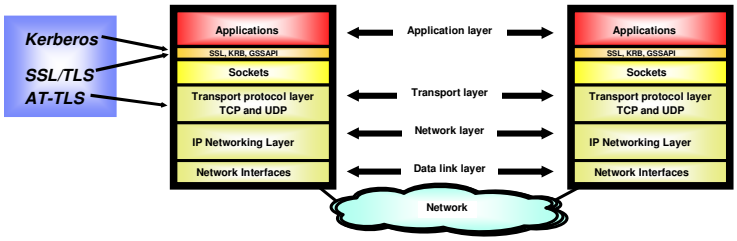
CS z/OS Application Transparent Transport Layer Security (AT-TLS) Background and Introduction

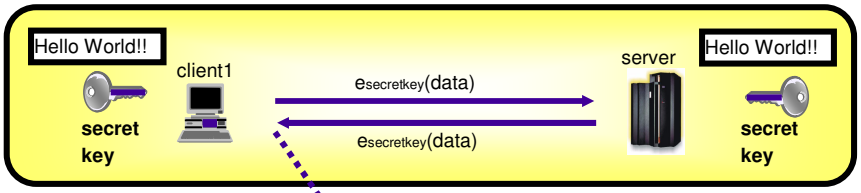
AT-TLS background and introduction

SSL/TLS and Kerberos overview on z/OS



- **Transport Layer Security (TLS) is defined by the IETF**
  - Based on Secure Sockets Layer (SSL)
    - SSL originally defined by Netscape to protect HTTP traffic
  - TLS defines SSL as a version of TLS for compatibility
    - TLS clients and server should drop to SSL V3 based on partner's capabilities
- **Traditionally provides security services as a socket layer service**
  - Requires reliable transport layer
    - UDP applications cannot be TLS enabled
  - Application source code changes are generally needed to SSL/TLS-enable a sockets program
- **z/OS applications can be TLS enabled with System SSL**
  - System SSL is part of z/OS Integrated Security Services element
  - System SSL supports z/OS UNIX System Services C/C++ applications only
- **Kerberos support is implemented using the Kerberos and GSSAPI functions of the z/OS Security Server and provides:**
  - Third-party authentication
  - Optional message integrity
  - Optional message privacy (encryption)
- **The Kerberos environment must be set up on the z/OS system.**
  - The Kerberos support is documented in the publication "Network Authentication and Privacy Service: Administration", SC24-5926
- **Some z/OS applications that are kerberized:**
  - FTP server and client
  - UNIX Telnet daemon (OTelnetD)
  - UNIX RSH daemon (ORshD)
  - z/OS WAS Server
- **Mostly of value where a Kerberos-based infrastructure already is in place**

Symmetric encryption (secret or shared key encryption)



- Symmetric encryption algorithms are fast and efficient.
- Key management is a major concern for secret key-based encryption - to whom do we send the key and how do we do that safely?

esecretkey(data)

client2

*x-yz/%¤&&8/dvvvv

Data that is encrypted with the secret key can only be decrypted by someone who has the same secret key (sharing the same secret).

Someone who does not have a copy of the secret key cannot decrypt any intercepted data.

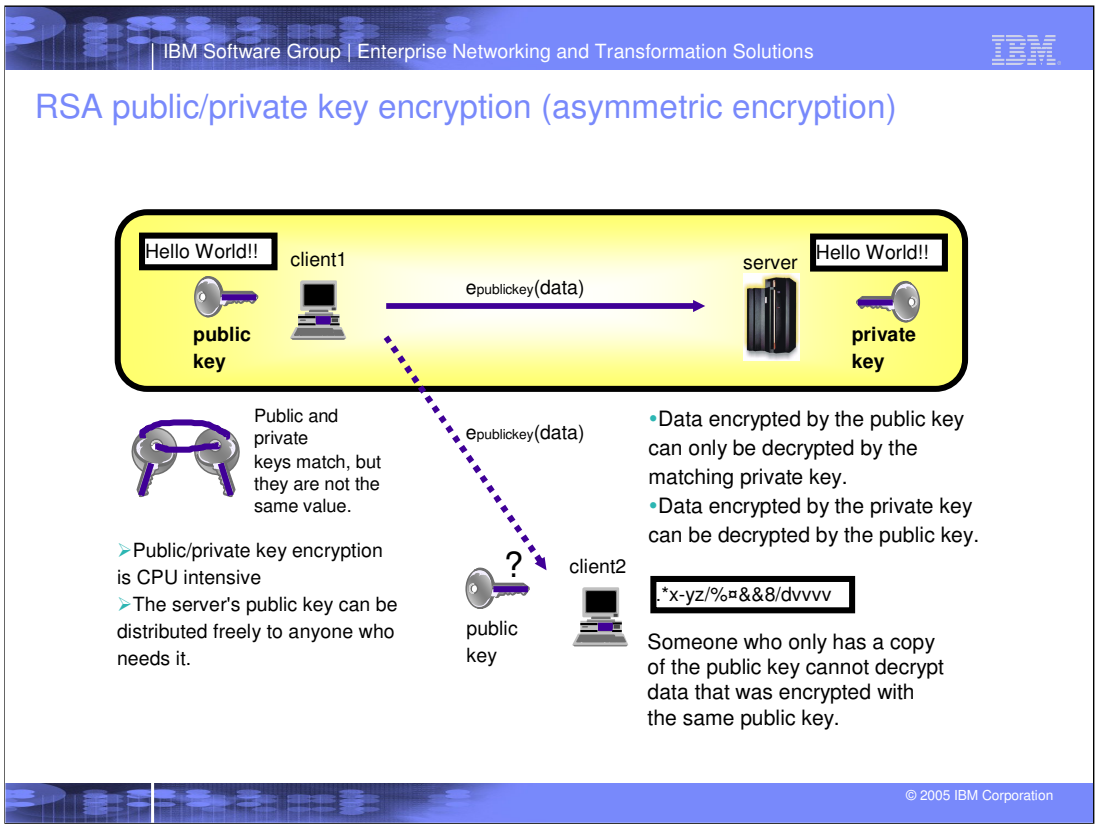
Examples of symmetric encryption algorithms are: DES and AES

Encryption is based on either a symmetric key or on a set of asymmetric keys.

In the Web server context, the asymmetric key concept is the one that is used in most cases.

A message that has been encrypted using the public key can only be decrypted using the accompanying private key.

A message that has been encrypted using the private key can be decrypted by everyone who has the accompanying public key.



Encryption is based on either a symmetric key or on a set of asymmetric keys.

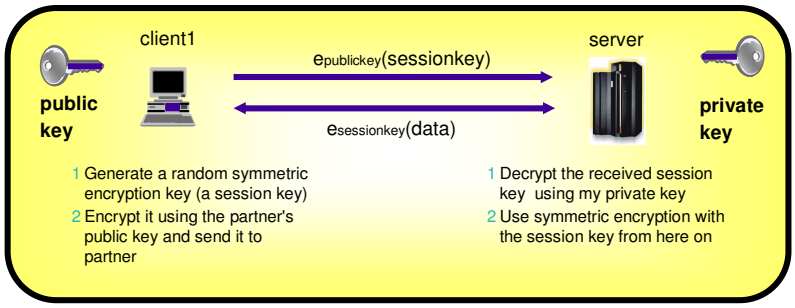
In the Web server context, the asymmetric key concept is the one that is used in most cases.

A message that has been encrypted using the public key can only be decrypted using the accompanying private key.

A message that has been encrypted using the private key can be decrypted by everyone who has the accompanying public key.

RSA does not in itself provide authentication, but combined with digital signatures, RSA can bve used to provide authentication.

Combining public/private key encryption with symmetric encryption - session keys



Secure Sockets Layer (SSL) uses RSA to generate symmetric session keys

1 Server has distributed a public key to client earlier.

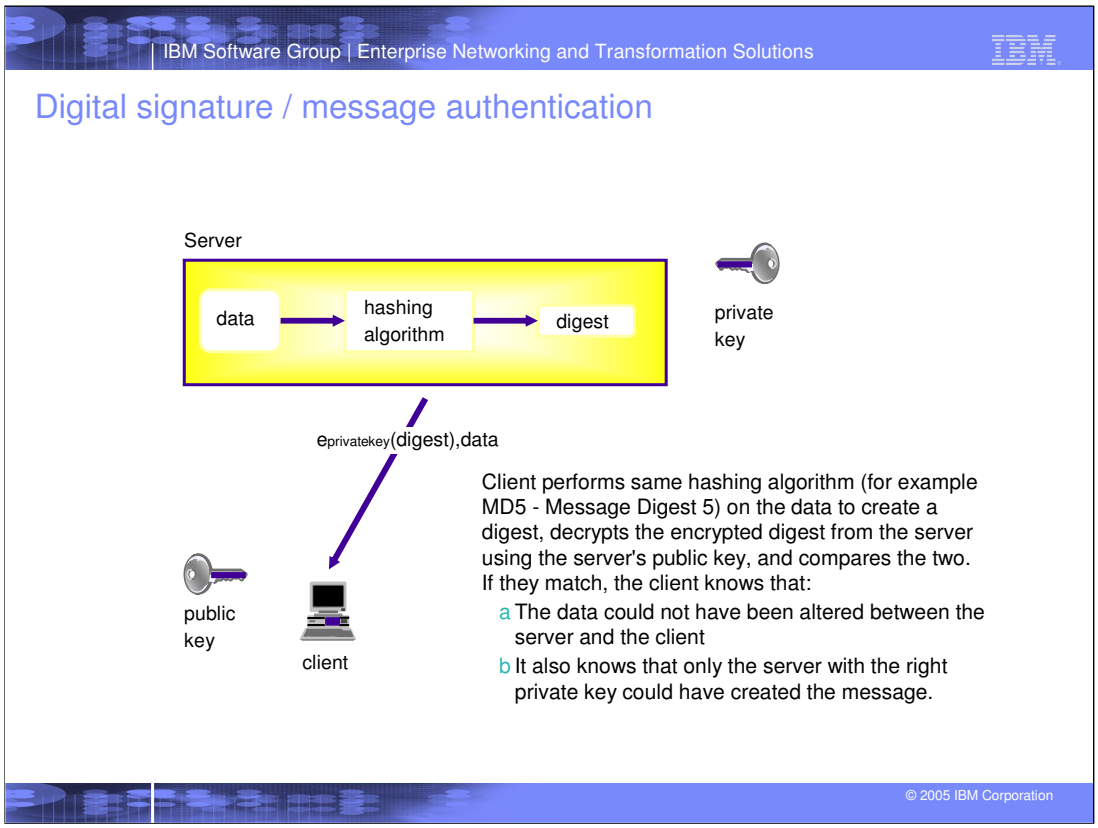
2 Client generates a random session key, encrypts it under the server's public key, and transmits it to the server. Only the server is able to decrypt this message.

3 Server decrypts the message and the server and the client use the session key for succeeding encrypted data exchanges in this session.

The client has the server's public key.

The client generates a random key that is to be used as a session key. The client encrypts the session key under the public key and transmits it to the server. Because the session key has been encrypted with the public key, only the server that has the corresponding private key can decrypt it and obtain the session key.

All further data exchanges are then encrypted using the session key, which only this client and the server have knowledge about.

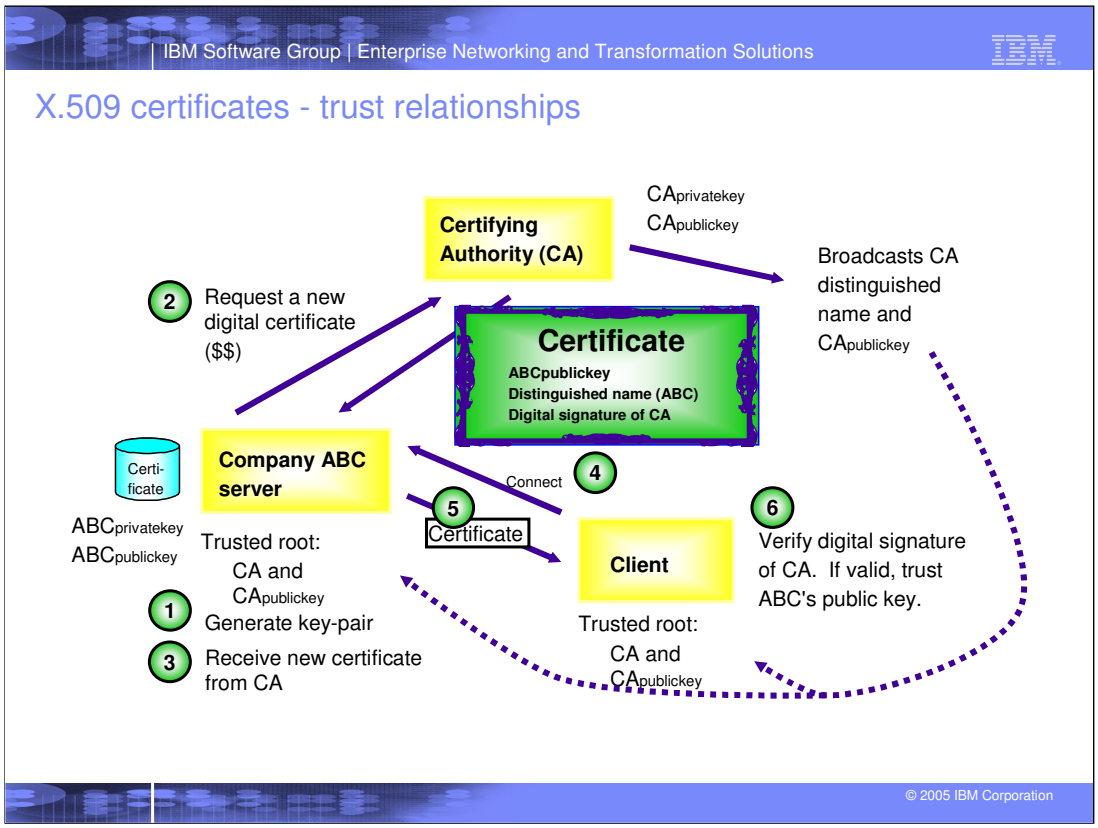


By using digital signatures, a client with a public key that matches the private key of the server, can verify that the message was not altered on the path from the server to the client. If the message had been altered, the encrypted digest could not be altered too, because only the correct server that has the private key would be able to create the correct encrypted digest.

Digital signatures become more complicated, if the data part of the message is encrypted under a session key, but the concept still works even in that situation.

Hashing algorithms are MD5 and SHA.

One can use the same key [pair for digital signature and for encryption, but in general it is not recommend. For digital signatuitures, the



The certificate that is returned from the certifying authority includes:

1 Distinguished name of the company or person (ABC)

2 The public key of ABC (was included in the request that was sent to the CA

3 Distinguished name of the CA

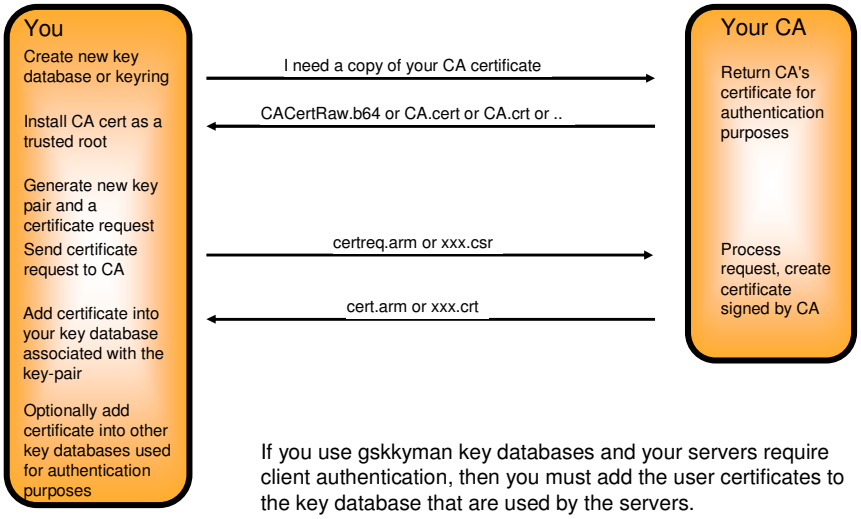
4 Issue date and expiry date

5 Digital signature of the CA

The server stores the certificate and uses it for as long as it is valid.

Servers and browsers define so-called trusted roots, which are the distinguished names of certifying authorities and their public keys. Both the server and the browser must trust the CA that issued the certificate (defined as trusted root) in

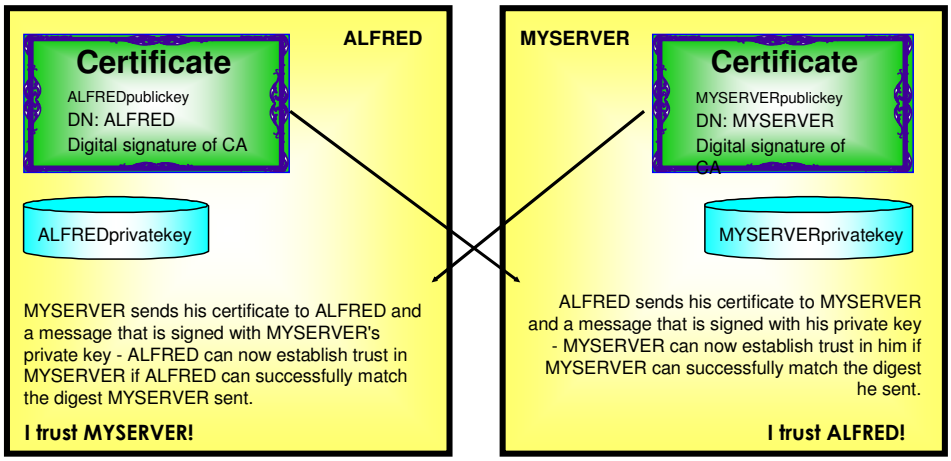
CA flow overview



If you use gskkyman key databases and your servers require client authentication, then you must add the user certificates to the key database that are used by the servers.

In general, if you need client authentication, using RACF keyrings would be recommended.

Server and client certificates



- In order to create session keys and encrypt the data stream, only the server needs to have a certificate.
- If the client has a certificate, then the server can use that to authenticate the client. On z/OS a client certificate can be mapped to a RACF user ID. Password authentication may still be required by the server application, but the password can at least now be submitted in encrypted form over the network.

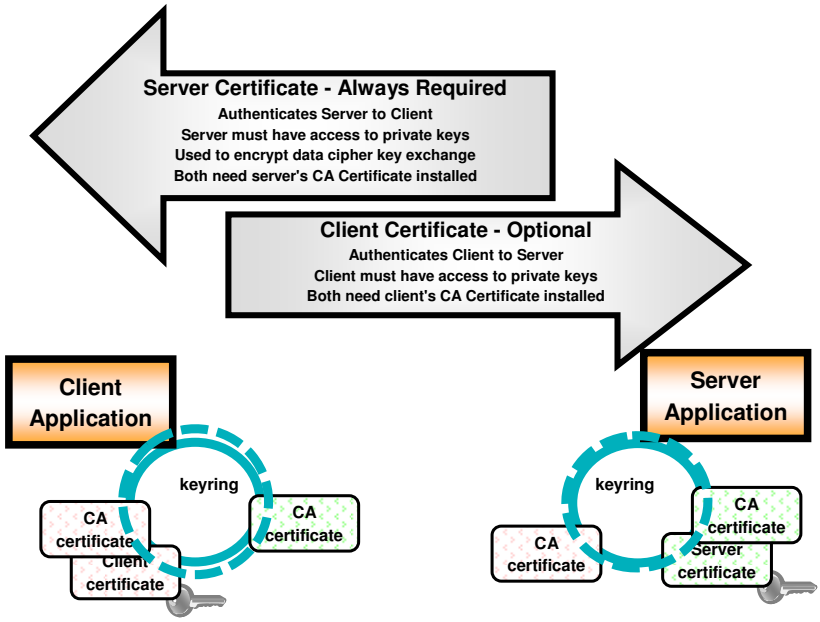
Background information

- **SSL: Secure Sockets Layer**
  - Created by Netscape
  - Originally implemented inside Web clients and servers
  - Above sockets and below application protocol
  - SSLv1 no longer supported
  - SSLv2 still some
    - mostly public access compatibility concern
  - SSLv3.0 improved security

- **TLS: Transport Layer Security**
  - TLSv1.0 (SSLv3.1)
  - IETF RFC 2246

- **End-to-end application pipe**
  - TCP connections
  - Server authentication
  - Optional client authentication
  - Authentication
    - Public key cryptography, third-party signed certificate
  - Data privacy
    - Negotiated private key cryptography
    - SSL record protocol

Background Information



Current SSL/TLS API support on z/OS is limited to C and Java

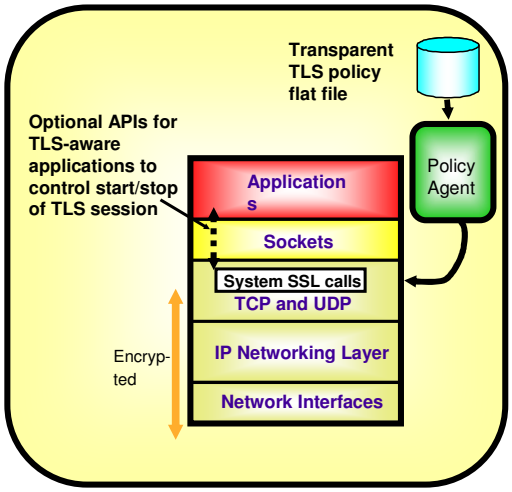
- **Application layer implementation**
  - Development expense repeated for each application
  - Toolkits available for limited programming environments
    - C and Java
    - Forking or Threaded POSIX model
    - Require application change
  - Many existing z/OS applications do not fit this model
    - COBOL, assembler sockets programs
    - CICS sockets transactions
    - etc.
  - Many applications purchased or otherwise not available for change
    - Source code is needed to enable them for SSL/TLS

- **Application specific deployment**
  - Unique configuration for each application
  - Different levels of SSL/TLS architecture support
  - Not all toolkits support/exploit z/OS and zSeries capabilities
    - RACF keyrings
    - Certificates associated with user IDs
    - Hardware cryptography

Application Transparent Transport Layer Security (AT-TLS)

- **Application Transparent**
  - Support existing applications without change
  - Allow applications to optionally exploit/control advanced features
    - Simple ioctl
    - Extract status, certificate, and associated user ID
    - Permit cleartext negotiation prior to starting secure connection
- **Transport Layer**
  - Implement inside the TCP layer of the stack
  - Common configuration through policy
  - Exploit z/OS
    - RACF
    - SystemSSL
    - ICSF
    - Hardware cryptography
- **Security**
  - Multiple Protocols
    - TLS (SSL V3.1)
    - SSL V3.0
    - SSL V2

Transparent application security: policy-controlled transparent SSL/TLS support - SSL/TLS for all z/OS sockets applications



- **Basic TCP/IP stack-based TLS**
  - TLS process performed at TCP layer without requiring any application change (transparent)
  - All connections to specified port are designated as TLS required
    - Can be further qualified by source/destination IP addresses
  - Transparent TLS policies managed via Policy Agent
- **Transparent TLS can be requested by aplication**
  - Application issues transparent TLS API calls to indicate that connection should start/stop using TLS
- **TCP/IP stack-based TLS with client identification services for application**
  - Application issues TLS API calls to receive user identity information based on X.509 client certificate
- **Available to any TCP application**
  - CICS Sockets and JES/NJE are primary focus of this support
  - All programming languages supported

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance 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 or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.