TDES replaced DES in most cases and is still heavily used in the industry**
- **As encryption usage increased, performance became more and more of an issue**
  - The DES algorithm, and therefore its variations like TDES, consume many CPU resources
  - To use TDES at high volume, crypto hardware acceleration is required on the platform

© 2008 IBM Corporation

Advanced Encryption Standard (AES)

- **Published in 2001**
- **Natural successor to TDES**
- **Block cipher that operates on 128-bit (16-byte) chunks of data**
- **Much more efficient**
  - Many times faster than TDES in software on current processor technology
  - Faster than TDES in most hardware crypto as well

© 2008 IBM Corporation

AES versus TDES Security

- **AES-*nnn* notation means an AES key whose key length is *nnn* bits**
- **AES key length is also its key strength**
  - No parity bits like in DES keys
- **AES is more secure than TDES**
  - TDES key strength is 112 bits
  - AES-128 key strength is 128 bits
  - AES-256 key strength is 256 bits

© 2008 IBM Corporation

Digest Algorithms

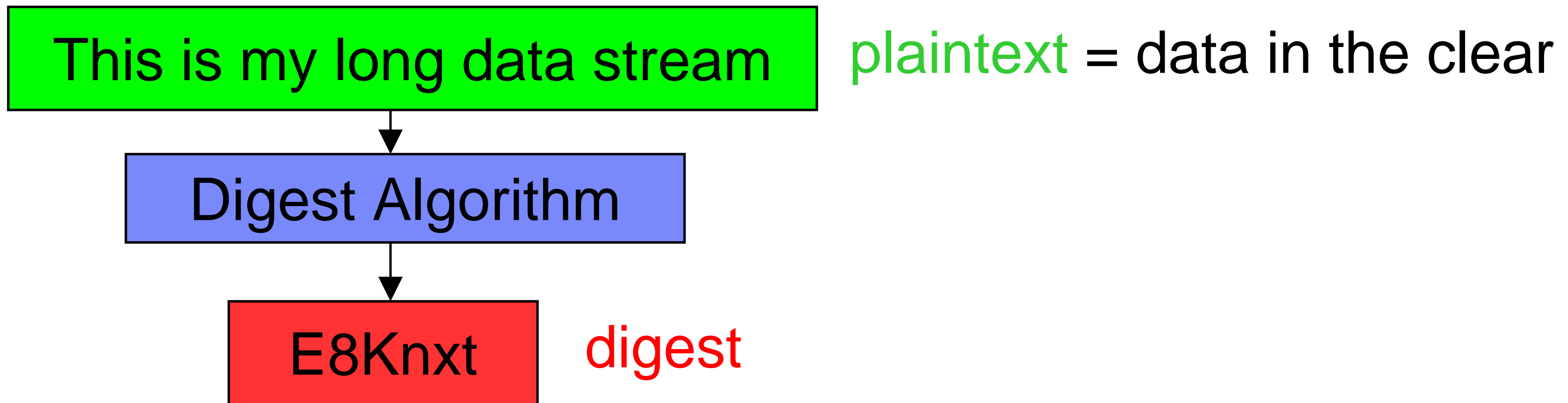
© 2008 IBM Corporation

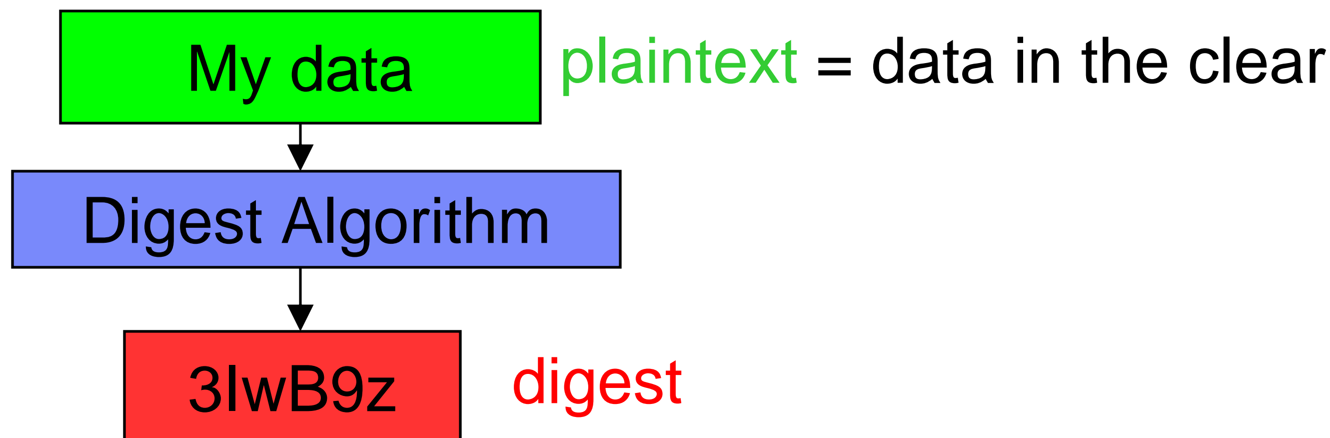
What is a Digest Algorithm?

- **Secure one-way hash algorithm**
- **Input is a pointer to variable length data**
  - There is no key as input to these algorithms
- **Output is a fixed-length digest value**
- **Algorithms are non-reversible**
  - If you have the digest value, you cannot figure out what the original data was

© 2008 IBM Corporation

Digest Cryptography Examples





© 2008 IBM Corporation

Message Digest algorithm 5 (MD5)

- **Developed in 1991**
- **Produces a 128-bit (16-byte) digest value**
- **Widely used for a variety of security purposes, like SSL**
- **Several vulnerabilities with the algorithm have been demonstrated over the years**
  - Including collisions (different input strings producing the same digest value)
  - Not recommended for use anymore in most situations

© 2008 IBM Corporation

Secure Hash Algorithm

- **SHA-1**
  - Published in 1995
  - Produces a 160-bit (20-byte) digest value
  - Widely used for a variety of security purposes
    - SSL, IPSec, S/MIME, PGP
- **SHA-256 and SHA-512**
  - Published in 2001
  - Generically referred to as “SHA-2”
  - SHA-256 produces a 256-bit (32-byte) digest value
  - SHA-512 produces a 512-bit (64-byte) digest value

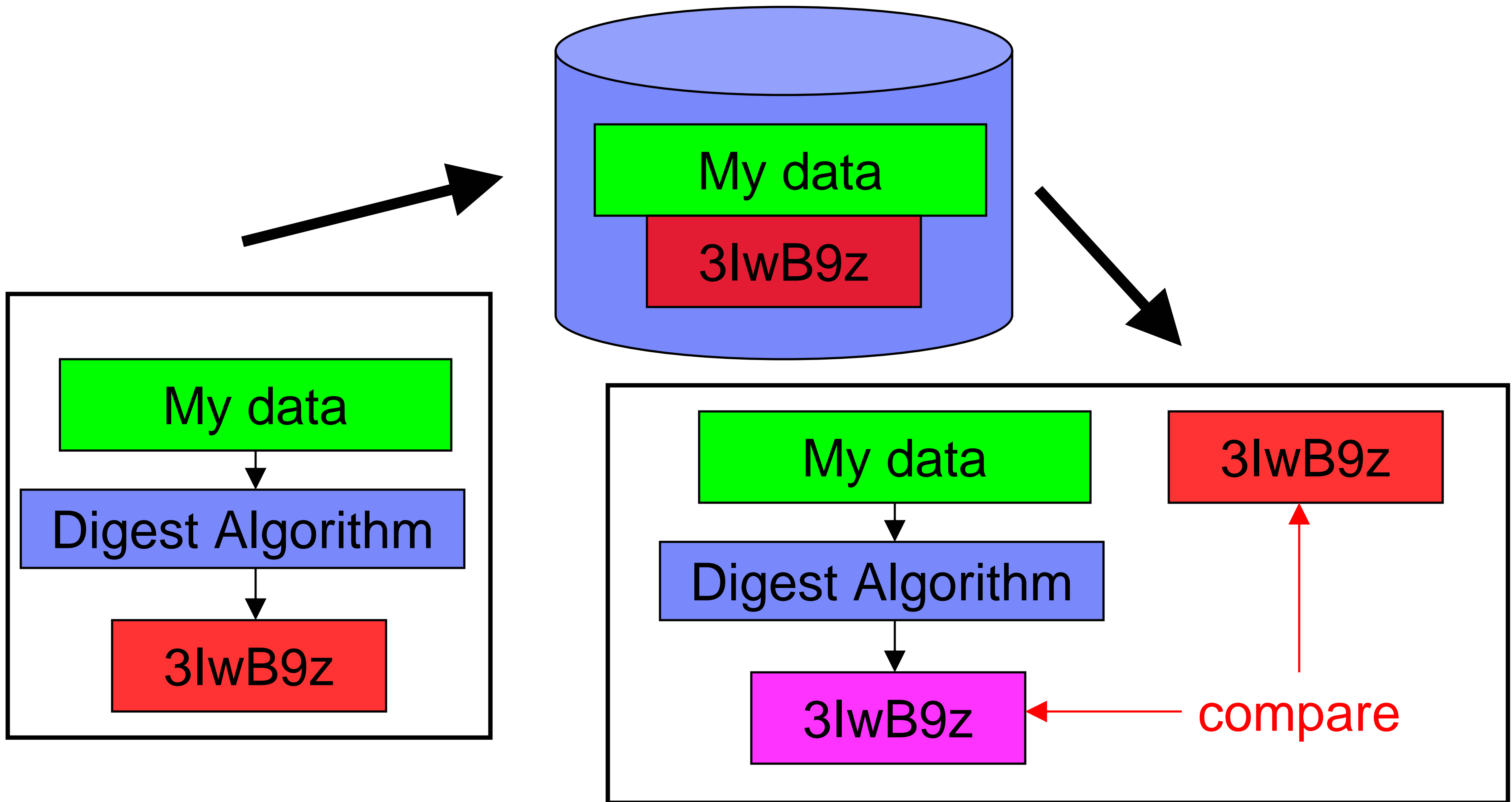
© 2008 IBM Corporation

Use Digests for Data/Message Integrity

- **Send/store a digest of the data along with the data itself**
  - Verify the digest value before using the received/stored data
  - Ability to detect accidental corruption of data
- **Use digest in conjunction with data encryption**
  - Send/store a digest of the plaintext data along with encrypted data
    - To use the received/stored data, decrypt the data and then verify the digest value
    - Ability to detect accidental or malicious corruption of data

© 2008 IBM Corporation

Data Integrity Example



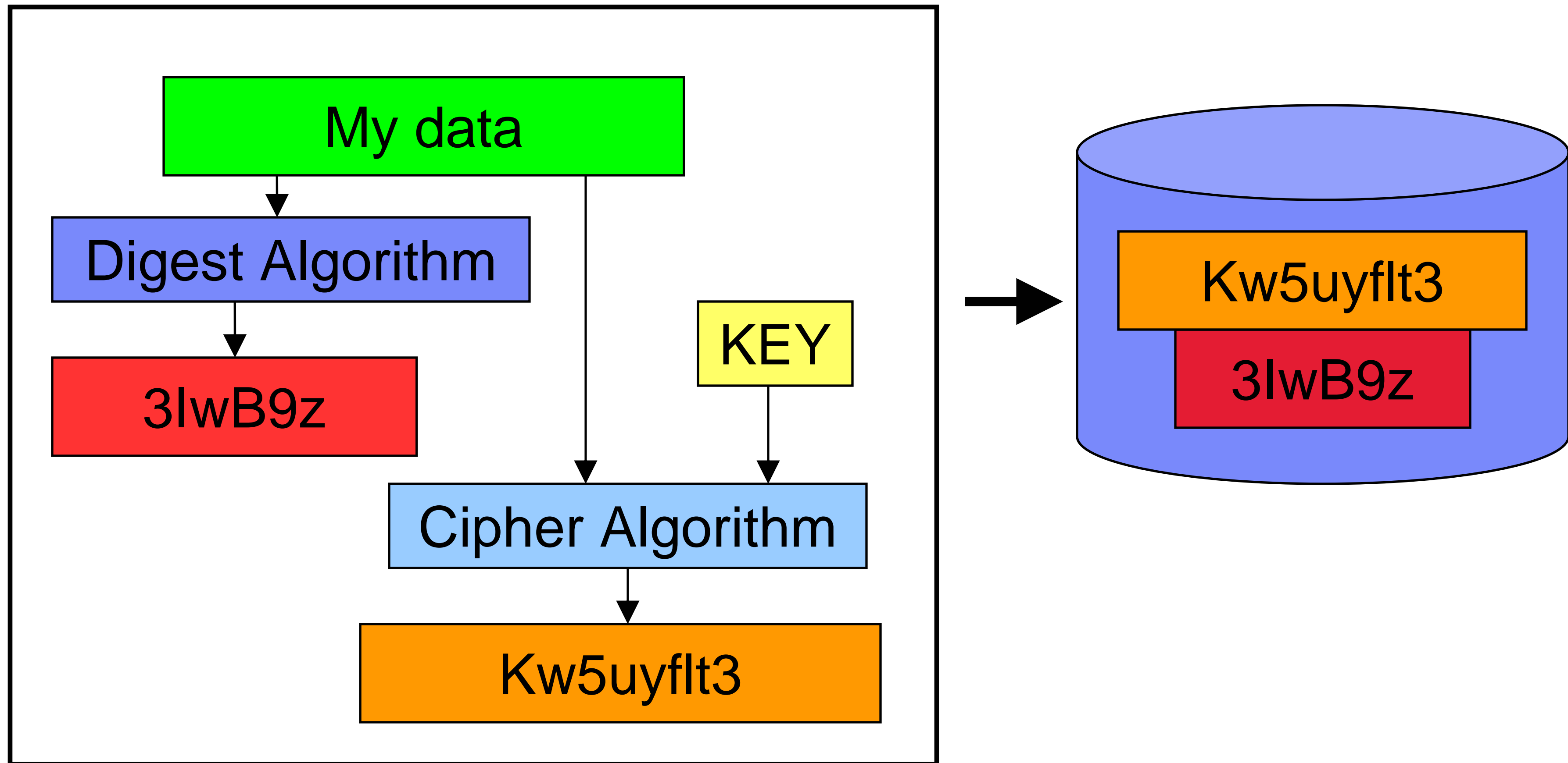
© 2008 IBM Corporation

Data Integrity Example Steps

- **Data creation process**
  1. Create the data
  2. Calculate the digest of the data
  3. Write the data and digest to DASD
- **Data access process**
  1. Read the data and saved digest from DASD
  2. Calculate the digest of the data read from DASD
  3. Compare the digest just calculated to the saved digest value – if they values match, the data is valid

© 2008 IBM Corporation

Data Integrity with Data Encryption Example – Part 1



© 2008 IBM Corporation

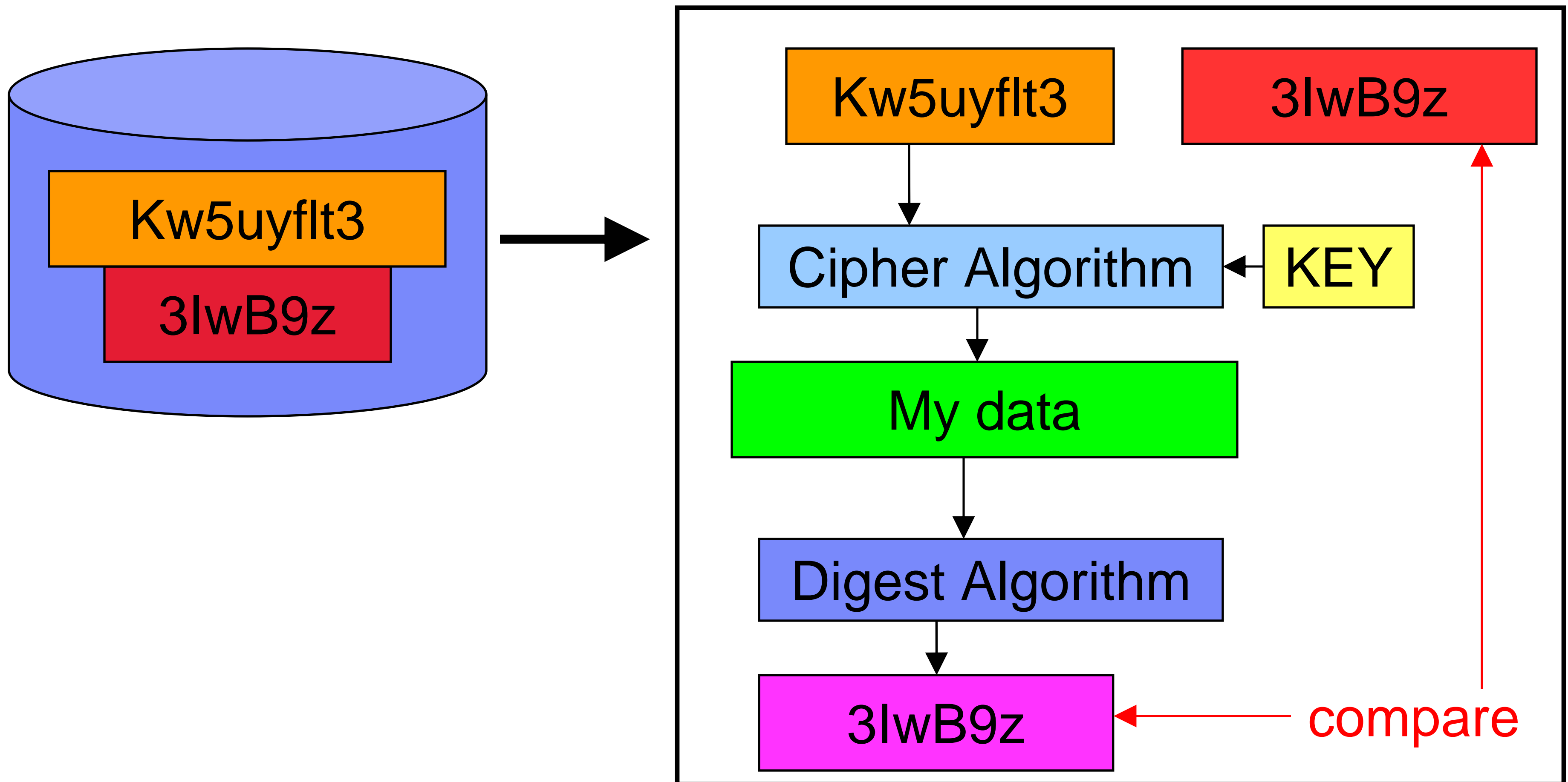
Data Integrity with Data Encryption Steps - Part 1

- **Data creation process**

1. Create the data
2. Calculate the digest of the data
3. Encrypt the data
4. Write the encrypted data and digest to DASD

© 2008 IBM Corporation

Data Integrity with Data Encryption Example – Part 2



© 2008 IBM Corporation

Data Integrity with Data Encryption Steps - Part 2

- **Data access process**
  1. Read the encrypted data and saved digest from DASD
  2. Decrypt the data
  3. Calculate the digest of the decrypted data
  4. Compare the digest just calculated to the saved digest value – if they values match, the data is valid

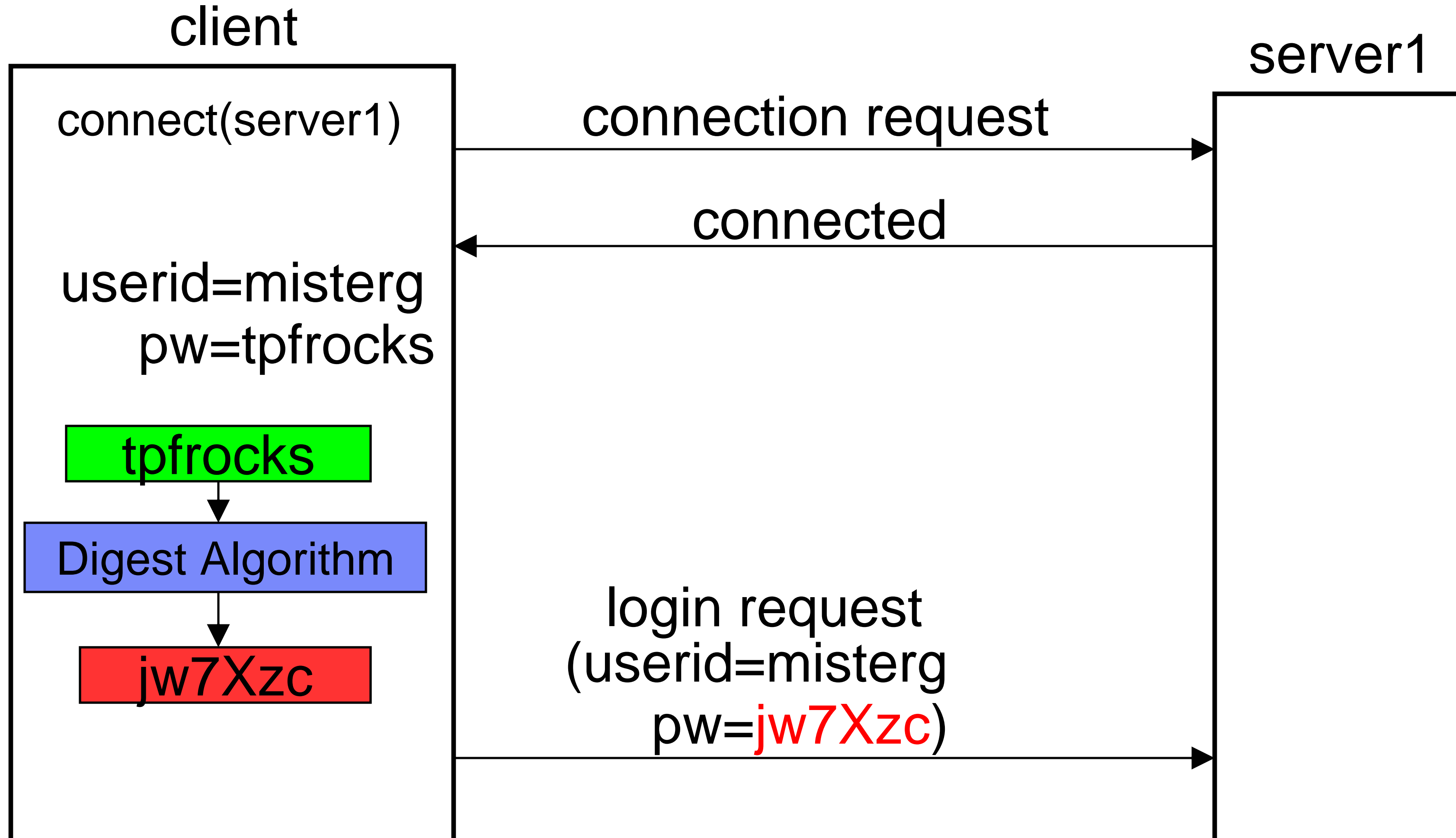
© 2008 IBM Corporation

Using Digests to Pass/Verify Sensitive Data

- **Digests can be used to send sensitive data (like passwords) over public networks**
  - Send the digest of the password rather than the password itself in the clear
- **Added security can be achieved using a *salt***
  - A salt is extra bits concatenated to the data (password) as input to the digest algorithm
  - Greatly reduces the effectiveness of certain attack methods like dictionary attacks
- **Replay attacks can be prevented using a *nonce***
  - Number used once (nonce) is random data that changes each time the data (password) is exchanged

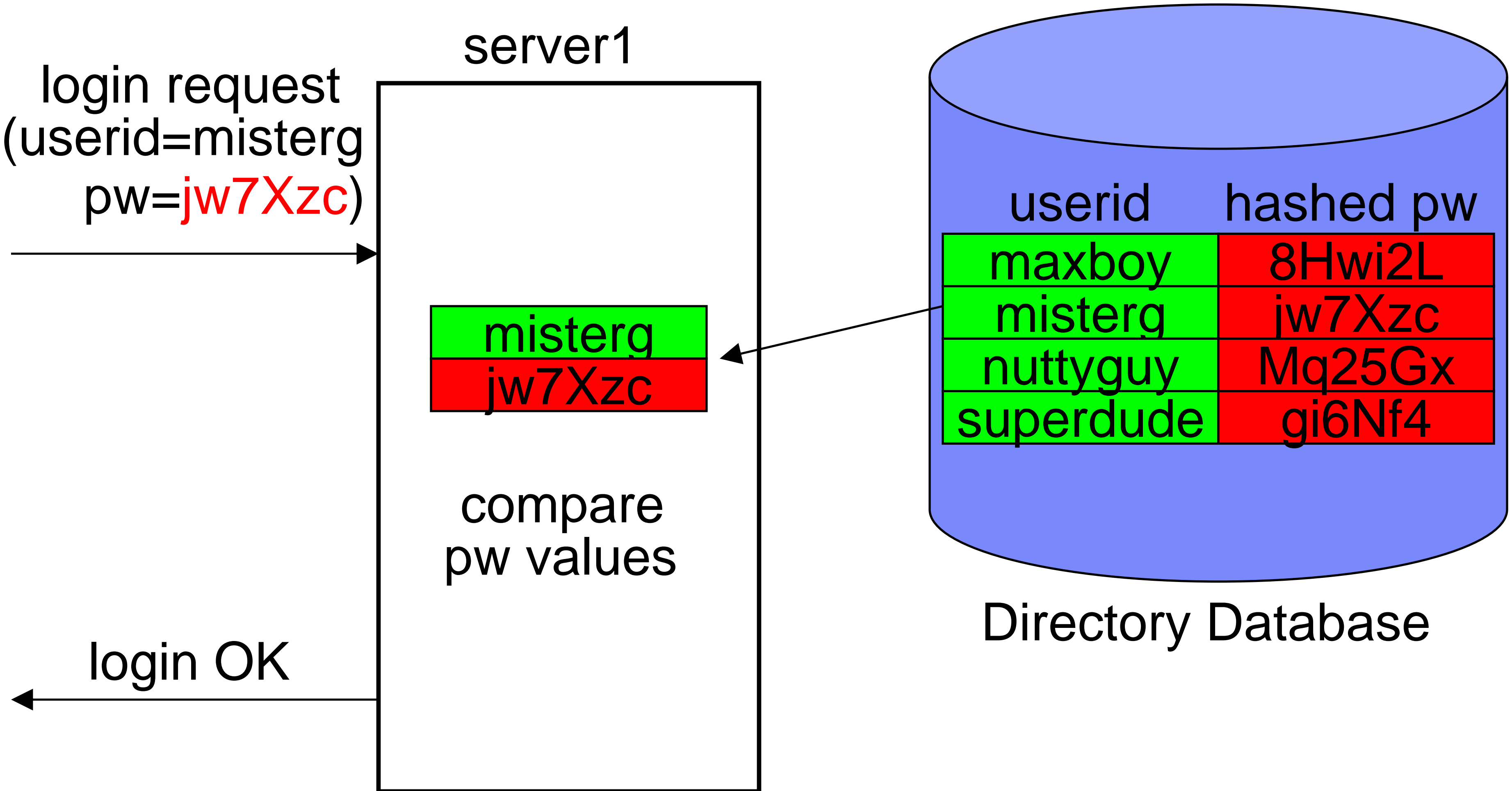
© 2008 IBM Corporation

Simple Hashed Password Example – Part 1



© 2008 IBM Corporation

Simple Hashed Password Example – Part 2



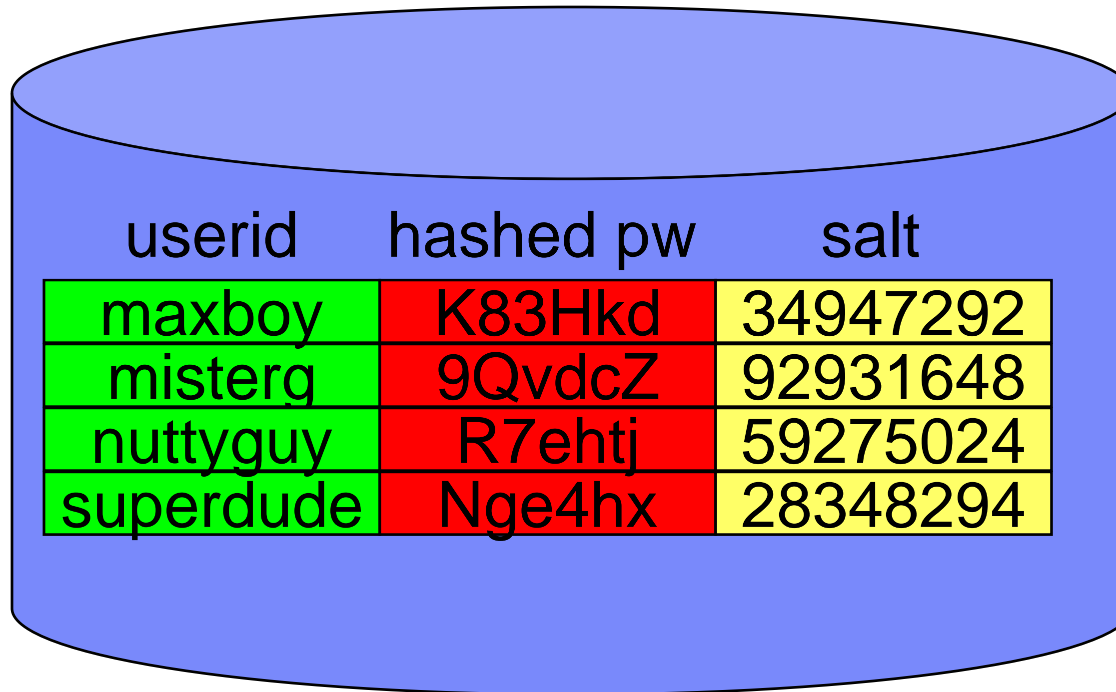
© 2008 IBM Corporation

Simple Hashed Password Example Steps

1. **Client connects to the server (server1)**
2. **Client enters userid=misterg and password pw=tpfrocks**
3. **Client creates a hash (digest) of the password**
4. **Client sends login request to the server containing the client's userid and hashed password value**
5. **Server finds the directory entry for the specified client (userid=misterg) in the directory database**
6. **Server verifies that the hashed password value passed by the client matches the hashed password value in the directory database entry**
7. **Server sends a login complete message to the client**

© 2008 IBM Corporation

Hashed Password with Salt Example – Part 1



userid	hashed pw	salt
maxboy	K83Hkd	34947292
misterg	9QvdcZ	92931648
nuttyguy	R7ehtj	59275024
superdude	Nge4hx	28348294

Directory Database

A unique salt value exists in each entry

© 2008 IBM Corporation

Hashed Password with Salt Example – Part 2

client

server1

connect(server1)

connection request

connected

userid=misterg
pw=tpfrocks

login request(userid=misterg)

salt=92931648

tpfrocks92931648

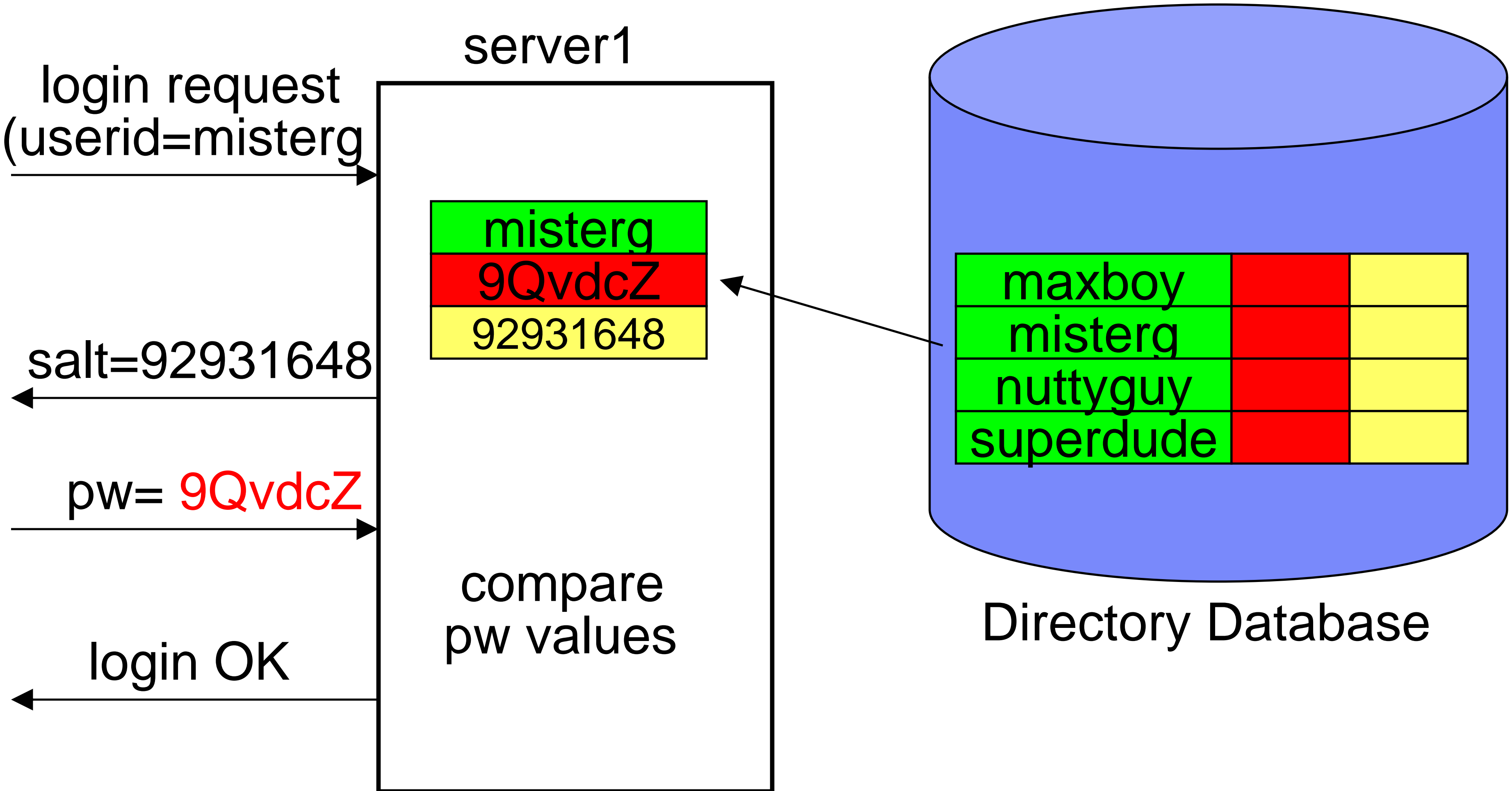
Digest Algorithm

9QvdcZ

pw= 9QvdcZ

© 2008 IBM Corporation

Hashed Password with Salt Example – Part 3



© 2008 IBM Corporation

Hashed Password with Salt Example Steps

1. **Client connects to the server (server1)**
2. **Client enters userid=misterg and password pw=tpfrocks**
3. **Client sends login request to server containing client's userid**
4. **Server finds the directory entry for the specified client (userid=misterg) in the directory database**
5. **Servers sends the salt value for this userid to the client**
6. **Client creates a hash (digest) of the password plus the salt**
7. **The client sends the hashed salted password to the server**
8. **Server verifies that the password value passed by the client matches the hashed salted password value in the directory database entry**
9. **Server sends a login complete message to the client**

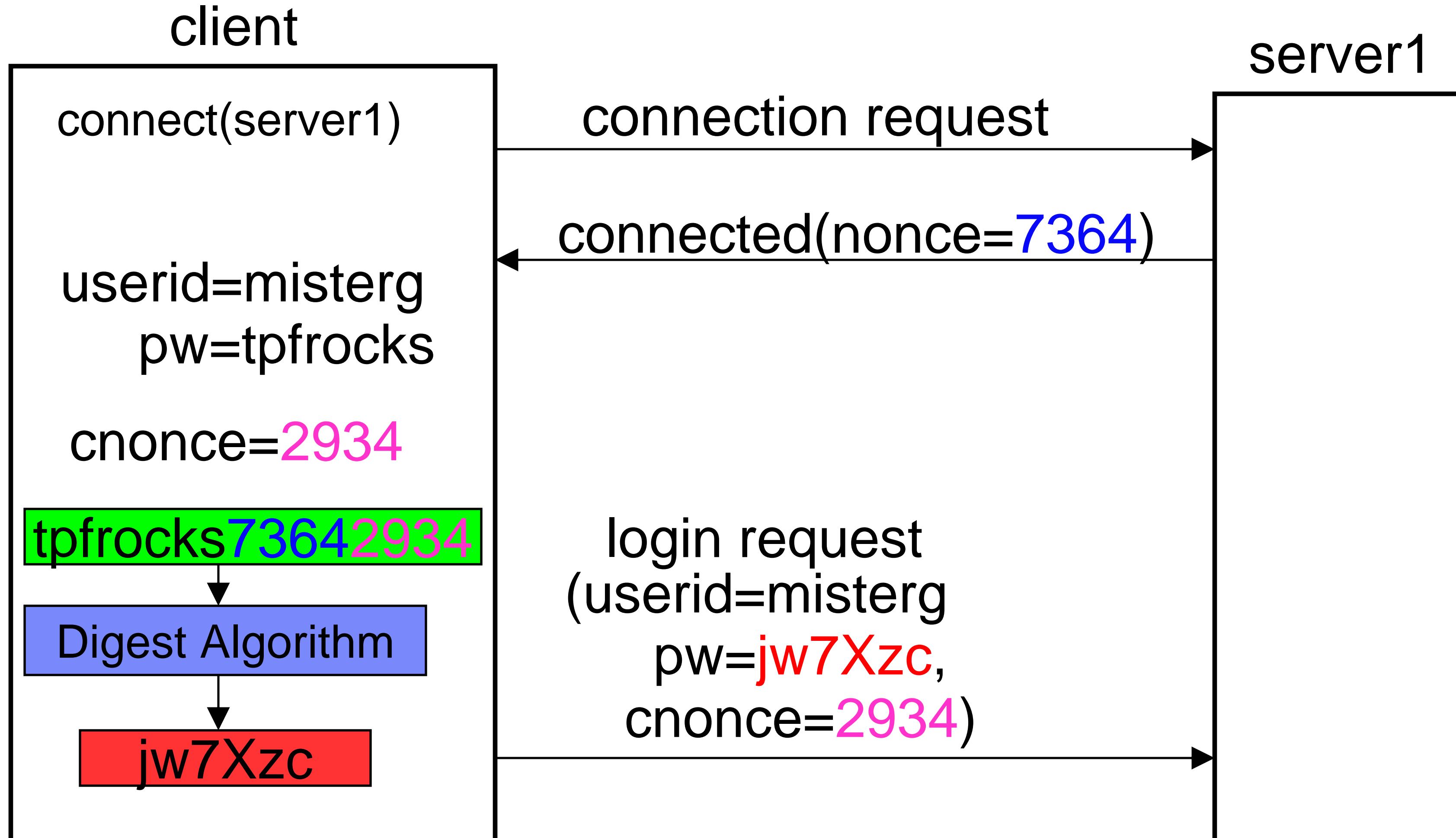
© 2008 IBM Corporation

Other Options Using Salt

- **To avoid the extra flows between client and server to obtain the salt value, other implementations options include:**
  - Having the salt value be the value of the userid
  - Using the same salt value for all users and the only places that know the salt value are the client and server software
    - The salt value never flows over the network and is not stored in the directory database

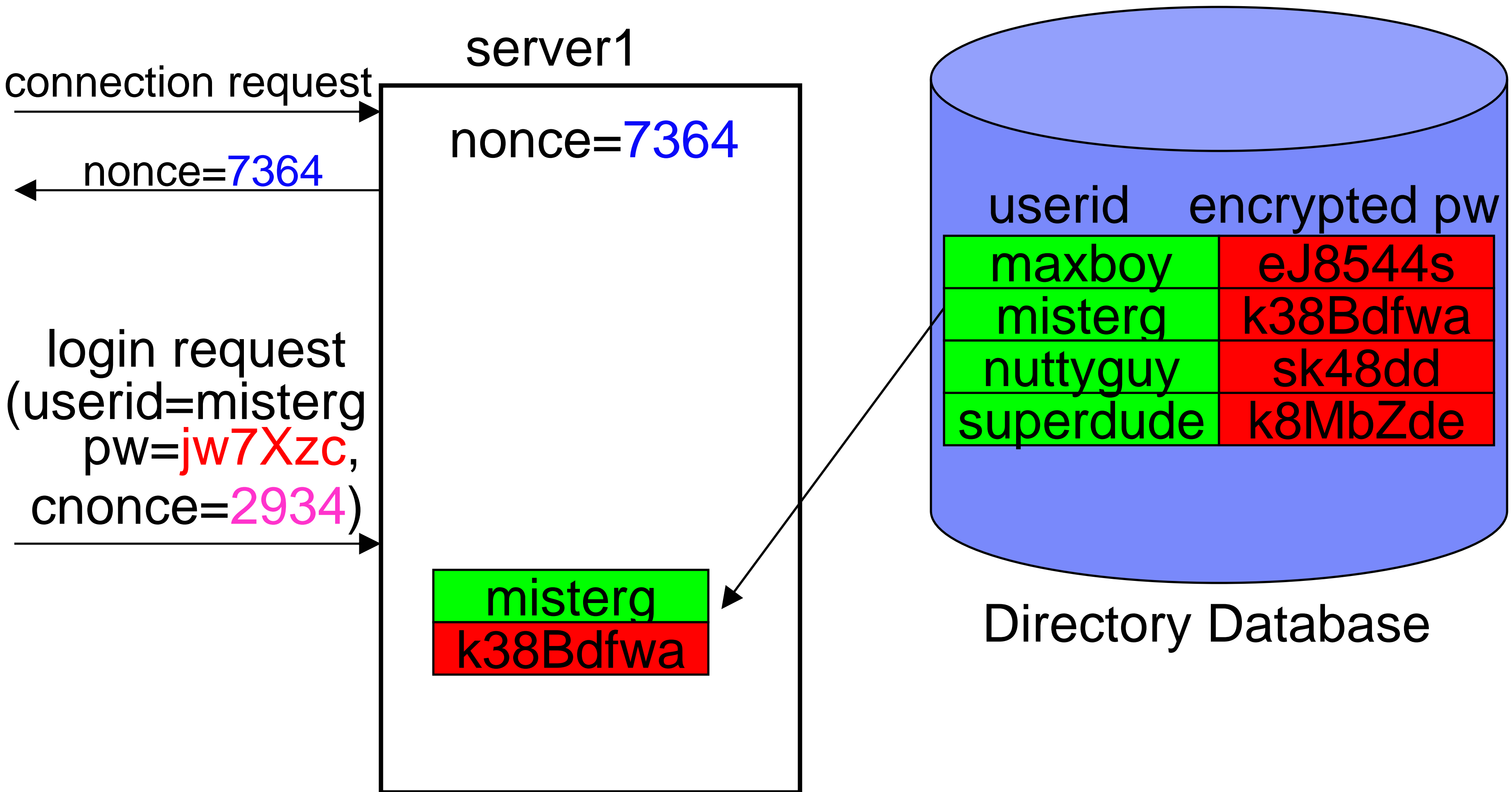
© 2008 IBM Corporation

Password Example with Nonce – Part 1



© 2008 IBM Corporation

Password Example with Nonce – Part 2



© 2008 IBM Corporation

Password Example with Nonce – Part 3

login request
(userid=misterg
pw=jw7Xzc,
cnonce=2934)

server1

nonce=7364

k38Bdfwa

KEY

Cipher Algorithm

tpfrocks

tpfrocks73642934

Digest Algorithm

jw7Xzc

compare hash values

login OK

© 2008 IBM Corporation

Password Example with Nonce Steps

1. Client connects to the server (server1)
2. Server generates a nonce and sends the value to the client
3. Client enters userid=misterg and password pw=tpfrocks
4. Client generates a client nonce (cnonce)
5. Client creates a hash (digest) of the password plus the nonce plus the cnonce
6. Client sends login request to server containing client's userid, hashed password, and the cnonce
7. Server finds the directory entry for the specified client (userid=misterg) in the directory database. The directory entry contains the encrypted password value
8. Server decrypts the password value from the directory database entry
9. Server creates a hash (digest) of the decrypted password plus the nonce plus the cnonce
10. Server verifies that the password value passed by the client matches the value calculated in step 9
11. Server sends a login complete message to the client

© 2008 IBM Corporation

The Power of the Nonce

- **In the previous example, the nonce value was 7364 and the cnonce value was 2934**
- **The next time this user (misterg) does a login, the generated nonce and cnonce values will be different; therefore, the hashed password value passed from client to server will be different**
  - This prevents an attacker with access to old network traces from pretending to be this user (replay attack)

© 2008 IBM Corporation

Combining Methods

- **It is possible to use both salt and nonce methods at the same time**
  - Input to the digest routine is the password plus the salt plus the nonce plus the cnonce
- **It is also possible to use nonce methods without requiring the password to be encrypted (via symmetric key cryptography) in the directory database entry**
  - The database entry would contain the hashed (digest) password value (including the salt)
  - Two hashes are done to create the password value passed from client to server
    - First hash is on the password plus salt
    - Second hash is on the first hash value plus the nonce and cnonce

© 2008 IBM Corporation

Public Key Cryptography

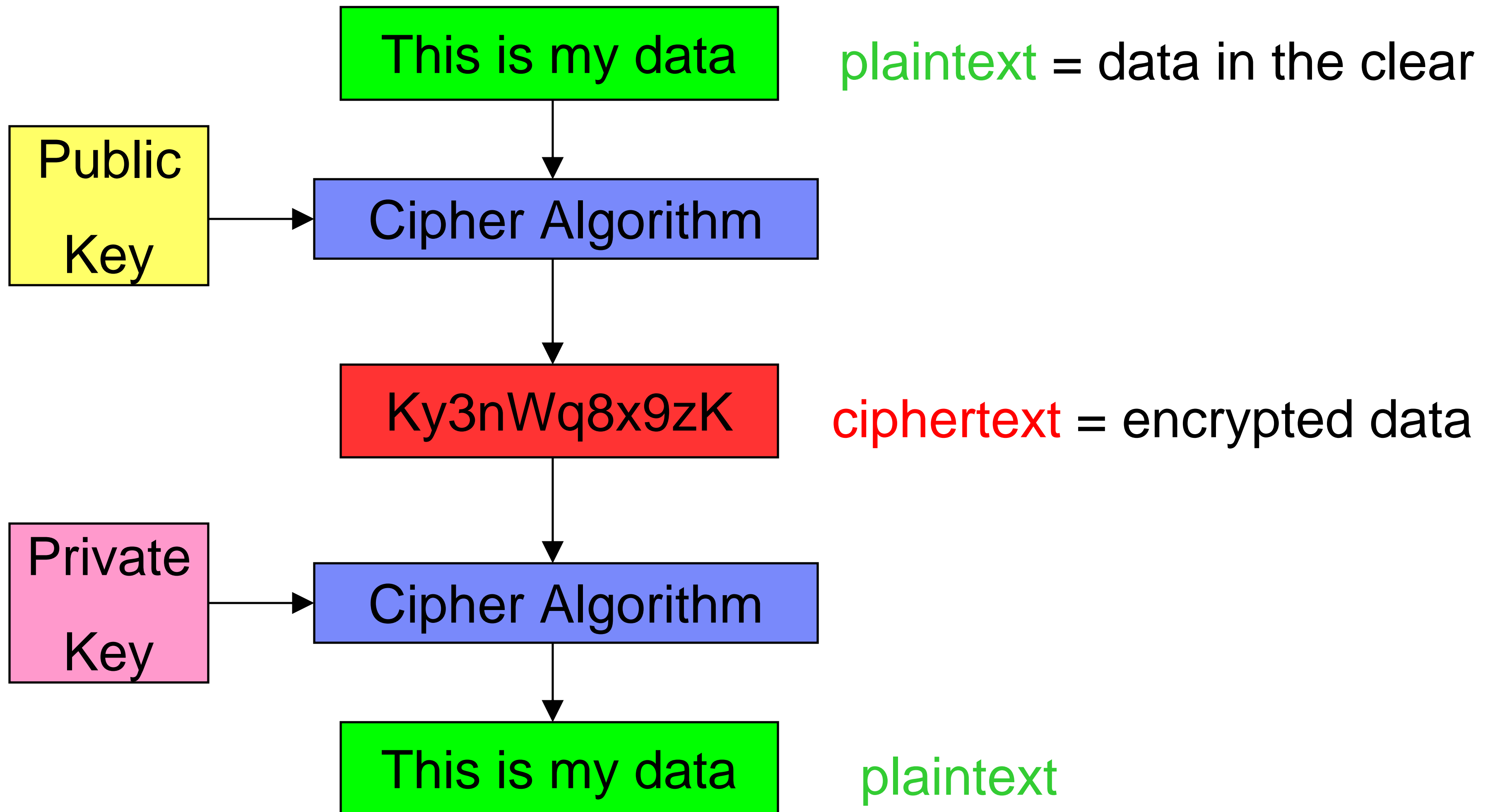
© 2008 IBM Corporation

Public Key (Asymmetric) Cryptography

- **Public key pair**
  - Public key
    - Can be openly distributed
  - Private key
    - Must be kept secret/protected
- **Any data encrypted with a public key can only be decrypted using the corresponding private key**
- **Any data encrypted with a private key can only be decrypted using the corresponding public key**

© 2008 IBM Corporation

Public Key Cryptography Example



© 2008 IBM Corporation

The Roots of Public Key Cryptography

- **Development of asymmetric key algorithms began in the early 1970s**
- **Diffie-Hellman (DH) key exchange was published in 1976**
- **Rivest, Shamir, and Adleman (RSA) algorithm was published in 1978**
  - RSA became very widely used after its patent expired in 2000
  - 1024-bit keys common, 2048-bit keys used in some instances
- **Digital Signature Standard (DSS) developed in 1991**
  - Based on Digital Signature Algorithm (DSA) that also uses large prime numbers as a fundamental part of its security

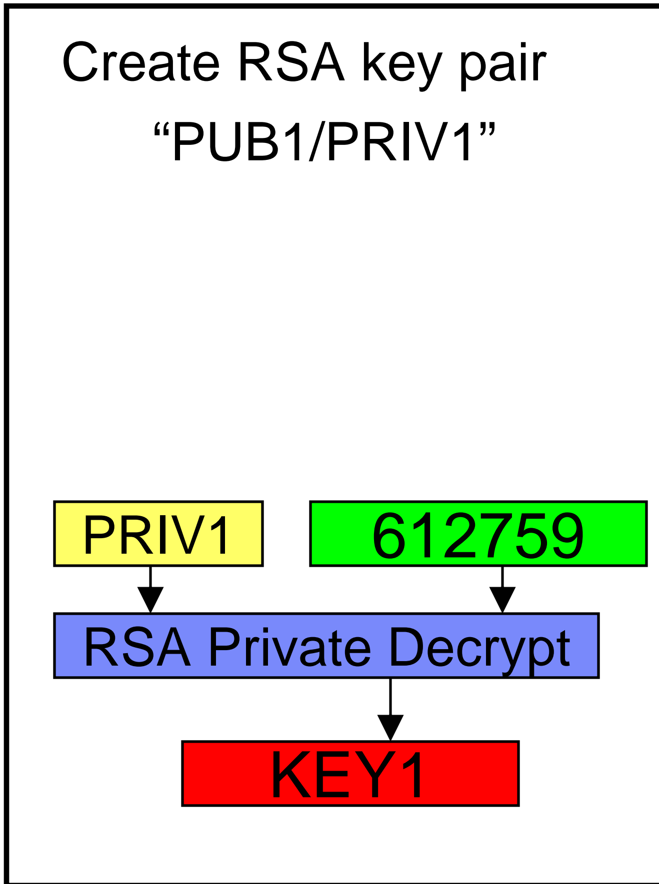
© 2008 IBM Corporation

Why Public Key Cryptography

- **Secure key exchange**
  - Solved the problem of symmetric key distribution/exchange over unsecure (public) networks
- **Digital Signatures**
  - Ability for the receiver of a message (data) to verify that the message was sent by the who the sender claims to be and that the message has not been altered after the sender built the message (non-repudiation)

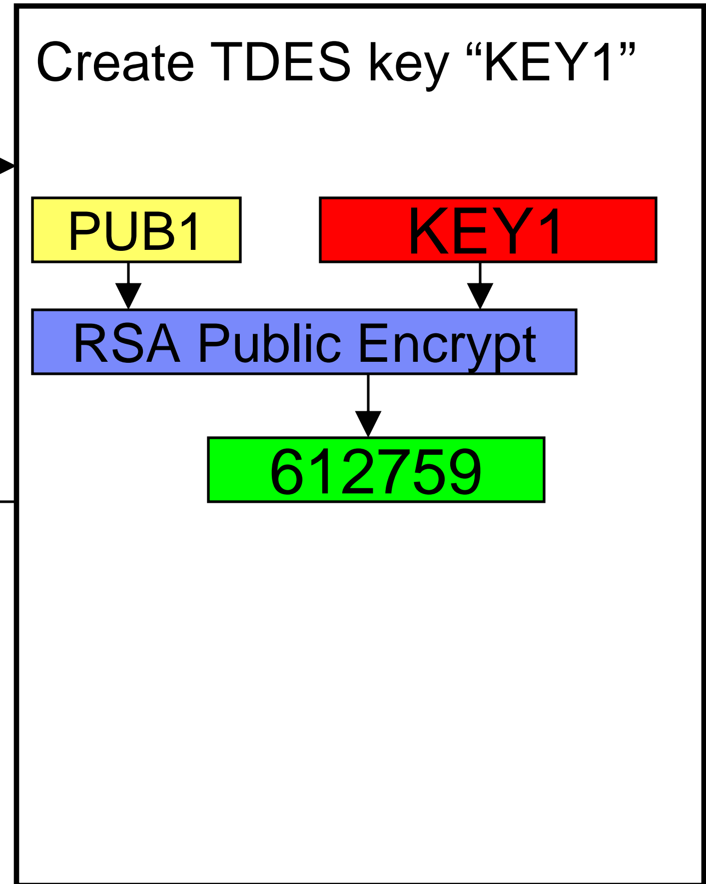
© 2008 IBM Corporation

Secure Key Exchange Example



PUB1

612759



© 2008 IBM Corporation

Secure Key Exchange Steps

1. **Node X creates an RSA public key pair. PUB1 is the public key and PRIV1 is the private key**
2. **Node Y creates a triple-DES (TDES) key called KEY1**
3. **Node X sends its public key (PUB1) to node Y**
4. **Node Y encrypts/wraps the TDES key (KEY1) using node X's public key (PUB1) and sends the encrypted data/key (612759) to node X**
5. **Node X decrypts/unwraps the data (612759) using its private key (PRIV1) and the result is the triple-DES key (KEY1) that node Y created**
   - Only node X has the value of the private key that is needed to decrypt/unwrap the encrypted symmetric (TDES) key

© 2008 IBM Corporation

Public Key Distribution

- **In practice, how a public key is distributed to remote partners requires more security**
  - Cannot just send the public key by itself in many cases because of “man in the middle” attacks
- **Digital Certificates solve this problem**
  - Certificate contains a node’s public key
  - Created/signed by trusted third-party called a certificate authority (CA)

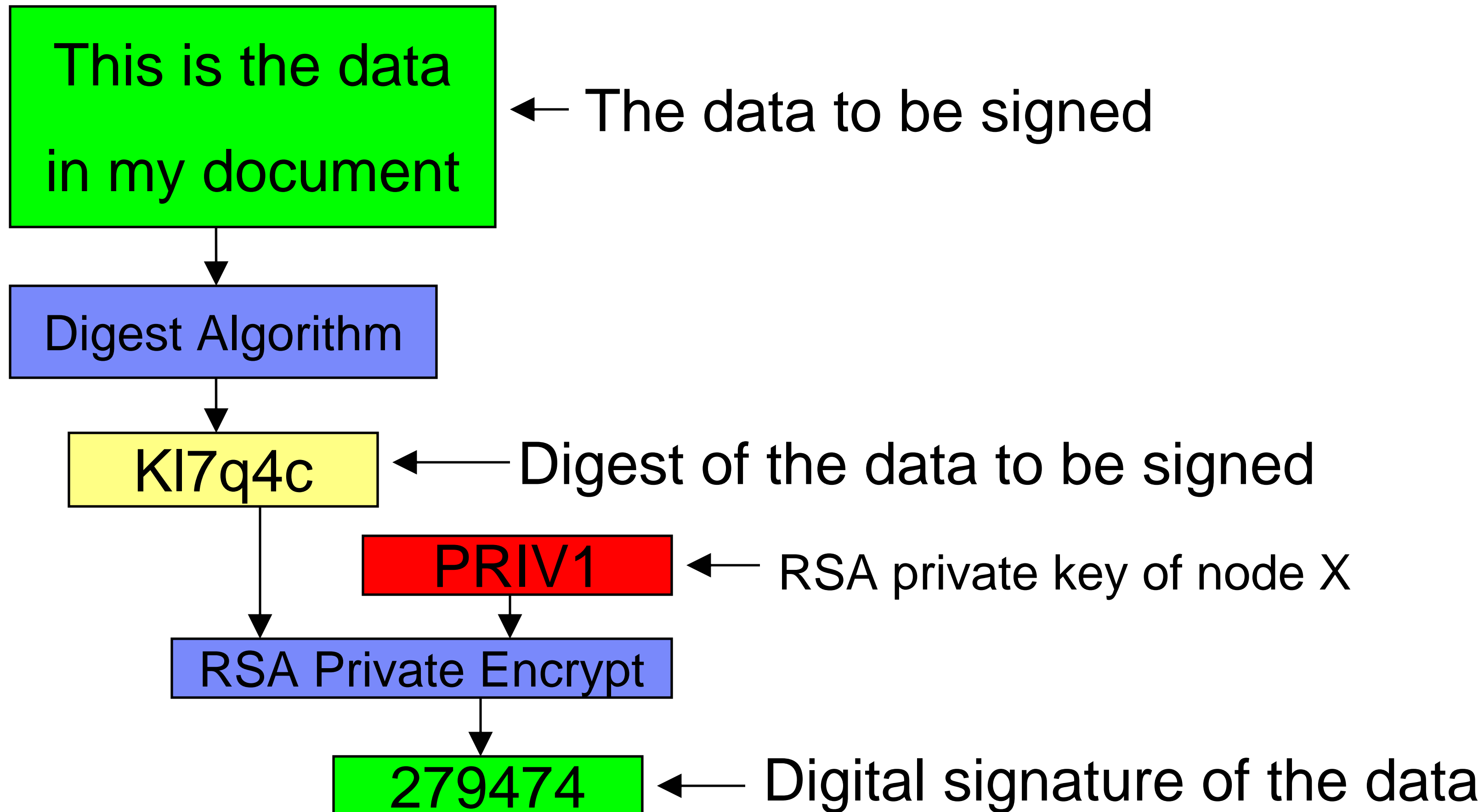
© 2008 IBM Corporation

Digital Signature Details

- **Public key cryptography can only encrypt/decrypt small amount of data**
  - Amount of data cannot be larger than the key size. For example, a 1024-bit RSA key cannot encrypt data larger than 128 bytes
- **You can digitally sign any length data**
  - To create a digital signature, first you create a digest of the data, then encrypt the digest using the private key
    - Even with SHA-512, the digest is only 64 bytes long and; therefore, can be encrypted using a 1024-bit RSA key

© 2008 IBM Corporation

Creating a Digital Signature on Node X



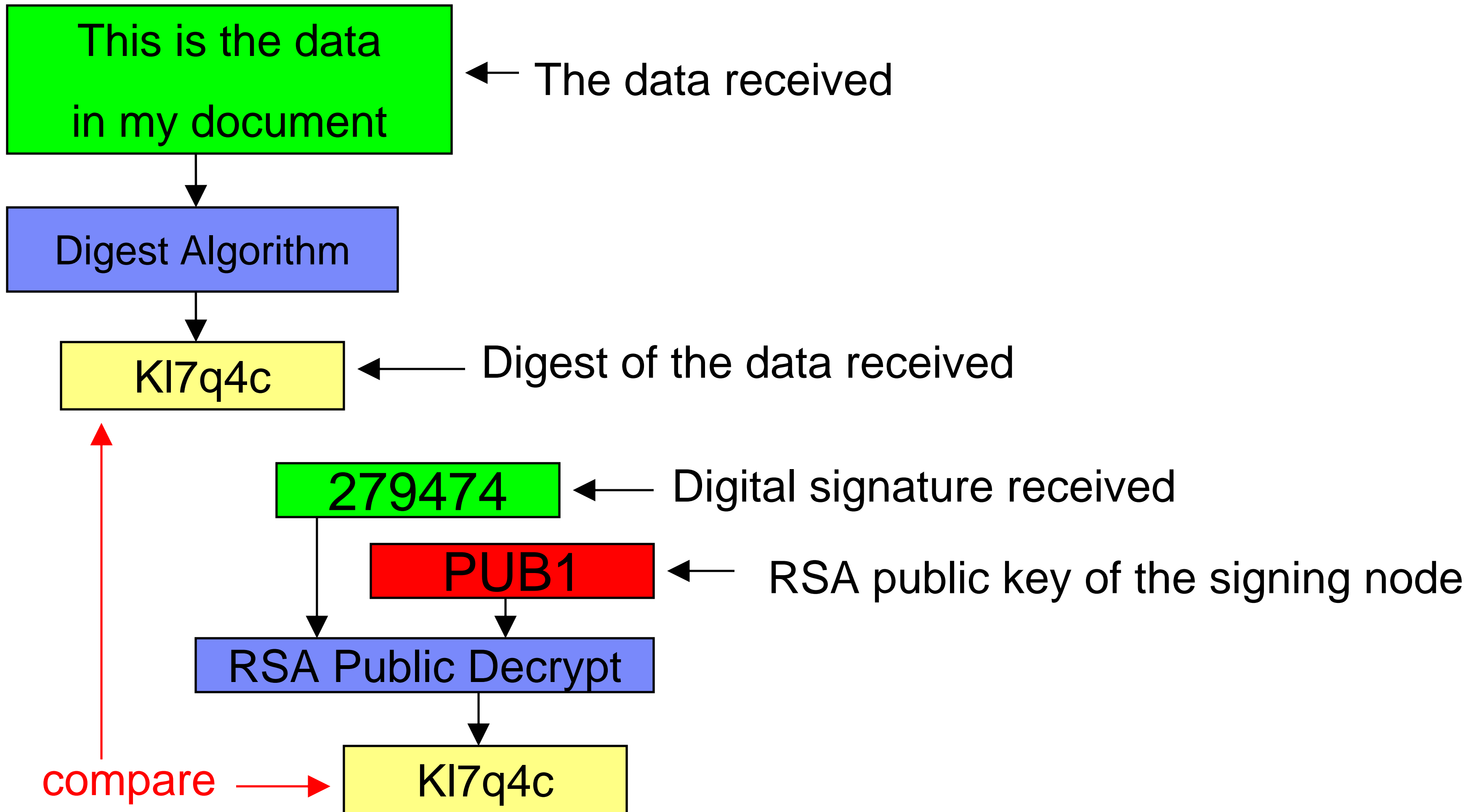
© 2008 IBM Corporation

Steps for Creating a Digital Signature on Node X

1. **A digest is created of the data to be signed**
2. **The digest is encrypted using node X's RSA private key and the result is a digital signature of the data**
3. **The data along with the digital signature are sent to node Y**

© 2008 IBM Corporation

Verifying a Digital Signature on Node Y



© 2008 IBM Corporation

Steps for Verifying a Digital Signature on Node Y

1. **Node Y creates a digest of the data received from node X**
2. **Node Y takes the digital signature received from node X and decrypts it using node X's RSA public key. The result should be the digest of the data**
3. **Node Y compares the output of steps 1 and 2. If the values match, the digital signature is valid, meaning node X did sign this data and the data has not been altered**

© 2008 IBM Corporation

Public Key Cryptography Trends

- **Many public key algorithms, including RSA, are designed on the premise that there is no viable way to factor the product of two very large prime numbers**
  - Because of advances in computer technology and factoring methods, key sizes have been increasing
    - 512-bit to 1024-bit, some 2048-bit, and even 4096-bit
    - Larger keys are a problem for many small devices and also for high volumes in server cryptographic hardware
- **Elliptic Curve Cryptography (ECC)**
  - Different type of public key cryptography
  - Provides equal or greater security with smaller key sizes
  - Several patents apply to ECC technology, which has prevented wide scale usage

© 2008 IBM Corporation

Hardware Cryptography

On System z

© 2008 IBM Corporation

IBM Hardware Cryptography… In The Beginning

- **CMOS Cryptographic Coprocessor Facility (CCF)**
  - Secure key operations on IBM Gx processors
- **4758 PCI cryptographic coprocessor**
  - PCICC feature on System z (introduced on z900)
  - Also available on System i, System p, and System x
  - Applications use IBM Common Cryptographic Architecture (CCA) APIs
  - Secure key operations only
  - Did some CCF functions more efficiently
  - Managed via Trusted Key Entry (TKE) workstation

© 2008 IBM Corporation

PCIX Cryptographic Coprocessor (**PCIXCC**)

- **Introduced on z990 in 2003**
- **Combined CCF and PCICC functions into a single crypto adapter**
- **Highly secure card (FIPS-140, level 4 certification)**
- **Supports secure key operations**

© 2008 IBM Corporation

PCI cryptographic accelerator (**PCICA**)

- **Supported on z900 and z990**
  - Also supported on System i and System p
- **Does RSA clear key operations very quickly**
- **Designed to improve SSL performance, specifically starting SSL sessions**
  - Starting an SSL session requires an RSA private key decrypt operation by the server
  - Each PCICA can start up to 1000 SSL sessions per second

© 2008 IBM Corporation

Crypto Express2

- **Introduced on z990 in 2005**
- **Combined PCICA and PCIXCC into one feature (adapter)**
- **Runs in one of two modes:**
  - Crypto Express2 Coprocessor (CEX2C)
    - Same normal path (secure key) operations as PCIXCC
  - Crypto Express2 Accelerator (CEX2A)
    - Fast path (clear key RSA) operations like PCICA
    - Each CEX2A can start up to 3000 SSL sessions per second

© 2008 IBM Corporation

Central Processor Assist for Cryptographic Functions (**CPACF**)

- **Coprocessor that comes with the server**
  - Not a separate crypto adapter
- **Performs very high volume clear-key operations**
- **Designed for heavy use crypto applications**
- **Introduced on z990**
  - Supported DES, TDES, and SHA-1
- **z9 added AES-128 and SHA-256 support**
- **z10 added AES-256 and SHA-512 support**

© 2008 IBM Corporation

History of Cryptography on TPF

© 2008 IBM Corporation

Cryptographic APIs for Application Programs

- **CIFRC API (predates TPF 4.1)**
  - Encrypt/decrypt 8 bytes of data using DES in software
- **CRYPC and tpf_cryptc APIs**
  - Clear key APIs where application passes the key as input
    - Key management is user responsibility
  - Encrypt or decrypt variable length user data using:
    - DES or TDES (TPF 4.1 PUT 21, z/TPF PUT 2)
    - AES-128 or AES-256 (z/TPF PUT 3)
  - Uses CPACF hardware if installed (for DES, TDES, AES-128); otherwise, software encryption is used

© 2008 IBM Corporation

Secure Sockets Layer (SSL) Support

- **Initial support (TPF 4.1 PUT 15)**
  - Allows data to be transmitted over public networks in a secure manner
- **Hardware acceleration for starting SSL sessions**
  - Using PCICA (TPF 4.1 PUT 21, z/TPF PUT 2)
  - Using CEX2A (z/TPF PUT 2)
- **AES cipher suites for SSL (z/TPF PUT 3)**
- **Hardware acceleration for SSL data messages using CPACF**
  - DES, TDES, SHA-1 (TPF 4.1 PUT 21, z/TPF PUT 2)
  - AES-128 (z/TPF PUT 3)
  - AES-256 (z/TPF PUT 5)

© 2008 IBM Corporation

Digest Algorithm APIs

- **Create or verify digests of user data**
  - SHA-1 APIs (z/TPF PUT 3)
  - SHA-256 APIs (z/TPF PUT 4)
- **Data to be digested can be in contiguous storage or non-contiguous storage**
- **Uses CPACF hardware**

© 2008 IBM Corporation

Hardware Tape Encryption (z/TPF, PUT 4)

- **Support for IBM TS1120 encrypted tape drives**
- **Ability to mount output tapes as encrypted**
  - Tape control unit encrypts the data before writing it to the tape
- **Ability to read encrypted tapes**
  - Tape control unit communicates with its key manager to obtain the necessary decryption key
- **Encrypted tapes allow tapes containing sensitive data to:**
  - Leave the data center (archive, exchange data with business partners)
  - Be stored locally and control access to that data

© 2008 IBM Corporation

Secure Key Management

- **Ability to create and manage symmetric keys in a secure manner**
  - DES, TDES, AES-128 (z/TPF PUT 4)
  - AES-256 (z/TPF PUT 5)
- **tpf_encrypt_data and tpf_decrypt_data APIs**
  - Ability to encrypt/decrypt large amounts of user data
  - Applications reference keys by name/token
- **Uses CPACF hardware**

© 2008 IBM Corporation

Trademarks

- IBM is a trademark of International Business Machines Corporation in the United States, other countries, or both.
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.

- Notes
- Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
- All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
- This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

- All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

- Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

- Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

- This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.

© 2008 IBM Corporation