z/OS

APAR OA26109 (RACF) and APAR OA26110 (SAF)
RACF support for z/OS Program Signature and Verification

z/OS

APAR OA26109 (RACF) and APAR OA26110 (SAF)
RACF support for z/OS Program Signature and Verification

Contents

General information

This information applies to APAR OA26109 for RACF and APAR OA26110 for SAF.

Part 1. Overview

Chapter 1. Overview

This document details the RACF support to enable program signing and verification. Beginning with z/OS Version 1 Release 10, you can use RACF to enable and control the digital signing and verification of programs. At your option, RACF can enforce that a program be digitally signed and verified before being loaded for execution on your z/OS system. In addition, you can authorize selected users to digitally sign programs that are bound at your installation.

The information within this document has been compiled from the separate manuals which make up the RACF library.

Chapter 2. Software

RACF support for program signature and verification in z/OS V1R10 requires APAR OA26109 (RACF) and APAR OA26110 (SAF). Additionally it requires OA26505 (Binder) and OA24692 (Loader).

Part 2. Information updates

The chapters in this part supplement the following books:

Table 1. z/OS Security Server publication updates

Chapter	Supplements...
Chapter 3, “Security administrator considerations,” on page 9	*z/OS Security Server RACF Security Administrator’s Guide*
Chapter 4, “Command considerations,” on page 29	*z/OS Security Server RACF Command Language Reference*
Chapter 5, “System programmer considerations,” on page 39	*z/OS Security Server RACF System Programmer's Guide*
Chapter 6, “Messages considerations,” on page 41	*z/OS Security Server RACF Messages and Codes*
Chapter 7, “Diagnosis considerations,” on page 47	*z/OS Security Server RACF Diagnosis*
Chapter 8, “Data area considerations,” on page 53	*z/OS Security Server RACF Data Areas*
Chapter 9, “Callable services considerations,” on page 57	*z/OS Security Server RACF Callable Services*
Chapter 10, “Macros and interface considerations,” on page 77	*z/OS Security Server RACF Macros and Interfaces*

Chapter 3. Security administrator considerations

This topic provides information for security administrators about enabling users to digitally sign programs and enabling RACF to verify signed programs. This information supplements *z/OS Security Server RACF Security Administrator's Guide*.

Overview of program signing and verification

You can use RACF® to enable and control the digital signature and verification of programs. At your option, RACF can enforce that a program be digitally signed and verified before being loaded for execution on your z/OS® system. In addition, you can authorize selected users to digitally sign programs that are bound at your installation.

If your installation develops programs, you might choose to enable users to digitally sign the programs you develop. By signing your programs, your customers or users can ensure that they are executing only valid, unchanged versions of the programs they obtain from you. This might be of interest if you are a software vendor.

Guideline: Before you begin enforcing program signing and verification, carefully plan and test procedures to enable your installation to recover from a detected

signature failure. Depending on how you customize the signature verification options for a signed program, an improperly signed module might fail to load. If the module is part of a critical business application, ensure that you have a tested recovery procedure in place to minimize the business impact.

RACF supports program signing and verification only for program objects, which are modules stored as members of a partitioned data set extended (PDSE) library.

Restriction: Program signing and verification are not supported for the following program modules:

- Program objects that are stored in z/OS UNIX® files
- Load modules that are stored as members of a partitioned data set (PDS) library

Terms to know

In this topic, the following terms are synonymously used to refer to a program module stored as a PDSE member:

- program object
- program module
- program
- module

Related information

For programming information about using the SIGN binder option to sign program modules, see *z/OS MVS Program Management: User's Guide and Reference*.

Task roadmap for program signing and signature verification

The following table shows the subtasks and associated instructions for enabling a user to digitally sign a program, and enabling RACF to verify a signed program.

Subtask	Associated instructions (see ...)
Enable a user to sign a program using code-signing certificates that you create using RACF.	"Steps for enabling a user to sign a program using RACF code-signing certificates" on page 13.
Enable a user to sign a program using code-signing certificates that you obtain from an external certificate authority (CA).	"Steps for enabling a user to sign a program using external code-signing certificates" on page 15.
Optionally, audit your installation's signed programs.	"Steps for discovering if signed programs currently execute on your systems (optional)" on page 21.
Prepare RACF to verify signed programs. (This is a one-time setup.)	"Steps for preparing RACF to verify signed programs (one-time setup)" on page 23.
Verify a signed program.	"Steps for verifying a signed program" on page 24.

Enabling a user to sign a program

This topic contains the following subtopics:

Overview of enabling a user to sign a program

An authorized user, or program builder, can sign a program object using the SIGN binder option at the time the program object is bound. Once signed, the program object contains signature information that can be verified at load time.

This overview contains the following topics:

Certificate objects required for program signing

To enable a user to sign a program, you must add certificate objects that meet the following requirements. These certificate objects are added to RACF when you perform the steps in "Task roadmap for enabling a user to sign a program" on page 13.

Requirements:

- Each user must have access to a key ring, called a *program-signing* key ring, that contains all of the following certificate objects:
  - An RSA private key to apply the digital signature.
  - The X.509 certificate, called a *code-signing* certificate, that corresponds to the RSA private key.
  - Each certificate-authority (CA) certificate (up to and including the root CA certificate) in the certificate chain of the code-signing certificate.

    Restrictions:

    - No more than ten certificates are supported in the certificate chain of the code-signing certificate.
    - Do not use a PKCS #11 token as a substitute for the *program-signing* key ring.
- The code-signing certificate and each CA certificate in the chain must be signed using one of the following signature algorithms:
  - `sha256WithRSAEncryption`
  - `sha1WithRSAEncryption`
- The code-signing certificate must have code-signing capability in one of the following ways:
  - Either the certificate has no KeyUsage extension, *or* the certificate has a KeyUsage extension with at least the digitalSignature and nonRepudiation indicators enabled.
- Each CA certificate in the chain must have certificate-signing capability in *both* of the following ways:
  - Either the certificate has no BasicConstraints extension, *or* the certificate has a BasicConstraints extension with the cA indicator enabled.
  - Either the certificate has no KeyUsage extension, *or* the certificate has a KeyUsage extension with at least the keyCertSign indicator enabled.

For examples of using RACDCERT GENCERT command to create certificates that meet these requirements, see "Steps for enabling a user to sign a program using RACF code-signing certificates" on page 13. Otherwise, contact your external certificate authority (CA) and see "Steps for enabling a user to sign a program using external code-signing certificates" on page 15.

For details about using the RACDCERT GENCERT command, see *z/OS Security Server RACF Command Language Reference*.

Details about defining IRR.PROGRAM.SIGNING profiles

When you perform the subtasks in "Task roadmap for enabling a user to sign a program" on page 13, you define APPLDATA information in one or more discrete profiles in the FACILITY class to specify the following:

- The name of the program-signing key ring that contains all certificate objects required for each user who is an authorized program signer.
- The hash algorithm (or message digestion algorithm) that will be used to sign the program.

Format of the profile name: The format of the IRR.PROGRAM.SIGNING profile name is based on how you choose to assign program-signing key rings to users who are authorized program signers.

The first three qualifiers of profile name must be IRR.PROGRAM.SIGNING. The rest of the profile name reflects the available options for assigning key rings to signers.

You can optionally append one or two additional qualifiers to the profile name, as shown in the following list. RACF checks the profiles in the order listed, and uses the first profile found that matches as follows:

1. **IRR.PROGRAM.SIGNING.***group***.***userid*

   This profile assigns the key ring based on the signer's current-connect group and user ID.
2. **IRR.PROGRAM.SIGNING.***userid*

   This profile assigns the key ring based on the signer's user ID.
3. **IRR.PROGRAM.SIGNING.***group*

   This profile assigns the key ring based on the signer's current-connect group.
4. **IRR.PROGRAM.SIGNING**

   This profile assigns the same key ring to all authorized signers.

Rule: No generic characters are allowed in the name of a IRR.PROGRAM.SIGNING profile.

Format of the APPLDATA value: The format of the APPLDATA value in the IRR.PROGRAM.SIGNING profiles is as follows:

[*hash-algorithm*][*owning-userid*]/*key-ring-name*

The variables of the APPLDATA value are defined as follows:

hash-algorithm
: Specifies the message digestion algorithm to be used for program signing. The default value is SHA256. No other values are supported.

owning-userid
: Specifies the user ID that owns the program-signing key ring. If you omit this value, RACF uses the key ring of the authorized program signer.

/*key-ring-name*
: Specifies the fully qualified name of the program-signing key ring. This value must be preceded by the forward slash (/).

Examples:

```
RDEFINE FACILITY IRR.PROGRAM.SIGNING.BUILD.RAMOS
  APPLDATA('BUILDID/BUILD.CODE.SIGNING.KEYRING')
RDEFINE FACILITY IRR.PROGRAM.SIGNING.RAMOS
  APPLDATA('SHA256 RAMOS/RAMOS.CODE.SIGNING.KEYRING')
RDEFINE FACILITY IRR.PROGRAM.SIGNING.PROD
  APPLDATA('/PROD.CODE.SIGNING.KEYRING')
RDEFINE FACILITY IRR.PROGRAM.SIGNING
  APPLDATA('RACFADM/CODE.SIGNING.KEYRING')
```

Rules:

- The only space character allowed in the APPLDATA value is the single space following the *hash-algorithm* value. If *hash-algorithm* is omitted, no space is allowed in the APPLDATA value.
- No extraneous characters are allowed in the APPLDATA value.

RACF does not check the format of the APPLDATA value when you define a IRR.PROGRAM.SIGNING profile. RACF checks the format when a user signs a program and RACF finds a matching IRR.PROGRAM.SIGNING profile.

Task roadmap for enabling a user to sign a program

The following table shows the subtasks and associated instructions for enabling a user to digitally sign a program. Perform one of the following subtasks for each user you want to enable to digitally sign a program. Base your choice of subtask on how you acquire your code-signing certificates.

Subtask	Associated instructions (see ...)
Enable a user to sign a program using code-signing certificates that you create using RACF.	"Steps for enabling a user to sign a program using RACF code-signing certificates."
Enable a user to sign a program using code-signing certificates that you obtain from an external certificate authority (CA).	"Steps for enabling a user to sign a program using external code-signing certificates" on page 15.

Steps for enabling a user to sign a program using RACF code-signing certificates

Before you begin:

- Determine your IRR.PROGRAM.SIGNING profile structure for assigning program-signing key rings to users who are authorized program signers.

  The following steps are based on defining the IRR.PROGRAM.SIGNING.*userid* profile. Therefore, the following examples define a program-signing key ring for each authorized program signer. For details about other options, see "Details about defining IRR.PROGRAM.SIGNING profiles" on page 12.

  Guideline: If you opt instead to define the **IRR.PROGRAM.SIGNING** profile to assign the same key ring to all authorized signers, you might use a profile in the RDATALIB class instead of the FACILITY class to authorize users to access the program-signing ring. A profile in the RDATALIB class allows you to authorize users to a specific key ring. For details, see "RACF Authorization" for `R_datalib` (IRRSDL00 or IRRSDL64) in *z/OS Security Server RACF Callable Services*.
- If you specify the PCICC option (in Step 1 on page 14) to store the private key in ICSF, and the CSFSERV and CSFKEYS classes are active, you might need additional authority in those classes. For information about these resources, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Perform the following steps to enable a user to digitally sign a program using code-signing certificates that you create using RACF.

1. If not already created, create a certificate-authority (CA) certificate that you can use to issue code-signing certificates for users who need to sign programs.

   Guideline: For added security, specify the PCICC option to generate and store the private key in ICSF, if available.

   Example:

```
RACDCERT CERTAUTH GENCERT
  SUBJECTSDN(OU('MyCompany Code Signing CA') O('MyCompany') C('US'))
  SIZE(2048) PCICC WITHLABEL('MyCompany Code Signing CA')
```

---

2. For each user, create a code-signing certificate signed by the CA certificate you created in Step 1.

   Rule: Do not specify the PCICC or ICSF option. The private key of the code-signing certificate must reside in RACF.

   Example:

```
RACDCERT ID(RAMOS) GENCERT
  SUBJECTSDN(CN('Ramos Code Signing Cert') O('MyCompany') C('US'))
  SIZE(1024) WITHLABEL('Ramos Code Signing Cert')
  SIGNWITH(CERTAUTH LABEL('MyCompany Code Signing CA'))
  KEYUSAGE(HANDSHAKE DOCSIGN)
```

---

3. For each user, create a program-signing key ring to hold the certificates you created in Steps 1 and 2.

   Rule: Specify only uppercase characters in the key ring name. This is because you must specify the ring name in the APPLDATA field of the FACILITY profile you create in Step 5.

   Example:

```
RACDCERT ID(RAMOS) ADDRING(RAMOS.CODE.SIGNING.KEYRING)
```

---

4. Add both of the certificates you created in Steps 1 and 2 to the key ring you created in Step 3.

   Rule: The code-signing certificate must be the default certificate in the ring.

   Example:

```
RACDCERT ID(RAMOS) CONNECT(CERTAUTH LABEL('MyCompany Code Signing CA')
  RING(RAMOS.CODE.SIGNING.KEYRING))
RACDCERT ID(RAMOS) CONNECT(ID(RAMOS) LABEL('Ramos Code Signing Cert') DEFAULT
  RING(RAMOS.CODE.SIGNING.KEYRING))
```

---

5. For each user, create a FACILITY class profile that specifies the hash algorithm and the name of the key ring to be used whenever the user digitally signs a program module.

   Example:

```
RDEFINE FACILITY IRR.PROGRAM.SIGNING.RAMOS
  APPLDATA('SHA256 RAMOS/RAMOS.CODE.SIGNING.KEYRING')
```

---

6. Permit each user to access his own key rings, if not already authorized.

   Example:

```
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(RAMOS) ACCESS(READ)
```

7. Activate your profile changes in the FACILITY class, as follows.
   - If the FACILITY class is not already active, activate and RACLIST the FACILITY class.

     Example:

     ```
     SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
     ```
   - If the FACILITY class is already active and RACLISTed, refresh the FACILITY class.

     Example:

     ```
     SETROPTS RACLIST(FACILITY) REFRESH
     ```

You have now enabled a user to digitally sign a program using code-signing certificates that you created using RACF.

Steps for enabling a user to sign a program using external code-signing certificates

Before you begin:

- Obtain or locate the root certificate-authority (CA) certificate of an external CA and store it in a cataloged, variable-byte (VB) MVS data set.
- Determine your IRR.PROGRAM.SIGNING profile structure for assigning program-signing key rings to users who are authorized program signers.

  The following steps are based on defining the IRR.PROGRAM.SIGNING.*userid* profile. Therefore, the following examples define a program-signing key ring for each authorized program signer. For details about other options, see “Details about defining IRR.PROGRAM.SIGNING profiles” on page 12.

  Guideline: If you opt instead to define the **IRR.PROGRAM.SIGNING** profile to assign the same key ring to all authorized signers, you might use a profile in the RDATALIB class instead of the FACILITY class to authorize users to access the program-signing ring. A profile in the RDATALIB class allows you to authorize users to a specific key ring. For details, see “RACF Authorization” for `R_datalib` (IRRSDL00 or IRRSDL64) in *z/OS Security Server RACF Callable Services*.

Perform the following steps to enable a user to digitally sign a program using code-signing certificates that you obtain from an external certificate-authority (CA).

1. If not already done, add the root CA certificate of the external CA to RACF, specifying the name of the data set where it is stored.

   Example:

   ```
   RACDCERT CERTAUTH ADD(CA.CERT.DSN) WITHLABEL('MyCompany Code Signing CA')
   ```

2. For each user, obtain a code-signing certificate from the external CA and add it to RACF. To do so, perform the following sub-steps.
   a. Create a self-signed code-signing certificate (as a placeholder) that will be signed by the external CA.

      Rule: Do not specify the PCICC or ICSF option. The private key of the code-signing certificate must reside in RACF.

      Example:

```
RACDCERT ID(RAMOS) GENCERT
  SUBJECTSDN(CN('Ramos Code Signing Cert') O('MyCompany') C('US'))
  SIZE(1024) WITHLABEL('Ramos Code Signing Cert')
  KEYUSAGE(HANDSHAKE DOCSIGN)
```

b. Create a PKCS #10 certificate request based on the placeholder certificate you created in Step 2a, specifying the name of the MVS data set where the certificate request will be stored.

   Example:

```
RACDCERT ID(RAMOS) GENREQ(LABEL('Ramos Code Signing Cert'))
  DSN(RAMOS.CERT.REQUEST.DSN)
```

c. Send the MVS data set (for example, `RAMOS.CERT.REQUEST.DSN`) containing the stored certificate request to the external CA.

d. Receive the signed certificate returned by the external CA and store it in a cataloged, variable-byte (VB) MVS data set (for example, `RAMOS.CERT.DSN`).

e. Add the new signed certificate to RACF, replacing the placeholder certificate you created in Step 2a.

   Example:

```
RACDCERT ID(RAMOS) ADD(RAMOS.CERT.DSN) WITHLABEL('Ramos Code Signing Cert')
```

---

3. For each user, create a program-signing key ring to hold the external certificates you added in Steps 1 and 2.

   Rule: Specify only uppercase characters in the key ring name. This is because you must specify the ring name in the APPLDATA field of the FACILITY profile you create in Step 5.

   Example:

```
RACDCERT ID(RAMOS) ADDRING(RAMOS.CODE.SIGNING.KEYRING)
```

---

4. Connect both of the certificates you added in Steps 1 and 2 to the key ring you created in Step 3.

   Rule: The code-signing certificate must be the default certificate in the ring.

   Example:

```
RACDCERT ID(RAMOS) CONNECT(CERTAUTH LABEL('MyCompany Code Signing CA')
  RING(RAMOS.CODE.SIGNING.KEYRING))
RACDCERT ID(RAMOS) CONNECT(ID(RAMOS) LABEL('Ramos Code Signing Cert') DEFAULT
  RING(RAMOS.CODE.SIGNING.KEYRING))
```

---

5. For each user, create a FACILITY class profile that specifies the hash algorithm and the name of the key ring to be used whenever the user digitally signs a program module.

   Example:

```
RDEFINE FACILITY IRR.PROGRAM.SIGNING.RAMOS
  APPLDATA('SHA256 RAMOS/RAMOS.CODE.SIGNING.KEYRING')
```

---

6. Permit each user to access his own key rings, if not already authorized.

   Example:

```
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(RAMOS) ACCESS(READ)
```

---

7. Activate your profile changes in the FACILITY class, as follows.

- If the FACILITY class is not already active, activate and RACLIST the FACILITY class.

  Example:

  ```
  SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
  ```

- If the FACILITY class is already active and RACLISTed, refresh the FACILITY class.

  Example:

  ```
  SETROPTS RACLIST(FACILITY) REFRESH
  ```

You have now enabled a user to digitally sign a program using code-signing certificates that you obtained from an external certificate-authority (CA).

Enabling RACF to verify signed programs

This topic contains the following subtopics:

Overview of enabling RACF to verify signed programs

You can enable RACF to verify signed programs by performing some (one-time) setup steps and then specifying the programs you want RACF to verify. Once RACF verifies the authority of a user to execute a controlled program, RACF (optionally) performs signature verification at the time the program object is loaded for execution.

This overview contains the following topics:

Initializing RACF program signature verification

When you perform "Steps for preparing RACF to verify signed programs (one-time setup)" on page 23, you initialize program signature verification. These steps include preparing several certificate objects and general resource profiles, defining the program verification module (IRRPVERS) as a signed program that must be verified when it is loaded, and finally loading and successfully verifying the signature of the IRRPVERS module for the first time.

To verify IRRPVERS, RACF uses the IBM® root CA certificate labeled `STG Code Signing CA` that is supplied with RACF. For a listing of the output from the RACDCERT LIST command for this certificate, see “New supplied certificate from IBM” on page 27.

Certificate objects required for verifying signed programs

To enable RACF to verify signed programs, you must add certificate objects that meet the following requirements. These certificate objects are added to RACF when you perform the steps in “Task roadmap for enabling RACF to verify signed programs” on page 21.

Requirements:

- You must add the TRUST attribute to the code-signing certificate-authority (CA) certificate that is supplied with RACF, so that RACF can use it.
- You must add a key ring, called the *signature-verification* key ring, and connect the RACF code-signing CA certificate and a root CA certificate for *each* trusted program signer. (For a program signed by a user in your installation, this root CA certificate is the CA certificate you added to the user's code-signing key ring in “Enabling a user to sign a program” on page 10.)

  Your installation can have only *one* signature-verification ring. This single ring represents your installation's trust policy for trusted program signers, and must contain the RACF code-signing CA certificate and all root CA certificates required to verify all the signed programs that you want RACF to verify.

  Restriction: Do not use a PKCS #11 token as a substitute for the *signature-verification* key ring.
- Each root CA certificate must be signed using one of the following signature algorithms:
  - `sha256WithRSAEncryption`
  - `sha1WithRSAEncryption`
- Each root CA certificate must have certificate-signing capability in *both* of the following ways:
  - Either the certificate has no BasicConstraints extension, *or* the certificate has a BasicConstraints extension with the cA indicator enabled.
  - Either the certificate has no KeyUsage extension, *or* the certificate has a KeyUsage extension with at least the keyCertSign indicator enabled.

Details about defining the IRR.PROGRAM.SIGNATURE.VERIFICATION profile

When you perform the subtasks in “Task roadmap for enabling RACF to verify signed programs” on page 21, you define APPLDATA information in a discrete profile called IRR.PROGRAM.SIGNATURE.VERIFICATION in the FACILITY class to specify the following:

- The name of the signature-verification key ring that contains all certificate objects required to verify each signed program.

Rule: No generic characters are allowed in the name of the IRR.PROGRAM.SIGNATURE.VERIFICATION profile.

Format of the APPLDATA value: The format of the APPLDATA value in the IRR.PROGRAM.SIGNATURE.VERIFICATION profile is as follows:

owning-userid/*key-ring-name*

The variables of the APPLDATA value are defined as follows:

owning-userid
: Specifies the user ID that owns the signature-verification key ring.

/key-ring-name
: Specifies the fully qualified name of the signature-verification key ring. This value must be preceded by the forward slash (/).

Example:

```
RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
  APPLDATA('RACFADM/CODE.SIGNATURE.VERIFICATION.KEYRING')
```

Rule: No spaces or extraneous characters are allowed in the APPLDATA value.

RACF does not check the format of the APPLDATA value when you define a IRR.PROGRAM.SIGNATURE.VERIFICATION profile. RACF typically checks the format when it verifies the signature of a signed program.

Customizing the SIGVER segment of PROGRAM profiles

The SIGVER segment of a profile in the PROGRAM class contains the signature verification options that apply to one or more programs that are protected by the profile. Customize the fields of the SIGVER segment using the SIGVER operand of the RALTER or RDEFINE command.

When you customize the SIGVER segment of a PROGRAM profile, you can specify options for the following suboperands of the SIGVER operand:

SIGREQUIRED	Specifies whether the program must be digitally signed.
FAILLOAD	Specifies the conditions under which the program should fail to load in the event of a signature verification failure.
SIGAUDIT	Specifies which signature verification events are logged.

For details about customizing the SIGVER segment using the RALTER and RDEFINE commands, see *z/OS Security Server RACF Command Language Reference*.

For examples of customizing the SIGVER segment, see “Steps for verifying a signed program” on page 24.

If you want to delegate authority for customizing the SIGVER segment to auditors or other users who do not have the SPECIAL attribute, see “Delegating the authority for specifying signature verification options.”

Delegating the authority for specifying signature verification options

If you want to delegate the authority for specifying signature verification options to users who do not have the SPECIAL attribute, you must use field-level access checking to authorize UPDATE access to the appropriate fields in the SIGVER segment of PROGRAM class profiles.

Users with the AUDITOR attribute cannot specify auditing options for signature verification unless you authorize them with UPDATE access to the SIGAUDIT field.

The following example authorizes a group called SIGNGRP to specify all signature verification options, and authorizes a second group called AUDGRP to control only the auditing options for signature verification.

Example:

```
SETROPTS CLASSACT(FIELD) RACLIST(FIELD)

RDEFINE FIELD PROGRAM.SIGVER.* UACC(NONE)
PERMIT PROGRAM.SIGVER.* CLASS(FIELD) ID(SIGNGRP) ACCESS(UPDATE)

RDEFINE FIELD PROGRAM.SIGVER.SIGAUDIT UACC(NONE)
PERMIT PROGRAM.SIGVER.SIGAUDIT CLASS(FIELD) ID(SIGNGRP AUDGRP) ACCESS(UPDATE)

SETROPTS RACLIST(FIELD) REFRESH
```

For a complete list of the resource name qualifiers that control each field of the SIGVER segment, see “Field-level access checking for the new SIGVER segment” on page 26.

Discovering if signed programs currently execute on your systems

You can optionally enable SMF logging of signature verification events by performing “Steps for discovering if signed programs currently execute on your systems (optional)” on page 21. By doing so, you can later examine the SMF records using the SMF data unload utility (IRRADU00) to discover if any of your controlled programs are digitally signed and if so, by whom. Once you identify a signer, obtain the signer's root CA in preparation for completing “Steps for verifying a signed program” on page 24.

For information about using the SMF data unload utility (IRRADU00), see *z/OS Security Server RACF Auditor's Guide*.

To enable SMF logging for this purpose, modify one or more PROGRAM profiles to specify the following signature verification options. Using these specific options ensures that no load failures occur due to signature verification failures.

SIGREQUIRED(NO)
: Specifies that no digital signatures are required.

FAILLOAD(NEVER)
: Specifies that no program load should fail due to a signature verification failure.

SIGAUDIT(ALL)
: Specifies that all signature verification events will be logged, regardless of success or failure.

Once you perform “Steps for discovering if signed programs currently execute on your systems (optional)” on page 21, if any controlled program is digitally signed, RACF will attempt to verify the signature upon load. Each signature verification will result in a failure until you complete “Steps for preparing RACF to verify signed programs (one-time setup)” on page 23 and “Steps for verifying a signed program” on page 24. Each signature verification failure will be logged to SMF and related error messages will be issued to the console.

Sample error messages:

```
ICH441I Program signature error 0x00/0x00000074 for program PROGXYZ
        in library LXYZR11.LIBRARY. Load processing continues.
ICH450I The RACF program verification module is not loaded.
        Program signature verification is not available.
```

Task roadmap for enabling RACF to verify signed programs

The following table shows the subtasks and associated instructions for enabling RACF to verify signed programs.

Subtask	Associated instructions (see ...)
Optionally discover if signed programs currently execute on your systems.	"Steps for discovering if signed programs currently execute on your systems (optional)."
Prepare RACF to verify signed programs. (This is a one-time setup.)	"Steps for preparing RACF to verify signed programs (one-time setup)" on page 23.
Verify a signed program.	"Steps for verifying a signed program" on page 24.

Steps for discovering if signed programs currently execute on your systems (optional)

Before you begin:

- For background information about these steps, see "Discovering if signed programs currently execute on your systems" on page 20.
- **Important:** When specifying SIGVER options for a generic program profile (such as the ** profile) for the purpose of discovering signed programs, observe the following guidelines. They are based on the assumption that while you are beginning to evaluate program verification, you have not yet planned for the impact it might have on your installation.

  Guidelines:

  – Set the SIGVER options as shown in the examples in these steps.
  – Avoid specifying SIGREQUIRED(YES) for a generic program profile because it might cause excessive logging and failure messages to the console, based on the SIGAUDIT setting.
  – Avoid specifying FAILLOAD(BADSIGONLY) or FAILLOAD(ANYBAD) for a generic program profile because it might fail critical programs and cause system failure.
- You might need assistance from your system programmer to complete Step 4 on page 22.

Optionally, perform the following steps to enable RACF to perform and log signature verification events for one or more controlled programs, without causing any program loads to fail due to signature verification failures.

1. Modify one or more PROGRAM class profiles that protect controlled programs to enable signature verification and logging without causing any program load to fail due to a signature verification failure.

   Example 1:

   ```
   RALTER PROGRAM MYPROG14
     SIGVER(SIGREQUIRED(NO) FAILLOAD(NEVER) SIGAUDIT(ALL))
   ```

   Important: When a controlled program has an *alias* (an alternate name that can be used to execute it), define both the real name and the alias name. This

might require additional PROGRAM profiles. For an example, see “Protecting programs” in *z/OS Security Server RACF Security Administrator’s Guide*.

If your installation already defined a PROGRAM class profile called ** to control all programs residing in controlled program libraries, you might want to enable signature verification logging for all of these programs by modifying this profile.

Example 2:

```
RALTER PROGRAM **
  SIGVER(SIGREQUIRED(NO) FAILLOAD(NEVER) SIGAUDIT(ALL))
```

Important: For important guidelines about modifying a generic program profile, such as the ** profile shown in Example 2, see “**Before you begin**”.

---

2. Activate your profile changes in the PROGRAM class, as follows.
   - If the PROGRAM class is not already active, activate the PROGRAM class.

     Example:

     ```
     SETROPTS WHEN(PROGRAM)
     ```
   - If the PROGRAM class is already active, refresh the PROGRAM class.

     Example:

     ```
     SETROPTS WHEN(PROGRAM) REFRESH
     ```

---

3. Display the SIGVER segment information for the profiles you modified in Step 1 and review your options.

   Example:

   ```
   RLIST PROGRAM ** SIGVER NORACF
   ```

   Your results will be similar to the following:

   Results:

   ```
   PROGRAM **

   SIGVER INFORMATION
   ------------------
   SIGREQUIRED=NO
   FAILLOAD=NEVER
   SIGAUDIT=ALL
   ```

---

4. (Optional) Ensure that your system programmer enables caching for program signature verification using the virtual lookaside facility (VLF) and restarts VLF. This avoids increasing load times for signed programs. For programming information, see “VLF considerations for program signature verification” on page 39.

You have now enabled RACF to log signature verification events for one or more controlled programs, *without* causing any program loads to fail due to signature verification failures.

Now, each time a signed program loads, RACF logs a signature verification failure to SMF and issues a failure message to the console. These failures continue until you complete “Steps for preparing RACF to verify signed programs (one-time setup)” on page 23 and “Steps for verifying a signed program” on page 24.

After an appropriate time interval during which these programs are loaded, examine the output of the SMF unload utility (IRRADU00) to discover if any controlled programs are digitally signed and if so, by whom. Once you identify the signer, obtain the signer's root CA in preparation for completing “Steps for verifying a signed program” on page 24.

When your analysis is complete, proceed to “Steps for preparing RACF to verify signed programs (one-time setup).”

Steps for preparing RACF to verify signed programs (one-time setup)

By performing these steps, you prepare RACF to verify signatures. However, RACF does not begin verifying the signatures of your programs until you complete “Steps for verifying a signed program” on page 24.

Before you begin: You will need assistance from your system programmer to complete Step 8.

Perform the following steps to prepare RACF to verify signed programs. Complete these steps one time only.

1. Create a key ring for your installation to use for signature verification. Specify the ring name of your choice.

   Example:

   ```
   RACDCERT ID(RACFADM) ADDRING(CODE.SIGNATURE.VERIFICATION.KEYRING)
   ```

   Rule: Specify only uppercase characters in the key ring name. This is because you must specify the ring name in the APPLDATA field of the FACILITY profile you create in Step 4.

   Guideline: Do not skip this step so that you can use the virtual CERTAUTH key ring. For best performance, define your signature verification ring by issuing a RACDCERT ADDRING command.

---

2. Add the TRUST attribute to the code-signing CA certificate that is supplied with RACF.

   Example:

   ```
   RACDCERT CERTAUTH ALTER(LABEL('STG Code Signing CA')) TRUST
   ```

---

3. Add the code-signing CA certificate that is supplied with RACF to the key ring you created in Step 1.

   Example:

   ```
   RACDCERT ID(RACFADM) CONNECT(CERTAUTH LABEL('STG Code Signing CA')
     RING(CODE.SIGNATURE.VERIFICATION.KEYRING))
   ```

---

4. Create a FACILITY class profile that specifies the name of the key ring you created in Step 1.

   Example:

   ```
   RDEFINE FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION
     APPLDATA('RACFADM/CODE.SIGNATURE.VERIFICATION.KEYRING')
   ```

---

5. Activate your profile changes in the FACILITY class, as follows.

- If the FACILITY class is not already active, activate and RACLIST the FACILITY class.

  Example:

  ```
  SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
  ```

- If the FACILITY class is already active and RACLISTed, refresh the FACILITY class.

  Example:

  ```
  SETROPTS RACLIST(FACILITY) REFRESH
  ```

---

6. Create a PROGRAM class profile that protects the program verification module called IRRPVERS and specifies its signature verification options.

   Examples:

   ```
   RDEFINE PROGRAM IRRPVERS ADDMEM('SYS1.SIEALNKE'//NOPADCHK) UACC(READ)
     SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
   ```

---

7. Activate your profile changes in the PROGRAM class, as follows.
   - If the PROGRAM class is not already active, activate the PROGRAM class.

     Example:

     ```
     SETROPTS WHEN(PROGRAM)
     ```

   - If the PROGRAM class is already active, refresh the PROGRAM class.

     Example:

     ```
     SETROPTS WHEN(PROGRAM) REFRESH
     ```

---

8. Contact your system programmer to complete this step.
   a. Notify your system programmer to initialize program signature verification by running the IRRVERLD program. The IRRVERLD program loads and verifies the program verification module (IRRPVERS) and must be run on all systems in a sysplex. For programming information, see “Initializing RACF verification of signed programs (IRRVERLD)” on page 39.
   b. Check with your system programmer to ensure that IRRVERLD successfully completed. If it did not, work with your system programmer to resolve error messages and then rerun.
   c. (Optional) Ensure that your system programmer enables caching for program signature verification using the virtual lookaside facility (VLF) and restarts VLF. This avoids increasing load times for signed programs. For programming information, see “VLF considerations for program signature verification” on page 39.

When the IRRVERLD program successfully executes, you have completed the one-time setup to prepare RACF to verify signed programs. To begin verifying one of your own signed programs, proceed to “Steps for verifying a signed program.”

Steps for verifying a signed program

Before you begin:

- Do not perform these steps until you complete “Steps for preparing RACF to verify signed programs (one-time setup)” on page 23.
- For each signed program that you want RACF to verify, obtain or locate the root certificate-authority (CA) certificate of each code signer.

– If the program was acquired from a software vendor, review the software documentation for information about obtaining the vendor's root CA certificate and adding it to RACF. Once you obtain the root CA certificate, store it in a cataloged, variable-byte (VB) MVS data set.
– If the program was signed by your installation (in Step 1 of "Steps for enabling a user to sign a program using RACF code-signing certificates" on page 13) or signed by an external CA (in Step 1 of "Steps for enabling a user to sign a program using external code-signing certificates" on page 15), locate the label name of root CA certificate in the RACF database.

  To list all CA certificates in the RACF database, issue the RACDCERT LIST CERTAUTH command.

Perform the following steps for *each* signed program you want RACF to verify.

1. Add the root CA certificate of the code signer to RACF as a trusted CA.

   Skip this step if you created the root CA of the code signer (in Step 1 of "Steps for enabling a user to sign a program using RACF code-signing certificates" on page 13), or if you obtained the root CA of the code signer from an external CA and added it to RACF (in Step 1 of "Steps for enabling a user to sign a program using external code-signing certificates" on page 15).

   a. If you obtained the root CA certificate of the code signer from a software vendor, add it to RACF, specifying the name of the data set where it is stored.

      Example:

      ```
      RACDCERT CERTAUTH ADD(VENDOR.CA.CERT.DSN)
        WITHLABEL('Vendor Code Signing CA')
        TRUST
      ```

   b. If the vendor's root CA certificate is already added to RACF, add the TRUST attribute if it is not already trusted.

      Example:

      ```
      RACDCERT CERTAUTH ALTER(LABEL('Vendor Code Signing CA')) TRUST
      ```

---

2. Add the root CA certificate to the key ring that your installation uses for signature verification. This is the ring you created in Step 1 of "Steps for preparing RACF to verify signed programs (one-time setup)" on page 23.

   Examples:

   ```
   RACDCERT ID(RACFADM) CONNECT(CERTAUTH LABEL('Vendor Code Signing CA')
     RING(CODE.SIGNATURE.VERIFICATION.KEYRING))

   -or-

   RACDCERT ID(RACFADM) CONNECT(CERTAUTH LABEL('MyCompany Code Signing CA')
     RING(CODE.SIGNATURE.VERIFICATION.KEYRING))
   ```

---

3. Create or modify the PROGRAM class profile that protects the signed program and specify the signature verification options.

   Important: If the program you want to verify is already protected by a generic program profile, such as the ** profile, create a new program profile to protect program. Do not specify the SIGVER options shown in the following examples for the ** profile because this might fail critical programs and lead to system failure. If several similarly named programs must be verified using the same

SIGVER options, you might choose to create a generic profile such as ABC*. If you do, ensure that no other programs are unintentionally verified based on their similar program names.

The following examples specify that the load of program MYPROG14 should fail if the signature cannot be verified for any reason and that only failures should be logged.

Examples:

```
RDEFINE PROGRAM MYPROG14 ADDMEM('SYS1.TEST.LOADDLL'//NOPADCHK) UACC(READ)
  SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))

-or-

RALTER PROGRAM MYPROG14
  SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

Important: When a controlled program has an *alias* (an alternate name that can be used to execute it), define both the real name and the alias name. This might require additional PROGRAM profiles. For an example, see "Protecting programs" in *z/OS Security Server RACF Security Administrator's Guide*.

If you want to delegate authority to perform this step to a user who does not have the SPECIAL attribute, see "Delegating the authority for specifying signature verification options" on page 19.

---

4. Activate your profile changes in the PROGRAM class.

   Example:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```

You have now enabled RACF to verify a signed program. If you specified the signature verification options shown in the example in Step 3 on page 25, the program will fail to load if RACF cannot verify the signature for any reason. If the program is part of a critical business application, be prepared to invoke a recovery procedure to minimize the business impact.

Field-level access checking for the new SIGVER segment

You can use RACF to control which users can access data in RACF profiles at the field level through *field-level access checking*. To do this, you create profiles in the FIELD class and permit users to the profiles.

You need not use field-level access checking to authorize READ access for users with the AUDITOR attribute. Users with the AUDITOR attribute are authorized to list all fields for all segments, other than the base segment, of any RACF profile.

To control fields in the SIGVER segment of general resource profiles in the PROGRAM class, define profiles in the FIELD class using profile names that following the form:

```
PROGRAM.SIGVER.field-name
```

where *field-name* specifies the name of the segment field that corresponds to the command operand controlling that field, as listed in the following table:

To control the use of this SIGVER operand:	Use this field name in the FIELD profile name:
SIGREQUIRED	SIGREQD

To control the use of this SIGVER operand:	Use this field name in the FIELD profile name:
FAILLOAD	FAILLOAD
SIGAUDIT	SIGAUDIT (See **Note**.)

Note: The SIGAUDIT field controls the audit policy related to digital signature verification of programs. Users with the AUDITOR attribute can list the SIGAUDIT field but they cannot not update it unless they have UPDATE authority through field-level access checking.

Updated sample for using the RACF database unload utility (IRRDBU00) with DB2

You can use the DB2® load utility or its equivalent to process the records produced by the database unload utility. Member RACDBULD of SYS1.SAMPLIB contains sample control statements for the DB2 load utility that map the output from IRRDBU00 and create a DB2 table for each IRRDBU00 record type.

Member RACDBULD is updated to create the following DB2 table in support of the following new IRRDBU00 record type:

Record type	Record name	DB2 table name
05F0	General Resource SIGVER Data	GENR_GRSIG_DATA

New supplied certificate from IBM

This topic contains a complete listing for the new certificate-authority (CA) certificate supplied with the applicable PTF for APAR OA26109.

For general information about supplied certificates, see “Supplied digital certificates” in *z/OS Security Server RACF Security Administrator’s Guide*.

The following listing was created using the RACDCERT LIST command, which is the preferred method for listing certificate information.

IBM STG Code-Signing Certificate Authority

```
Label: STG Code Signing CA
 Certificate ID: 2QiJmZmDhZmjgeLjx0DDloSFQOKJh5WJlYdAw8FA
 Status: NOTRUST
 Start Date: 2008/07/01 00:00:00
 End Date:   2028/06/30 23:59:59
 Serial Number:
      >00<
 Issuer's Name:
      >CN=STG Code Signing CA.OU=IBM Code Signing.O=IBM Corporation.C=US<
 Subject's Name:
      >CN=STG Code Signing CA.OU=IBM Code Signing.O=IBM Corporation.C=US<
 Key Usage: CERTSIGN
 Private Key Type: None
 Ring Associations:
 *** No rings associated ***
```

Note: Start and end times are listed in Greenwich Mean Time (GMT).

Chapter 4. Command considerations

The following commands are updated to support program signing and verification:

RALTER (Alter general resource profile)

The following updates to the syntax and parameter descriptions of this command are made to support program signing and verification.

Syntax

```
[subsystem-prefix]{RALTER | RALT}
        [ SIGVER(
          [ SIGREQUIRED( YES | NO ) | NOSIGREQUIRED ]
          [ FAILLOAD( ANYBAD | BADSIGONLY | NEVER ) | NOFAILLOAD ]
          [ SIGAUDIT( ALL | SUCCESS | ANYBAD | BADSIGONLY | NONE )
            | NOSIGAUDIT ]
          )
         | NOSIGVER ]
```

Parameters

SIGVER | NOSIGVER

SIGVER

Specifies the options for verifying the signatures of programs that are protected by this general resource profile.

Rule: Specify SIGVER only for profiles in the PROGRAM class. Any options specified with the SIGVER operand are ignored for profiles in a class other than the PROGRAM class.

Restriction: Digital signature verification is supported only for program objects that are stored as members of a partitioned data set extended (PDSE) library. Digital signature verification is *not* supported for programs that are stored as members of a partitioned data set (PDS) library.

Any options specified with the SIGVER operand are ignored for unsupported programs.

Note: Regardless of the SIGREQUIRED setting, specifying FAILLOAD(NEVER) and SIGAUDIT(NONE) is equivalent to having no SIGVER segment.

For detailed information, see Chapter 3, “Security administrator considerations,” on page 9.

SIGREQUIRED | NOSIGREQUIRED

SIGREQUIRED

Specifies whether programs that are protected by this profile must be digitally signed.

YES

Specifies that programs must be digitally signed.

When you specify SIGREQUIRED(YES), the following conditions apply to any program that is protected by this general resource profile:

- If the program has a digital signature:
  - Signature verification processing occurs.
  - The program continues to load according to the FAILLOAD setting.

- Logging occurs according to the SIGAUDIT setting.

- If the program has no digital signature:
  - Signature verification processing occurs, resulting in a signature verification failure.
  - The program continues to load according to the FAILLOAD setting.
  - Logging occurs according to the SIGAUDIT setting.

NO
Specifies that programs need not be digitally signed.

When you specify SIGREQUIRED(NO), the following conditions apply to any program that is protected by this general resource profile:

- If the program has a digital signature:
  - Signature verification processing occurs.
  - The program continues to load according to the FAILLOAD setting.
  - Logging occurs according to the SIGAUDIT options.
- If the program has no digital signature:
  - No signature verification occurs.
  - The program continues to load. The FAILLOAD setting is ignored.
  - No logging occurs. The SIGAUDIT setting is ignored.

NOSIGREQUIRED
Resets the SIGREQUIRED value to NO.

FAILLOAD | NOFAILLOAD

FAILLOAD
Specifies the conditions under which the program fails to load in the event that a signature verification failure occurs.

ANYBAD
Specifies that the program fails to load when a signature verification failure occurs, regardless of the cause. Such failures include those resulting from an incorrect signature, or an error establishing the trust of the signer. This setting includes failures related to administrative errors, such as a missing or incorrectly defined key ring.

The ANYBAD setting includes the failures covered by the BADSIGONLY setting, and also includes errors establishing the trust of the signer.

BADSIGONLY
Specifies that the program fails to load only when the signature verification failure is caused by an incorrect digital signature. Such failures include only those resulting from a signature that fails verification or a signature structure that is missing or improperly formatted.

In contrast to ANYBAD, the BADSIGONLY setting does not cause a program to fail to load when the program has a valid signature originating from an untrusted signer.

NEVER
Specifies that the program never fails to load when a signature verification failure is detected.

NOFAILLOAD
Resets the FAILLOAD value to NEVER.

SIGAUDIT | NOSIGAUDIT

SIGAUDIT
Specifies which signature verification events are logged. Messages are issued to the console only for signature verification failures that are logged.

ALL
Logs all signature verifications, whether successful or not.

SUCCESS
Logs only signature verification successes. In other words, the digital signature is valid and the root CA certificate is trusted.

ANYBAD
Logs all signature verification failures, regardless of the cause of the failure. Such failures include those resulting from an incorrect signature, or an error establishing the trust of the signer. This setting includes failures related to administrative errors, such as a missing or incorrectly defined key ring.

The ANYBAD setting logs the failures covered by the BADSIGONLY setting, and also logs errors encountered when establishing the trust of the signer.

BADSIGONLY
Logs only signature verification failures caused by an incorrect digital signature. Such failures include only those resulting from a signature that fails verification or a signature structure that is missing or improperly formatted.

In contrast to ANYBAD, the BADSIGONLY setting does not log a signature verification failure when the program has a valid signature originating from an untrusted signer.

NONE
Logs no digital signature verification events.

NOSIGAUDIT
Resets the SIGAUDIT value to NONE.

NOSIGVER
Deletes the SIGVER segment.

Example

Operation User SECADM wants to update the signature verification options for a controlled program called MYPROG14 program to specify that it must now be digitally signed before it can be loaded, that the program should fail to load if its digital signature cannot be verified for any reason, and that logging of signature verification events should occur for only failures.

Known The user has the SPECIAL attribute. The MYPROG14 program is a program object that resides in a partitioned data set extended (PDSE) library.

Command
```
RALTER PROGRAM MYPROG14
 SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

Defaults None.

RDEFINE (Define general resource profile)

The following updates to the syntax and parameter descriptions of this command are made to support program signing and verification.

Syntax

```
[subsystem-prefix]{RDEFINE | RDEF}
        [ SIGVER(
          [ SIGREQUIRED( YES | NO ) ]
          [ FAILLOAD( ANYBAD | BADSIGONLY | NEVER ) ]
          [ SIGAUDIT( ALL | SUCCESS | ANYBAD | BADSIGONLY | NONE ) ]
          ) ]
```

Parameters

SIGVER

Specifies the options for verifying the signatures of programs that are protected by this general resource profile.

Rule: Specify SIGVER only for profiles in the PROGRAM class. Any options specified with the SIGVER operand are ignored for profiles in a class other than the PROGRAM class.

Restriction: Digital signature verification is supported only for program objects that are stored as members of a partitioned data set extended (PDSE) library. Digital signature verification is *not* supported for programs that are stored as members of a partitioned data set (PDS) library.

Any options specified with the SIGVER operand are ignored for unsupported programs.

Note: Regardless of the SIGREQUIRED setting, specifying FAILLOAD(NEVER) and SIGAUDIT(NONE) is equivalent to having no SIGVER segment.

For detailed information, see Chapter 3, “Security administrator considerations,” on page 9.

SIGREQUIRED

Specifies whether programs that are protected by this profile must be digitally signed.

YES

Specifies that programs must be digitally signed.

When you specify SIGREQUIRED(YES), the following conditions apply to any program that is protected by this general resource profile:

- If the program has a digital signature:
  - Signature verification processing occurs.
  - The program continues to load according to the FAILLOAD setting.
  - Logging occurs according to the SIGAUDIT setting.
- If the program has no digital signature:
  - Signature verification processing occurs, resulting in a signature verification failure.
  - The program continues to load according to the FAILLOAD setting.
  - Logging occurs according to the SIGAUDIT setting.

NO
Specifies that programs need not be digitally signed.

When you specify SIGREQUIRED(NO), the following conditions apply to any program that is protected by this general resource profile:

- If the program has a digital signature:
  - Signature verification processing occurs.
  - The program continues to load according to the FAILLOAD setting.
  - Logging occurs according to the SIGAUDIT options.
- If the program has no digital signature:
  - No signature verification occurs.
  - The program continues to load. The FAILLOAD setting is ignored.
  - No logging occurs. The SIGAUDIT setting is ignored.

If SIGREQUIRED is not specified, SIGREQUIRED(NO) is the default value.

FAILLOAD
Specifies the conditions under which the program fails to load in the event that a signature verification failure occurs.

ANYBAD
Specifies that the program fails to load when a signature verification failure occurs, regardless of the cause. Such failures include those resulting from an incorrect signature, or an error establishing the trust of the signer. This setting includes failures related to administrative errors, such as a missing or incorrectly defined key ring.

The ANYBAD setting includes the failures covered by the BADSIGONLY setting, and also includes errors establishing the trust of the signer.

BADSIGONLY
Specifies that the program fails to load only when the signature verification failure is caused by an incorrect digital signature. Such failures include only those resulting from a signature that fails verification or a signature structure that is missing or improperly formatted.

In contrast to ANYBAD, the BADSIGONLY setting does not cause a program to fail to load when the program has a valid signature originating from an untrusted signer.

NEVER
Specifies that the program never fails to load when a signature verification failure is detected.

If FAILLOAD is not specified, FAILLOAD(NEVER) is the default value.

SIGAUDIT
Specifies which signature verification events are logged. Messages are issued to the console only for signature verification failures that are logged.

ALL
Logs all signature verifications, whether successful or not.

SUCCESS
Logs only signature verification successes. In other words, the digital signature is valid and the root CA certificate is trusted.

ANYBAD
Logs all signature verification failures, regardless of the cause of the

failure. Such failures include those resulting from an incorrect signature, or an error establishing the trust of the signer. This setting includes failures related to administrative errors, such as a missing or incorrectly defined key ring.

The ANYBAD setting logs the failures covered by the BADSIGONLY setting, and also logs errors encountered when establishing the trust of the signer.

BADSIGONLY
Logs only signature verification failures caused by an incorrect digital signature. Such failures include only those resulting from a signature that fails verification or a signature structure that is missing or improperly formatted.

In contrast to ANYBAD, the BADSIGONLY setting does not log a signature verification failure when the program has a valid signature originating from an untrusted signer.

NONE
Logs no digital signature verification events.

If SIGAUDIT is not specified, SIGAUDIT(NONE) is the default value.

Example

Operation User SECADM wants to control the XYZLIB64 program and specify that it must be digitally signed before it can be loaded, that the program should fail to load if its digital signature cannot be verified for any reason, and that logging of signature verification events should occur for only failures. The XYZLIB64 program does not require program-accessed data set checking.

Known The user has the SPECIAL attribute. The XYZLIB64 program is a program object that resides in a partitioned data set extended (PDSE) library named SYS1.XYZ.LOADDLL.

Command
```
RDEFINE PROGRAM XYZLIB64 UACC(READ)
 ADDMEM('SYS1.XYZ.LOADDLL'//NOPADCHK)
 SIGVER(SIGREQUIRED(YES) FAILLOAD(ANYBAD) SIGAUDIT(ANYBAD))
```

Defaults AUDIT(FAILURES(READ)) OWNER(SECADM) LEVEL(0)

RLIST (List general resource profile)

The following updates to the syntax and parameter descriptions of this command are made to support program signing and verification.

Syntax

```
[subsystem-prefix]{RLIST | RL}
        [ SIGVER ]
```

Parameters

SIGVER
Specifies that the contents of the SIGVER segment are to be listed for profiles in the PROGRAM class.

Example

Operation The security administrator wants to list the settings related to digital signature verification for the program called XYZLIB64.

Known The security administrator has the SPECIAL attribute.

Command `RLIST PROGRAM XYZLIB64 SIGVER NORACF`

Defaults None.

Output See Figure 1.

```
RLIST PROGRAM XYZLIB64 SIGVER NORACF
CLASS      NAME
-----      ----
PROGRAM    XYZLIB64

SIGVER INFORMATION
------------------
SIGREQUIRED = YES
FAILLOAD = ANYBAD
SIGAUDIT = ANYBAD
```

Figure 1. Output for RLIST of the SIGVER segment

Chapter 5. System programmer considerations

Program signing and signature verification

RACF supports signing and verification of program objects. This means that IBM or a vendor can ship program objects containing a digital signature (as well as the digital certificate chain for the user who performed the program bind). An installation can then choose to verify the integrity of these program objects (for example, the System SSL modules) when they are loaded into virtual storage.

Installations that choose to exploit signature verification will experience some performance overhead when the signature of a signed program object is verified before being loaded into memory. However, most of this overhead is from validating the certificate chain that was used during the signing process and can be lessened using the virtual lookaside facility (VLF) to cache certificates after they are validated. Most of the overhead from validating a certificate chain is then incurred only once, the first time a program object from a particular signer is loaded. Subsequent calls to load any program object from the same signer will be able to avoid the performance overhead.

For information about enabling VLF caching, see "VLF considerations for program signature verification." For information about initializing program signature verification, see "Initializing RACF verification of signed programs (IRRVERLD)." For implementation details, see ″Enabling RACF to verify signed programs″ in *z/OS Security Server RACF Security Administrator's Guide*

VLF considerations for program signature verification

RACF can use VLF to cache signature verification data in order to improve the performance of signature verification of signed program objects. This in turn can improve the load time of signed program objects. This is a consideration only for installations that choose to exploit signature verification.

For background information, see "Program signing and signature verification." For implementation details, see ″Enabling RACF to verify signed programs″ in *z/OS Security Server RACF Security Administrator's Guide*

Dependencies

In order for this function to be available, VLF must be active and the active COFVLF*xx* member of SYS1.PARMLIB must include statements defining the VLF classes used for signature verification data. Update the COFVLFxx member of SYS1.PARMLIB as follows:

```
CLASS NAME(IRRSPS0)         /* Signature Verification Data in Memory */
EMAJ(PERFCACHE)             /* Major name = PERFCACHE                */
```

Initializing RACF verification of signed programs (IRRVERLD)

The IRRVERLD program initializes RACF verification of signed programs by loading and verifying the RACF program verification module (IRRPVERS). IRRPVERS is a signed program that performs verification of signed programs, and must be loaded and verified before RACF is able to verify other signed programs. Before you run IRRVERLD, the security administrator must configure RACF to verify signed programs, and must define IRRPVERS as a signed program that must be verified when it is loaded. See ″Enabling RACF to verify signed programs″ in *z/OS Security*

Server RACF Security Administrator's Guide for more information about implementing program signature verification.

Important:: Run the IRRVERLD program on each system in a sysplex.

IRRVERLD has no parameters and is run using JCL. For example:

```
//IRRVERLD    JOB
//IRRVERLD    EXEC   PGM=IRRVERLD
```

If IRRVERLD does not successfully complete, work with your RACF security administrator to resolve error messages and rerun IRRVERLD. After IRRVERLD successfully completes, the IRRPVERS module is active and RACF is enabled to verify signed programs. The IRRPVERS module remains active until the next IPL. At IPL time, RACF initialization automatically reloads and verifies IRRPVERS unless the RACF configuration for program signing is disabled between IPLs. You can rerun IRRVERLD at any time to check the status of the IRRPVERS module.

IRRVERLD return codes

The IRRVERLD program sets the following return codes:

Hex	(Decimal)	Meaning
0	(0)	IRRVERLD completed successfully. The IRRPVERS module was loaded and verified. RACF program verification is active.
4	(4)	The IRRPVERS module was previously loaded and verified. RACF program verification is already active.
8	(8)	An error occurred during the load and verification of IRRPVERS. RACF program verification is not active.
10	(16)	A severe error occurred.

RACF virtual storage requirements

The following row has been added to the RACF estimated storage usage table:

Storage subpool	Usage	How to estimate size
ECSA	RACF program verification module (IRRPVERS)	500 000

Chapter 6. Messages considerations

ICH440I **Program signature error *retcode/rsncode* for program *program-name* in library *library-name*. The program was not loaded.**

Explanation: RACF detected an error with the cryptographic signature of the identified program.

A subsequent message is issued that provides more information about this error.

Note: This message is only issued if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the condition being audited.

System action: The load fails.

RACF Security Administrator Response:

Refer to the subsequent message for more information. Additional information is provided in the return and reason code displayed in the message. These codes are from the VERFINAL function of the R_PgmSignVer (IRRSPS00) callable service and descriptions for these codes can be found in *z/OS Security Server RACF Callable Services*. A reason code greater than or equal to 100 might indicate a setup problem with the verification key ring, which can be fixed by the security administrator. Other reason codes must be reported to the provider of the failing module.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH441I **Program signature error *retcode/rsncode* for program *program-name* in library *library-name*. Load processing continues.**

Explanation: RACF detected an error with the cryptographic signature of the identified program. The FAILLOAD setting in the SIGVER segment of the PROGRAM class profile allows the load to continue.

A subsequent message is issued that provides more information about this error.

Note: This message is only issued if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the condition being audited.

System action: Load processing continues.

RACF Security Administrator Response:

Refer to the subsequent message for more information. Additional information is provided in the return and reason code displayed in the message. These codes are from the VERFINAL function of the R_PgmSignVer (IRRSPS00) callable service and descriptions for these codes can be found in *z/OS Security Server RACF Callable Services*. A reason code greater than or equal to 100 might indicate a setup problem with the verification key ring, which can be fixed by the security administrator. Other reason codes must be reported to the provider of the failing module.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH442I **The digital signature appears to be valid but the root signer is not trusted.**

Explanation: The digital signature in the program is correct, but the root CA certificate of the certificate chain contained with the signature has not been designated as trusted, or the setup configuration is preventing RACF from being able to determine the trusted status. This message can result from the any of the following conditions:

- The IRR.PROGRAM.SIGNATURE.VERIFICATION profile is not defined in the FACILITY class.
- The APPLDATA field of the IRR.PROGRAM.SIGNATURE.VERIFICATION profile is missing or incorrect. (The APPLDATA is used to identify the key ring that contains the trusted root certificates.)
- The APPLDATA identifies a key ring that does not exist.
- The root CA certificate of the certificate chain contained with the signature has not been added to the specified key ring, or has been added with the NOTRUST flag.
- The root CA certificate in the certificate chain contained with the signature has the NOTRUST flag on.

Notes:

1. The program name is identified in message ICH440I or ICH441I. One of these messages precedes this message.
2. This message is only issued if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the specific condition being audited.
3. There might also be diagnostic information in a LOGREC record.

System action: If message ICH441I precedes this message, the program load continues. If message ICH440I precedes this message, the load fails.

RACF Security Administrator Response:

If you trust the certificate chain associated with the signed program, you must place the root CA certificate into the appropriate key ring.

You can temporarily bypass this error in any of the following ways:

- If you have specified FAILLOAD(ANYBAD) in the SIGVER segment of the RACF PROGRAM class profile that protects this program, then specify FAILLOAD(BADSIGONLY). This change enables the program to continue.
- Specify SIGAUDIT(BADSIGONLY) or NOSIGAUDIT to stop this message being issued for this program again.
- Remove the SIGVER segment from the PROGRAM class profile.
- Delete the PROGRAM class profile if it is not being used to restrict or audit access to the program.

Note: The current security policy has flagged this condition as an error. Bypassing the error prevents this message from being issued when the program is loaded, but reduces system security and does not resolve the problem. Once you have added the root CA certificate into the verification key ring, revisit your FAILLOAD and SIGAUDIT settings.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH443I The digital signature is not valid.

Explanation: The digital signature in the program does not match the hash of the program computed by RACF. This message indicates that the program has been either modified since it was created, or that it was not properly signed.

Notes:

1. The program name is identified in message ICH440I or ICH441I. One of these messages precedes this message.
2. This message is only issued if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the specific condition being audited.
3. There might also be diagnostic information in a LOGREC record.

System action: If message ICH441I precedes this message, the program load continues. If message ICH440I precedes this message, the load fails.

RACF Security Administrator Response:

Save the current program, and then replace the program with a copy from the installation media. If replacing the program with a copy from the installation media does not resolve the problem, replace the program with a copy from program provider.

You can temporarily bypass this error in any of the following ways:

- If the load fails, change the SIGVER segment of the RACF PROGRAM class profile that protects this program to specify FAILLOAD(NEVER). This change enables the program to continue.
- Specify SIGAUDIT(NONE) or NOSIGAUDIT to stop this message being issued for this program again.
- Remove the SIGVER segment from the PROGRAM class profile.
- Delete the PROGRAM class profile if it is not being used to restrict or audit access to the program.

Note: The current security policy has flagged this condition as an error. Bypassing the error prevents this message from being issued when the program is loaded, but reduces system security and does not resolve the problem. Once you have resolved the problem, revisit your FAILLOAD and SIGAUDIT settings.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH444I The program contains an incorrect certificate chain. Reason code X*'rsncode'*.

Explanation: When a program is signed during the bind process, the program object contains a digital signature and the digital certificate chain for the user who performed the program bind. This message indicates that the digital certificate chain is incorrect.

The reason code in this message indicates the reason for the failure. This reason code originates from the R_PgmSignVer callable service (IRRSPS00), which is called to verify the signature and certificate chain when the program is loaded. In *z/OS Security Server RACF Callable Services*, there is a specific set of return and reason codes documented for function code X'0007' (VERFINAL). The relevant reason codes are documented under SAF return code 8 and RACF return code 16.

Notes:

1. The program name is identified in message ICH440I or ICH441I. One of these messages precedes this message.
2. This message is only issued if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the specific condition being audited.
3. There might also be diagnostic information in a LOGREC record.

System action: If message ICH441I precedes this message, the program load continues. If message ICH440I precedes this message, the load fails.

RACF Security Administrator Response:

Inform the provider of the program with the information in this message. Either the program was not built

correctly, or it has been modified. A new copy of the module with the correct signature and certificate chain is required.

You can temporarily bypass this error in any of the following ways:

- If the load fails, change the SIGVER segment of the RACF PROGRAM class profile that protects this program to specify FAILLOAD(NEVER). This change enables the program to continue.
- Specify SIGAUDIT(NONE) or NOSIGAUDIT to stop this message being issued for this program again.
- Remove the SIGVER segment from the PROGRAM class profile.
- Delete the PROGRAM class profile if it is not being used to restrict or audit access to the program.

Note: The current security policy has flagged this condition as an error. Bypassing the error prevents this message from being issued when the program is loaded, but reduces system security and does not resolve the problem. Once you have resolved the problem, revisit your FAILLOAD and SIGAUDIT settings.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH445I A digital signature is required but the program is not signed.

Explanation: The SIGVER segment of the RACF PROGRAM class profile protecting this program specifies SIGREQUIRED(YES). This indicates that this program requires a signature, but the program is not digitally signed.

Notes:

1. The program name is identified in message ICH440I or ICH441I. One of these messages precedes this message.
2. This message is issued only if the audit specifications, in the SIGVER segment of the PROGRAM profile, result in the specific condition being audited.

System action: If an ICH441I message precedes this message, then the program load continues. If an ICH440I message precedes this message, the load fails.

RACF Security Administrator Response:

Contact the provider of the program and request a digitally signed version of this program.

You can temporarily bypass this error in any of the following ways:

- If the load fails, change the SIGVER segment of the RACF PROGRAM class profile that protects this program to specify SIGREQUIRED(NO). This change enables the program to continue.
- Specify SIGAUDIT(NONE) or NOSIGAUDIT to stop this message being issued for this program again.
- Remove the SIGVER segment from the PROGRAM class profile.
- Delete the PROGRAM class profile if it is not being used to restrict or audit access to the program.

Note: The current security policy has flagged this condition as an error. Bypassing the error prevents this message from being issued when the program is loaded, but reduces system security and does not resolve the problem. Once you have resolved the problem, revisit your FAILLOAD and SIGAUDIT settings.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH446I The digital signature has been removed from the program.

Explanation: The PSDE directory indicates that the program member is digitally signed, but the program does not contain a digital signature. This message indicates that the program has been modified since it was created.

Notes:

1. The program name is identified in the ICH440I message, which precedes this one.
2. This message is issued only if the audit specifications in the SIGVER segment of the PROGRAM profile result in the specific condition being audited.

System action: The load fails.

RACF Security Administrator Response:

Save the current program, and then replace the program with a copy from the installation media. If replacing the program with a copy from the installation media does not resolve the problem, replace the program with a copy from program provider.

You can temporarily bypass this error in any of the following ways:

- Remove the SIGVER segment from the PROGRAM class profile.
- Delete the PROGRAM class profile if it is not being used to restrict or audit access to the program.

Note: The current security policy has flagged this condition as an error. Bypassing the error allows the load to continue and prevents this message from being issued when the program is loaded, but reduces system security and does not resolve the problem. Once you have resolved the problem, revisit your SIGVER segment settings.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH447I RACF was unable to load and verify the program verification module.

Explanation: An error occurred while RACF was attempting to load and verify the program verification module (IRRPVERS).

System action: No program signatures are verified until the error is resolved and the program verification module is loaded.

RACF Security Administrator Response:

This message might have the following preceding messages:

- Either ICH451I, or
- Three of the following messages:
  1. ICH440I
  2. ICH442I, ICH443I, ICH444I, ICH445I, or ICH446I
  3. ICH451I

See these message descriptions and resolve the problem. After the problem is resolved, notify your system programmer to run the IRRVERLD program to load the program verification module (IRRPVERS).

If this message is not preceded by three other messages, there might be a problem in the PROGRAM class profile covering resource IRRPVERS. Ensure that this profile is correctly defined, and that the data set name in the member list points to the data set that contains the program verification module (IRRPVERS). You must also ensure that the SIGVER segment of this profile is defined, and does not contains the following values:

- FAILLOAD(NEVER)
- SIGAUDIT(NONE)
- SIGREQUIRED(NO)

If the IRRPVERS profile in the PROGRAM class is correctly defined, ensure that the SETR WHEN(PROGRAM) option is set and refreshed.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH448I RACF program signature verification module is loaded. Program signature verification is available on this system.

Explanation: The RACF initialization process or the IRRVERLD program has loaded and verified the program verification module (IRRPVERS). Program signature verification is available on this system.

System action: Subsequent program verification operations complete normally.

RACF Security Administrator Response:

None

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH449I RACF program signature verification is already loaded.

Explanation: The IRRVERLD program detected that the program verification module (IRRPVERS) has already been loaded and verified.

System action: The IRRVERLD program ends with return code 4. No changes are made to the system.

RACF Security Administrator Response:

None

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH450I The RACF program verification module is not loaded. Program signature verification is not available.

Explanation: An attempt was made to load a signed program that is covered by a profile in the PROGRAM class. The profile indicates that the signature is to be verified. However, the program verification module (IRRPVERS) has not been loaded. Program verification is only available when the program verification module (IRRPVERS) has been loaded.

Note: The program name is identified in message ICH440I or ICH441I. One of these messages precedes this message.

System action: Depending on the program configuration options set in the SIGVER segment of the PROGRAM profile, the attempt to load the program either succeeds or fails.

RACF Security Administrator Response:

Notify your system programmer to run the IRRVERLD program to load and verify the program verification module (IRRPVERS).

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH451I RACF encountered an error while attempting to load the program verification module. Operation code = Xaa Return code Xbbbb and Reason code Xcccc. Supplemental diagnostic code 1 = X'dddddddd'. Supplemental diagnostic code 2 = X'eeeeeeee'. Supplemental diagnostic code 3 = X'ffffffff'. Supplemental diagnostic code 4 = X'gggggggg'. Supplemental diagnostic code 5 = X'hhhhhhhh'.

Explanation: A system service failed while RACF attempted to load the program verification module

(IRRPVERS). The failing system service, return code, and reason code, are defined in the following table:

Operation code (X'aa')	Failing system service	Return code	Reason code
X'01'	IEANTCR	X'bbbb'	X'cccc'
X'02'	IEANTRT	X'bbbb'	X'cccc'
X'03'	CSVDYLPA REQUEST=ADD	X'bbbb'	X'cccc'
X'04'	BLDL	X'bbbb'	X'cccc'
X'05'	STORAGE OBTAIN	X'bbbb'	X'cccc'
X'06'	LOAD	X'bbbb'	X'cccc'
Note: The return code and reason code from the failing service are included in this message.			

If the operation code is X'03', the supplemental diagnostic codes have values. You can use the supplemental diagnostic values in the following table to determine the problem:

Supplemental Diagnostic Code	Value
1	LpmeaOutputFlags
2	LpmeaRetcode
3	LpmeaRsncode
4	LpmeaAbendCode
5	LpmeaAbendRsnCode
See the CSVDYLPA ADD service in *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for more information about the supplemental diagnostic codes.	

If the operation code is not X'03', the supplemental codes have no meaning.

System action: No program signatures are verified until the error is resolved and the program verification module is loaded. Depending on the signature verification options set in the SIGVER segment of the PROGRAM profile, the attempt to load the program might fail.

RACF Security Administrator Response:

Determine the reason for the system service failure using the return codes and resolve the problem. After the problem is resolved, notify your system programmer to run the IRRVERLD program to load the program verification module (IRRPVERS).

Destination: Descriptor code is 6. Routing codes are 9 and 11.

ICH452I The RACF program verification module self-test failed. Program signature verification is not available.

Explanation: The program verification module (IRRPVERS) encountered an error while performing a self test during an attempt to initialize.

System action: No program signatures are verified until the error is resolved and the program verification module is loaded. A record is added to LOGREC with additional diagnostic information.

RACF Security Administrator Response:

Contact IBM support.

Destination: Descriptor code is 6. Routing codes are 9 and 11.

Chapter 7. Diagnosis considerations

This information supplements *z/OS Security Server RACF Diagnosis Guide*.

The following callable service is added:

Table 2. Callable services type numbers

CALLABLE SERVICE	Service Number or TYPE (HEX)	Service Number or TYPE (DECIMAL)
IRRSPS00	35	53

Variable error data is recorded by the following callable service:

Variable data recorded by RACF Callable Services

R_PgmSignVer (CSECT IRRSPS00)

- Error with range
  - Service name: IRRRPS00
  - Variable data: Too many ranges
  - Primary symptom string return code: Index of invalid range
  - Secondary symptom string reason code: 0
- Error with end address
  - Service name: IRRRPS00
  - Variable data: End Address before start

    ```
    Start: 0Xaaaaaaaa aaaaaaaa
    End: 0Xbbbbbbbb bbbbbbbb"
    ```

    - Aaaaaaa aaaaaaaa=start address
    - Bbbbbbbb bbbbbbbb = end address
  - Primary symptom string return code: Index of invalid range
  - Secondary symptom string reason code: 0
- Extract error
  - Service name: SIGINIT
  - Variable data: Extract error:

    ```
    0xaaaaaaaa
    0xbbbbbbbb
    0xcccccccc
    ```

    - Aaaaaaaa=SAFRC
    - Bbbbbbb=RACFRC
    - Ccccccccc=RACFRSN

    Primary symptom string return code: RACROUTE return code

    Secondary symptom string reason code: RACROUTE reason code
- Error with APPLDATA algorithm
  - Service name: SIGINIT
  - Variable data: APPLDATA invalid algorithm: `alg`
    - Alg=What was specified in appldata
  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0

- Error locating APPLDATA in profile
  - Service name: SIGINIT
  - Variable data: APPLDATA not found in profile
  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- APPLDATA error with leading/trailing blanks
  - Service name: SIGINIT
  - Variable data: APPLDATA: Leading/Trailing blank encountered
  - Primary symptom string return code: Variable ‘I’ – Index of first blank in appldata
  - Secondary symptom string reason code: 0
- Error with APPLDATA algorithm name
  - Service name: SIGINIT
  - Variable data: APPLDATA: algorithm name too long
  - Primary symptom string return code: Variable ‘I’ – Index of first blank in appldata
  - Secondary symptom string reason code: 0
- RING error, ID too long
  - Service name: SIGINIT
  - Variable data: RING: / not found or ID too long
  - Primary symptom string return code: Variable ‘I’, index of ‘/’
  - Secondary symptom string reason code: Variable ‘j’ Index of ‘ ‘
- RING error, name too long
  - Service name: SIGINIT
  - Variable data: RING: Name too long or not specified
  - Primary symptom string return code: Variable ‘I’, index of ‘/’
  - Secondary symptom string reason code: Variable ‘j’ Index of ‘ ‘
- Key error in default certificate
  - Service name: SIGINIT
  - Variable data: No Private Key in default certificate
  - Primary symptom string return code: Number of certificates in ring
  - Secondary symptom string reason code: Index of default certificate in ring
- Error with key analysis
  - Service name: SIGINIT
  - Variable data: Key analysis error
  - Primary symptom string return code: RC from analysisKey() routine
  - Secondary symptom string reason code: 0
- Error with number of certificates in ring
  - Service name: SIGINIT
  - Variable data: Too many certificates in ring
  - Primary symptom string return code: Number of certificates in ring
  - Secondary symptom string reason code: 0
- Error with ring default certificate
  - Service name: SIGINIT
  - Variable data: No default certificate found in ring

  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- Error with certificate ordering
  - Service name: SIGINIT
  - Variable data: Certificate ordering error
  - Primary symptom string return code: RC from OrderPKCS7CA
  - Secondary symptom string reason code: 0
- Error with trust chain length
  - Service name: SIGINIT
  - Variable data: Trust chain too long
  - Primary symptom string return code: Number of items in trust chain
  - Secondary symptom string reason code: 0
- Error with R_datalib
  - Service name: SIGINIT
  - Variable data: R_datalib error:

    ```
    Function Code X,
    RC=0Xaaaaaaaa 0Xbbbbbbbb 0Xcccccccc
    ```
  - Primary symptom string return code: R_datalib Function code
  - Secondary symptom string reason code: R_datalib SAFRC
- Digest failure error
  - Service name: IRRRPS21
  - Variable data: Digest failure
  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- Digest failure 2 error
  - Service name: IRRRPS21
  - Variable data: Digest failure 2
  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- Number of ranges error
  - Service name: IRRRPS21
  - Variable data: Too many ranges
  - Primary symptom string return code: # specified ranges
  - Secondary symptom string reason code: 0
- Error with end address
  - Service name: IRRRPS21
  - Variable data: End address before start:

    ```
    Start: 0Xaaaaaaaa aaaaaaaa
    End: 0Xbbbbbbbb bbbbbbbb"
    ```
    - Aaaaaaa aaaaaaaa=start address
    - Bbbbbbbb bbbbbbbb = end address
  - Primary symptom string return code: Failing range index
  - Secondary symptom string reason code: 0
- Context error
  - Service name: IRRRPS21
  - Variable data: New context

  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- New digest error
  - Service name: IRRRPS21
  - Variable data: New digest
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Digest update error
  - Service name: IRRRPS21
  - Variable data: Digest update
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Digest update 2 error
  - Service name: IRRRPS21
  - Variable data: Digest update 2
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- New context error
  - Service name: IRRRPS31
  - Variable data: New context
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Pk service error
  - Service name: IRRRPS31
  - Variable data: Pk service
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Rsa sign error
  - Service name: IRRRPS31
  - Variable data: Rsa sign
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Rsa verify error
  - Service name: IRRRPS31
  - Variable data: Rsa verify
  - Primary symptom string return code: CLiC rc
  - Secondary symptom string reason code: 0
- Decode failure 1 error
  - Service name: IRRRPS51
  - Variable data: Decode Failure 1
  - Primary symptom string return code: RC from decodeSimple()
  - Secondary symptom string reason code: 0
- Decode failure 2 error
  - Service name: IRRRPS51
  - Variable data: Decode Failure 2

```
aaaaaaaa bbbbbbbb cccccccc dddddddd
eeeeeeee ffffffff gggggggg hhhhhhhh iiiiiiiiI
```

  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- Decode failure 3 error
  - Service name: IRRRPS51
  - Variable data: Decode Failure 3 (hex data)
  - Primary symptom string return code: 0
  - Secondary symptom string reason code: 0
- Decode failure 4 error
  - Service name: IRRRPS51
  - Variable data: Decode failure 4
  - Primary symptom string return code: RC from DecodeSimple
  - Secondary symptom string reason code: 0
- Decode failure 5 error
  - Service name: IRRRPS51
  - Variable data: Decode failure 5
  - Primary symptom string return code: Rc from analysiskey()
  - Secondary symptom string reason code: 0
- Decode failure 6 error
  - Service name: IRRRPS51
  - Variable data: Decode failure 6
  - Primary symptom string return code: Value of variable seqCount should be 5
  - Secondary symptom string reason code: 0
- Error with signature algorithm
  - Service name: VERFINAL
  - Variable data: Unsupported signature algorithm
  - Primary symptom string return code: Value of CxSignAlg
  - Secondary symptom string reason code: 0
- APPLDATA format error
  - Service name: VERFINAL
  - Variable data: Appldata format error:

    ```
    #### aaaaa
    ```

  - Primary symptom string return code: Variable BlankPos: Index of ‘ ‘ in Appldata
  - Secondary symptom string reason code: Variable DelimPos: index of ‘/’ in ApplData
- Extract error
  - Service name: VERFINAL
  - Variable data: Extract Error:

    ```
    0xaaaaaaaa
    0xbbbbbbbb
    0xcccccccc
    ```

    - Aaaaaaa=safrc
    - Bbbbbbbb=racfrc
    - Ccccccccc=racfrsn

  - Primary symptom string return code: RACROUTE
  - Secondary symptom string reason code: RACROUTE
- Decode X509 error
  - Service name: VERFINAL
  - Variable data: Decode X509 error
  - Primary symptom string return code: Rc from Decode X509
  - Secondary symptom string reason code: Certificate number (I loop index)
- SetCertificate error
  - Service name: VERFINAL
  - Variable data: SetCertificate Error
  - Primary symptom string return code: Rc From
  - Secondary symptom string reason code: Certificate number
- R_datalib error
  - Service name: VERFINAL
  - Variable data: R_datalib error:

    ```
    Function code aaaaaaaa,
    RC=0xbbbbbbbb 0xcccccccc 0xdddddddd
    ```

    - Aaaaaaaa=r_datalib func code
    - Bbbbbbbb=saf RC from r_datalib
    - Cccccccc=RACFRC from r_datalib
    - Dddddddd=RACFRsn from r_datalib
  - Primary symptom string return code: R_datalib function code
  - Secondary symptom string reason code: R_datalib RC

Chapter 8. Data area considerations

This information supplements *z/OS Security Server RACF Data Areas*.

Information for the following data areas is updated to support program signing and verification:

COMP: Common SAF/RACF Parameter List for z/OS UNIX System Services

Common name:

Common SAF/RACF parameter list for z/OS UNIX System Services security functions

Macro ID:

IRRPCOMP

DSECT Name:

COMP, IUSP, CSID, EXID, GETG, CHKP, GMAP, CKPO, QRYS, CMOD, CLID, CAUD, COWN, UMSK, KACC, QRYF, KFOW, MKRT, PTRC, MFSP, RAUD, GUGP, FORK, MISP, IACC, IOWN, CKO2, GETE, DKEY, DINF, DRUR, DAUT, GINF, RAUX, INTA, ADMN, UMAP, CDDL, KERB, TKTS, PKIS, CACH, PRXY, RACL, PGSN, WPRV, SECL

Size:

Section	Size
PRXY	44 bytes
RACL	40 bytes
PGSN	12 bytes
WPRV	12 bytes
SECL	24 bytes

Offsets			Len	Name	Description
Dec	Hex	Type			
0	(0)	STRUCTURE	12	PGSN	
0	(0)	ADDRESS	4	PGSN_NUM_PARMS@	Address of a fullword containing the total number of parameters included in COMP and PGSN.
4	(4)	ADDRESS	4	PGSN_FUNC@	Address of 2-byte function code. Constants for the function codes are supplied below.

Offsets			Len	Name	Description
Dec	Hex	Type			
8	(8)	ADDRESS	4	PGSN_FUNC_PARML@	Address of the function specific parameter list corresponding to the function code. See *z/OS Security Server RACF Callable Services* for function specific parameter lists for callable service R_PgmSignVer.

Constants

PGSN Function code constants:

Len	Type	Value	Name	Description
4	DECIMAL	1	PGSN_FC_SIGINIT	Initialize program signing
4	DECIMAL	2	PGSN_FC_SIGUPDATE	Sign intermediate program data
4	DECIMAL	3	PGSN_FC_SIGFINAL	Finalize program signature
4	DECIMAL	4	PGSN_FC_SIGCLEAN	Terminate signature operation
4	DECIMAL	5	PGSN_FC_VERINIT	Initialize signature verification
4	DECIMAL	6	PGSN_FC_VERUPDATE	Digest intermediate program data
4	DECIMAL	7	PGSN_FC_VERFINAL	Perform final verification
4	DECIMAL	8	PGSN_FC_VERCLEAN	Terminate verification operation
4	DECIMAL	9	PGSN_FC_VERINTER	Interrogate directive Seefor function specific parameter lists for callable service R_PgmSignVer.

COMY: 64-BIT enabled SAF callable services

Macro ID:
IRRPCOMY

DSECT Name:
COMY, RAUX, PGSN

Size:

Section	Size
COMY	48 bytes
PGSN	24 bytes
RAUX	144 bytes

Offsets			Len	Name	Description
Dec	Hex	Type			
0	(0)	STRUCTURE	24	PGSN64	

Offsets			Len	Name	Description
Dec	Hex	Type			
0	(0)	ADDRESS	8	PGSN64_NUM_PARMS@	Address of a fullword containing the total number of parameters included in COMP and PGSN.
4	(4)	ADDRESS	8	PGSN64_FUNC@	Address of a 2-byte function code. See data area COMP for the function code constants.
8	(8)	ADDRESS	8	PGSN645_FUNC_PARML@	Address of the function specific parameter list corresponding to the function code. See *z/OS Security Server RACF Callable Services* for function specific parameter lists for callable service R_PgmSignVer.

Cross Reference

Name	Hex Offset	Hex Value
PGSN64	0	
PGSN64_NUM_PARMS@	0	
PGSN64_FUNC@	8	
PGSN645_FUNC_PARML@	10	

FC: z/OS UNIX System Services Security Function Code Table

Constants

Len	Type	Value	Name	Description
1	DECIMAL	53	IRRSPS00#	Function code 53 - R_PgmSignVer

Cross Reference

Name	Hex Offset	Hex Value	Level
IRRSPS00#	0	35	2

Chapter 9. Callable services considerations

For the program signing and verification enhancement, a new callable service, R_PgmSignVer, is introduced. Callable service R_datalib is also updated with new attribute flag CDDL(X)_ATT_SKIPAUTH for bypassing authorization checks. In addition, R_admin reference information is updated to describe the SIGVER segment fields.

R_datalib (IRRSDL00 or IRRSDL64): OCSF data library

R_datalib is updated with new attribute flag CDDL(X)_ATT_SKIPAUTH for bypassing authorization checks. The **Attributes** parameter is updated as follows:

Attributes

The name of a 4-byte area containing bit settings that direct the function to be performed. The bit settings are mapped as follows:

- The DataGetFirst (X'01') and DataGetNext (X'02') functions

  X'80000000' - CDDL(X)_ATT_ALL_KEYTYPES flag. This flag directs R_datalib to differentiate between PCICC key types and ICSF key types, DSA key types and PKCS #1 key types when returning the Function Specific Parameter List field *Private_key_type*. When this flag is off, R_datalib will handle the PCICC key type as an ICSF key type and return the value X'00000002'. It will also handle the DSA key type as a PKCS #1 key type and return the value X'00000001'.

  X'20000000' - CDDL(X)_ATT_SKIPAUTH flag. This flag directs R_datalib to bypass authorization checks to RACF key rings for a supervisor state or system key caller. This does not bypass the authorization required in order to retrieve private key information, nor does this bypass authorization checks for PKCS #11 tokens. This flag is ignored for problem state callers.

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- The CheckStatus (X'04') function

  X'20000000' - CDDL(X)_ATT_SKIPAUTH flag. This flag directs R_datalib to bypass authorization checks to RACF key rings for a supervisor state or system key caller. This does not bypass the authorization required in order to retrieve private key information, nor does this bypass authorization checks for PKCS #11 tokens. This flag is ignored for problem state callers.

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- The GetUpdateCode (X'05) function

  X'20000000' - CDDL(X)_ATT_SKIPAUTH flag. This flag directs R_datalib to bypass authorization checks to RACF key rings for a supervisor state or system key caller. This does not bypass the authorization required in order to retrieve private key information, nor does this bypass authorization checks for PKCS #11 tokens. This flag is ignored for problem state callers.

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- The NewRing (X'07') function

  X'80000000' - CDDL(X)_ATT_REUSE_RING flag. This flag directs R_datalib to reuse the existing key ring and remove all the certificates from it. When this flag is off, it indicates the creation of a new key ring.

All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- The DataPut (X'08') function

  X'80000000' - CDDL(X)_ATT_TRUST flag. This flag is used to add certificates, with the TRUST status. When this flag is off, RACF will determine the status based on the following factors in the same way that the RACDCERT ADD command behaves:

  – Whether the issuer of the certificate is trusted
  – Whether the signature of the certificate can be verified
  – Whether the certificate has expired
  – Whether the validity date range of the certificate is within that of its issuer

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements. If the certificate already exists, turning on this attribute will change its status from NOTRUST to TRUST when connecting it to the key ring. However, if the status is already HIGHTRUST, it will remain unchanged.

  X'40000000' - CDDL(X)_ATT_HIGHTRUST flag. This flag is used to add or change certificates with the HIGHTRUST status if the certificate to be added or changed is under CERTAUTH; otherwise this value will be treated as CDDL(X)_ATT_TRUST, that is, add or change certificates with the TRUST status.

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- The DataRemove (X'09') function

  X'80000000' - CDDL(X)_ATT_DEL_CERT_TOO flag. This flag is used to indicate the deletion of the certificate from the RACF database after being removed from the ring, if the certificate is not connected to any other rings. When this flag is off, it indicates the removal of the certificate from the key ring only. When this attribute is specified and the DIGTCERT class is RACLISTed, a successful DataRemove returns 4 4 12 instead of 0 0 0 to indicate that a DataRefresh call is needed.

  All other bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

- All other functions

  All bit positions are reserved and must be set to zero to ensure compatibility with future enhancements.

R_admin reference information

The following general resource administration table is added to define SIGVER segment field names and their usage. All field names relate directly to the RDEFINE, RALTER, and RLIST keywords. Although the fields are alphabetized in the following table, there is no defined order in which the fields are returned when using the extract functions. See z/OS Security Server RACF Command Language Reference for questions pertaining to field usage and data. Note that within the command image generated internally, RACF truncates long keywords to 12 characters.

Table 3. SIGVER segment fields

Field name	Flag byte value	RDEFINE/RALTER keyword reference (R_admin keyword table has a 12 character limit)	Allowed on add requests	Allowed on alter requests	Returned on extract requests
FAILLOAD	'Y'	SIGVER(FAILLOAD(xx))	Yes	Yes	Yes
	'N'	SIGVER(NOFAILLOAD)	No	Yes	
SIGAUDIT	'Y'	SIGVER(SIGAUDIT(xx))	Yes	Yes	Yes
	'N'	SIGVER(NOSIGAUDIT)	No	Yes	
SIGREQD (boolean)	'Y'	SIGVER(SIGREQUIRED(YES))	Yes	Yes	Yes
	'N'	SIGVER(SIGREQUIRED(NO))	Yes	Yes	

R_PgmSignVer (IRRSPS00): Program Sign and Verify

Function

The R_PgmSignVer service provides the functions required to apply a digital signature to a z/OS program object, and the functions required to verify such a signature. The signing services are intended for use by the z/OS program binder. The verification services are intended for use by the z/OS loader.

The signing services consist of the following functions:

1. Initialize signing – Allocates and initializes a work area to perform message digestion (hash) against the program's data, and reads the digital certificates from the program signing key ring.
2. Digest intermediate program data – Hashes a portion of the program's data for signing.
3. Generate signature – Hashes the final portion of the program's data, if provided, and generates the digital signature by encrypting the calculated hash using the private key from the default certificate in the key ring. Any resources obtained by the initialize signing function are freed before returning.
4. Cleanup – Frees resources obtained by the initialize signing function. To be called in the event that signature generation will not be completed by the generate signature function (for example, for recovery cleanup).

The verification services consist of the following functions:

1. Initialize signature-verification – Allocates and initializes a work area to perform message digestion (hash) against the program's data, and hashes any initial program data that is supplied.
2. Digest intermediate program data – Hashes a portion of the program's data for verification.
3. Final verification – Hashes the final portion of the program's data, if provided. The signature provided with the program is then decrypted with the public key from the end-entity certificate that accompanies the signature, and the two hash values are compared to verify the signature. Any resources obtained by the initialize signature-verification function are freed before returning.
4. Cleanup – Frees resources obtained by the initialize signature-verification function. To be called in the event that signature-verification will not be completed by calling the final verification function (for example, for recovery cleanup)

5. Interrogate directive – Generate appropriate return code and perform auditing according to security settings when a program signature cannot be verified.

Requirements

Authorization:	Any PSW key in supervisor or problem state. **Note:** In the following documentation, a caller that is in either supervisor state or a system key is referred to as “authorized”. Otherwise, the caller is referred to as “unauthorized”.
Dispatchable unit mode:	Task of user
Cross memory mode:	PASN = HASN
AMODE:	31 or 64
RMODE:	Any
ASC mode:	Primary mode
Recovery mode:	ESTAE. Caller cannot have a FRR active.
Serialization:	Enabled for interrupts
Locks:	No locks held
Control parameters:	The parameter list and the work area must be in the primary address space. The words containing the ALETs must be in the primary address space. The Num_parms parameter must be in the primary address space.

Linkage conventions

Callers in 31-bit addressing mode should link-edit the IRRSPS00 stub module with their code, and use the IRRPCOMP mapping macro. Callers in 64-bit addressing mode should link-edit the IRRSPS64 stub module with their code, and use the IRRPCOMY mapping macro.

RACF authorization

For unauthorized callers of the program signing services, the caller must have sufficient authority to use the key ring specified in the parameter list (or if not specified, then as defined in the appropriate profile in the FACILITY class) and the private key contained within it as determined by the R_datalib callable service and ICSF.

For the signature-verification services, there are no authorization requirements, regardless of the caller’s state.

Format

```
CALL IRRSPS00 (Work_area,
               ALET, SAF_return_code,
               ALET, RACF_return_code,
               ALET, RACF_reason_code,
               Num_parms,
               Function_code,
               Function_parmlist
              )
```

Parameters

Work_area
The name of a 1024-byte work area for SAF. The work area must be in the primary address space.

ALET
The name of a word containing the ALET for the following parameter. Each parameter must have an ALET specified. Each ALET must be 0 for this service. The words containing the ALETs must be in the primary address space.

SAF_Return_Code
The name of a fullword in which the SAF router returns the SAF return code.

RACF_Return_Code
The name of a fullword in which the service routine stores the return code.

RACF_Reason_Code
The name of a fullword in which the service routine stores the reason code.

Num_parms
Specifies the name of a fullword that contains the total number of parameters in the parameter list. The contents of this field must be set to binary ten.

Function_code
The name of a 2-byte area containing the Function code. The function code has one of the following values:

X'0001'
Initialize signing. (Function name SIGINIT.) This function must be called before calling any of the other signing functions.

X'0002'
Digest intermediate program data for signature generation. (Function name SIGUPDAT.) This function is optional. It should be called only if all the program's data cannot be processed on one call to generate signature. It may be called multiple times before calling generate signature.

X'0003'
Generate signature. (Function name SIGFINAL.) This function finalizes the signature generation and returns the result. It also frees any work area storage that may have been allocated.

X'0004'
Terminates the signing operation and frees resources allocated by SIGINIT. (Function name SIGCLEAN.) This function should be called only if signature generation is not to be finalized with a call to SIGFINAL. Note that all R_PgmSignVer functions will perform this cleanup if they return an error to the caller. The caller needs to call the cleanup function only if it is terminating for its own reason.

X'0005'
Initialize signature-verification and optionally digest initial program data. (Function name VERINIT.) This function must be called before calling any of the other verification functions except VERINTER (interrogate directive).

X'0006'
Digest intermediate program data for signature-verification. (Function name VERUPDAT.) This function is optional. It should be called only if

all the program's data cannot be processed on the VERINIT and VERFINAL calls. It may be called multiple times before performing final verification.

X'0007'
Perform final verification. (Function name VERFINAL.) This function finalizes the signature-verification and returns the result. It also audits the event and frees any work area storage that may have been allocated. If all the program data can be specified in a single call, then VERFINAL can be called without first calling VERINIT. See Table 10 on page 65 for more information.

X'0008'
Terminates the signing operation and frees resources allocated by VERINIT. (Function name VERCLEAN.) This function should be called only if signature generation is not to be finalized with a call to VERFINAL. Note that all R_PgmSignVer functions will perform this cleanup if they return an error to the caller. The caller only needs to call the cleanup function if it is terminating for its own reason.

X'0009'
Interrogate directive. (Function name VERINTER.) This function examines the directive (supplied within the ICHSFENT in the function-specific parameter list) to determine the appropriate action. This would be used for the cases where VERFINAL will not be called. For example, when digital signature processing is required but the module does not have a digital signature. This function is not available to unauthorized callers.

Function_parmlist
Specifies the name of the function code specific parameter list area for the Function_code specified.

All address fields are 8-byte addresses. When referring to 31-bit storage addresses, the caller must make sure that the high-order word of the address field is set to binary zeros.

Table 4. Function_parmlist for SIGINIT

Field	Attributes	Usage	Description
PGSN_SI_PLIST	Structure	In	Function-specific parameter list for signing initialization.
PGSN_SI_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'SIGINIT '.
PGSN_SI_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_SI_PGM_NAME_LEN	4 byte numeric	In	Length of the name of the program being signed. The length must not exceed 8 characters.
PGSN_SI_PGM_NAME@	Address of	In	Address of the name of the program being signed. **Note:** This parameter is used to derive the name/token that is used for subsequent calls. As such, it does not necessarily need to be the program name, but must be a unique value which does not result in a name collision with other signing operations.

Table 4. Function_parmlist for SIGINIT (continued)

Field	Attributes	Usage	Description
PGSN_SI_KEYRING_NAME@	Address of	In	Address of the name of the SAF key ring that contains the certificates to be used for signing. The address is meaningful only if PGSN_SI_KEYRING_LEN is a non-zero value. The name that this address points to has the following syntax: *owning-userid/ring-name* The owning-userid (but not the slash) may be omitted if the key ring is owned by the user ID associated with the calling application.
PGSN_SI_KEYRING_LEN	4 byte numeric	In	Length of the name of the SAF key ring that contains the certificates to be used for signing. Set this field to binary zero to have the security manager determine the key ring to use.
PGSN_SI_SIGINFO_LEN	4 byte numeric	Out	Length of the ZOSSignatureInfo structure which will be returned as part of the signature area structure in the SIGFINAL call.
PGSN_SI_DIGEST_ALG	1 byte numeric	In	Numeric value indicating what message digest algorithm to use for the signing. Set this field to binary zero to have the security manager determine the algorithm to use. A value of 1 indicates that SHA256 is to be used.

Table 5. Function_parmlist for SIGUPDAT

Field	Attributes	Usage	Description
PGSN_SU_PLIST	Structure	In	Function-specific parameter list for intermediate signing.
PGSN_SU_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'SIGUPDAT'.
PGSN_SU_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_SU_PGM_NAME_LEN	4 byte numeric	In	Length of the name of the program being signed. The length must not exceed 8 characters.
PGSN_SU_PGM_NAME@	Address of	In	Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call.
PGSN_SU_PGM_DATA@	Address of	In	Address of a structure specifying the intermediate range(s) of data to sign. The structure is mapped by PGSN_DATA_RANGE. See usage note 7 in "Usage notes for program verification" on page 74 for the format of this structure.

Table 6. Function_parmlist for SIGFINAL

Field	Attributes	Usage	Description
PGSN_SF_PLIST	Structure	In	Function-specific parameter list for final signing.
PGSN_SF_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'SIGFINAL'.
PGSN_SF_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_SF_PGM_NAME_LEN	4 byte numeric	In	Length of the name of the program being signed. The length must not exceed 8 characters.

Table 6. Function_parmlist for SIGFINAL (continued)

Field	Attributes	Usage	Description
PGSN_SF_PGM_NAME@	Address of	In	Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call.
PGSN_SF_PGM_DATA@	Address of	In	Address of a structure specifying the final range(s) of data to sign. The structure is mapped by PGSN_DATA_RANGE. See usage note 7 in "Usage notes for program verification" on page 74 for the format of this structure.
PGSN_SF_SIG_AREA@	Address of	Out	Address of the allocated signature area structure. See usage note 6 in "Usage notes for program signing" on page 72 for the format of the area.
PGSN_SF_SUBPOOL	1 byte numeric	In	Subpool to be used for allocation of the signature data structure. For unauthorized callers, this must be a value in the range 1 – 127.

Table 7. Function_parmlist for SIGCLEAN

Field	Attributes	Usage	Description
PGSN_SC_PLIST	Structure	In	Function-specific parameter list for signing cleanup.
PGSN_SC_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'SIGCLEAN'.
PGSN_SC_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_SC_PGM_NAME_LEN	4 byte numeric	In	Length of the name of the program being signed. The length must not exceed 8 characters.
PGSN_SC_PGM_NAME@	Address of	In	Address of the name of the program being signed. Must be the same as the value supplied on the SIGINIT call.

Table 8. Function_parmlist for VERINIT

Field	Attributes	Usage	Description
PGSN_VI_PLIST	Structure	In	Function-specific parameter list for verification initialization.
PGSN_VI_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'VERINIT '.
PGSN_VI_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_VI_PGM_NAME_LEN	4 byte numeric	In	For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers.
PGSN_VI_PGM_NAME@	Address of	In	For unauthorized callers, address of the name of the program being verified. Ignored for authorized callers.
PGSN_VI_CONTEXT@	Address of	Out	For authorized callers, address of the allocated verify context that the caller should pass in to subsequent verification calls. Ignored for unauthorized callers.
PGSN_VI_PGM_DATA@	Address of	In	Address of a structure specifying the initial range(s) of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note 7 in "Usage notes for program verification" on page 74 for the format of this structure.

Table 8. Function_parmlist for VERINIT (continued)

Field	Attributes	Usage	Description
PGSN_VI_SIGINFO@	Address of	In	Address of the ZOSSignatureInfo structure extracted from the program object being verified.
PGSN_VI_SIGINFO_LEN	4 byte numeric	In	Length of the ZOSSignatureInfo structure extracted from the program object being verified.
PGSN_VI_DIGEST_ALG	1 byte numeric	In	Numeric value indicating what message digest algorithm to use for the verification. A value of 0 means the value contained in the ZOSSignatureInfo structure should be used. This is the only supported value.

Table 9. Function_parmlist for VERUPDAT

Field	Attributes	Usage	Description
PGSN_VU_PLIST	Structure	In	Function-specific parameter list for intermediate verification.
PGSN_VU_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'VERUPDAT'.
PGSN_VU_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_VU_PGM_NAME_LEN	4 byte numeric	In	For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers.
PGSN_VU_PGM_NAME@	Address of	In	For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers.
PGSN_VU_CONTEXT@	Address of	In	For authorized callers, address of the verify context area allocated on the VERINIT call. Ignored for unauthorized callers.
PGSN_VU_PGM_DATA@	Address of	In	Address of a structure specifying the intermediate range(s) of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note 7 in "Usage notes for program verification" on page 74 for the format of this structure.

Table 10. Function_parmlist for VERFINAL

Field	Attributes	Usage	Description
PGSN_VF_PLIST	Structure	In	Function-specific parameter list for final verification.
PGSN_VF_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'VERFINAL'.
PGSN_VF_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_VF_PGM_NAME_LEN	4 byte numeric	In	For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers. If the length is zero, it is assumed that no VERINIT call was made, and the signature is generated based on the data supplied in this call, using the default digest algorithm.

Table 10. Function_parmlist for VERFINAL (continued)

Field	Attributes	Usage	Description
PGSN_VF_PGM_NAME@	Address of	In	For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers.
PGSN_VF_CONTEXT@	Address of	In	For authorized callers, address of the verify context area allocated on the VERINIT call. Ignored for unauthorized callers. If the address is zero, it is assumed that no VERINIT call was made, and the signature is generated based on the data supplied in this call, using the default digest algorithm.
PGSN_VF_PGM_DATA@	Address of	In	Address of a structure specifying the final range(s) of data to verify. The structure is mapped by PGSN_DATA_RANGE. See usage note 7 in "Usage notes for program verification" on page 74 for the format of this structure.
PGSN_VF_LOGSTRING@	Address of	In	Address of an area that consists of a 1 byte length field followed by character data (up to 255 bytes) to be included in any audit records that are created. If the address or the length byte is 0, this parameter is ignored.
PGSN_VF_ICHSFENT@	Address of	In	For authorized callers, address of the FASTAUTH entity parameter mapping containing the directive (previously retrieved from RACF by Contents Supervision). This parameter is optional. See usage notes 6 and 16 in "Usage notes for program verification" on page 74. Ignored for unauthorized callers.
PGSN_VF_SIGINFO@	Address of	In	Address of the ZOSSignatureInfo structure extracted from the program object being verified. This field is required if VERFINAL is the only call being made. It is ignored if it was already passed to VERINIT.
PGSN_VF_SIGINFO_LEN	4 byte numeric	In	Length of the ZOSSignatureInfo structure extracted from the program object being verified. This field is required if VERFINAL is the only call being made. It is ignored if it was already passed to VERINIT.

Table 11. Function_parmlist for VERCLEAN

Field	Attributes	Usage	Description
PGSN_VC_PLIST	Structure	In	Function-specific parameter list for verification cleanup.
PGSN_VC_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'VERCLEAN'.
PGSN_VC_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
PGSN_VC_PGM_NAME_LEN	4 byte numeric	In	For unauthorized callers, length of the name of the program being verified. The length must not exceed 8 characters. Ignored for authorized callers.
PGSN_VC_PGM_NAME@	Address of	In	For unauthorized callers, address of the name of the program being verified. Must be the same as the value supplied on the VERINIT call. Ignored for authorized callers.

Table 11. Function_parmlist for VERCLEAN (continued)

Field	Attributes	Usage	Description
PGSN_VC_CONTEXT@	Address of	In	For authorized callers, address of the verify context area allocated on the VERINIT call. Ignored for unauthorized callers.

Table 12. Function_parmlist for VERINTER

Field	Attributes	Usage	Description
PGSN_ID_PLIST	Structure	In	Function-specific parameter list for interrogating the directive.
PGSN_ID_EYE	8 characters	In	Eyecatcher, 8 characters. Actual value must be set by invoker: 'VERINTER'.
PGSN_ID_VERS	4 byte numeric	In	The version number for this function-specific parameter list. The contents of this field must be set to binary zero.
*	4 characters	In	Reserved
PGSN_ID_ ICHSFENT@	Address of	In	For authorized callers, address of the FASTAUTH entity parameter mapping (previously retrieved from RACF by Contents Supervision). Ignored for unauthorized callers.
PGSN_ID_LOGSTRING@	Address of	In	Address of an area that consists of a 1 byte length field followed by character data (up to 255 bytes) to be included in any audit records that are created. If the address or the length byte is 0, this parameter is ignored.
PGSN_ID_EVENT	1 byte numeric	In	Constant indicating what sigver event was detected: • x'01' – Digital signature processing is required but the module does not have a digital signature. • x'02' – Digital signature processing is required. The PDSE directory entry for the module indicates it's signed but the digital signature is missing.

Return and reason codes

R_PgmSignVer may return the following values in the return and reason code parameters:

Table 13. Return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
0	0	0	Successful completion
4	0	0	RACF not installed
8	8	4	An internal error has occurred during RACF processing of the requested function.
8	8	8	Unable to establish a recovery environment.
8	8	12	Function not available for unauthorized callers.
8	100	xx	A parameter list error has been detected. The RACF reason code identifies the parameter in error. The reason code is the offset of the parameter in error, relative to the start of COMP or COMY.

Table 13. Return and reason codes (continued)

SAF return code	RACF return code	RACF reason code	Explanation
8	104	yy	A function-specific parameter list (pointed to by the *Function_parmlist* parameter) error has been detected. The RACF reason code identifies the field in error. The reason code is the offset of the field in error, relative to the start of the function-specific parameter list. When the field is an address, the error may pertain to the address itself, or to something to which it points.

In addition to the above, R_PgmSignVer may return function specific return and reason codes:

Table 14. SIGINIT specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Signature operation is already in progress for the specified program name.
8	8	104	Security Manager is unable to determine the key ring to use.
8	8	108	Syntax error in supplied key ring name, or in the key ring name contained within the APPLDATA of the RACF FACILITY class profile.
8	8	112	Key ring does not exist or does not contain a default certificate.
8	8	116	Caller not authorized to use R_datalib to access the key ring.
8	8	120	Certificate chain in the key ring is incomplete.
8	8	124	Certificate chain contains more than 10 certificates, or key ring contains more than 50 certificates. (Some of these might not constitute part of the trust chain. However, you should not connect any certificates that do not.)
8	8	128	CA certificate in the key ring does not have certificate signing capability. (KeyUsage extension present but keyCertSign flag is off or BasicConstraints extension is present but cA flag is off.)
8	8	132	Default certificate in key ring does not have a private key.
8	8	136	Default certificate in key ring does not have code signing capability. (KeyUsage extension present but digitalSignature or nonRepudiation flag is off.)
8	8	140	The certificate signature algorithm of one or more certificates in the key ring is not supported.
8	8	144	The key type of one or more certificates in the key ring is not supported. This reason code will also be issued if the private key of the signing certificate is stored in ICSF.
8	8	148	The specified message digest algorithm not supported.
8	8	152	CA or signing certificate is expired or not yet active.

Table 14. SIGINIT specific return and reason codes (continued)

SAF return code	RACF return code	RACF reason code	Explanation
8	12	xx	Unexpected error returned from R_datalib. RACF reason code is the DataGetFirst/DataGetNext reason code returned by R_datalib.
8	16	xx	Unexpected error returned from IEANTCR. RACF reason code is the return code returned by IEANTCR.
8	20	0x00xxyyyy	An unexpected error is returned from ICSF. The hexadecimal reason code value is formatted as follows: **xx** ICSF return code. **yyyy** ICSF reason code.

Table 15. SIGUPDAT specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Signature operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	2nn	xx	Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service.

Table 16. SIGFINAL specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Signature operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	16	xx	Unexpected error returned from IEANTDL. RACF reason code is the return code returned by IEANTDL.
8	2nn	xx	Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service.

Table 17. SIGCLEAN specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Signature operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	16	xx	Unexpected error returned from IEANTDL. RACF reason code is the return code returned by IEANTDL.

Table 18. VERINIT specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Verification operation is already in progress for the specified program name.
8	12	xx	Unexpected error returned from IEANTCR. RACF reason code is the return code returned by IEANTCR.
8	16	116	The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program.
8	2nn	xx	Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service.

Table 19. VERUPDAT specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Verification operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	16	116	The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program.
8	2nn	xx	Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service.

Table 20. VERFINAL specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
0	0	See reason codes below for SAF return code 8, RACF return code 16	Signature failed verification. Continue the load.
8	8	100	Verification operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	16	See below	Signature failed verification. Fail the load.

Table 20. VERFINAL specific return and reason codes (continued)

SAF return code	RACF return code	RACF reason code	Explanation
The following group of reason codes are considered problems with the program signature (the zOSSignatureInfo structure). These would cause the load to fail when FAILLOAD(BADSIGONLY) or FAILLOAD(ANYBAD) is in effect.			
		4	The ZOSSignatureInfo structure is missing or not correct.
		8	Signature algorithm in ZOSSignatureInfo is not supported.
		12	Signer certificate is revoked. The certificate status is NOTRUST.
		16	Certificate chain is incomplete.
		20	One or more CA certificates do not have certificate signing capability. (KeyUsage extension present but keyCertSign flag is off or BasicConstraints extension is present but cA flag is off.)
		24	End-entity certificate does not have code signing capability. (KeyUsage extension present but digitalSignature or nonRepudiation flag is off.)
		28	The certificate signature algorithm of one or more certificates is not supported.
		32	The type or size of key found in one or more certificates is not supported.
		36	CA or signing certificate was expired or not yet active at the time the module was signed.
		40	Digital signature not valid.
		44	Unsupported certificate format.
The following group of reason codes are the additional conditions that would cause the load to fail due to signature processing, but do not represent a bad signature. These would cause the load to fail when FAILLOAD(ANYBAD) is in effect, but not FAILLOAD(BADSIGONLY).			
		100	The program appears to be correctly signed but one of the following conditions exists: • The root CA certificate in the zOSSignatureInfo structure of the program object is not connected to the signature-verification key ring. • The root CA certificate is marked NOTRUST.
		104	The FACILITY class profile, IRR.PROGRAM.SIGNATURE.VERIFICATION, is missing.
		108	The APPLDATA information in the FACILITY class profile, IRR.PROGRAM.SIGNATURE.VERIFICATION, is missing or not correct.
		112	The signature-verification key ring is missing.
		116	The program verification module (IRRPVERS) is not loaded. See *z/OS Security Server RACF Security Administrator's Guide* and *z/OS Security Server RACF System Programmer's Guide* for information about configuring and loading the verification module with the IRRVERLD program.

Table 20. VERFINAL specific return and reason codes (continued)

SAF return code	RACF return code	RACF reason code	Explanation
		120	An error occurred while performing a cryptographic self test on the IRRPVERS module during initialization. Contact IBM support.
8	2nn	xx	Unexpected error from the cryptographic module. The return code is 200+*nn* where *nn* identifies the function being performed. The reason code is the return code from the cryptographic module. This information should be reported to IBM service.

Table 21. VERCLEAN specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
8	8	100	Verification operation has not been initialized for the specified program name.
8	12	xx	Unexpected error returned from IEANTRT. RACF reason code is the return code returned by IEANTRT.
8	16	xx	Unexpected error returned from IEANTDL. RACF reason code is the return code returned by IEANTDL.

Table 22. VERINTER specific return and reason codes

SAF return code	RACF return code	RACF reason code	Explanation
0	0	0	Continue the load.
8	8	0	Fail the load.

Usage notes

Usage notes for program signing

1. This service tracks the resources used for signing using a task related name/token pair. The 16–byte token name has the following format:

   `IRRPSIGN`*`program-name`*

   Where *program-name* is one of the parameters provided by the caller. Consequently, for any given series of SIGINIT, SIGUPDAT, SIGFINAL, and SIGCLEAN calls used to sign a single program object, the program name value must be the same.
2. Calls to this service using different program name values are considered independent operations.
3. For a given program name, SIGINIT must be called prior to calling any of SIGUPDAT, SIGFINAL, or SIGCLEAN.
4. For a given program name, SIGINIT cannot be called a second time without terminating the first SIGINIT with a call to SIGFINAL or SIGCLEAN.
5. For a given program name, it is the caller's responsibility to call the SIGCLEAN function in the event that signature generation will not be completed by calling SIGFINAL. Note that all R_PgmSignVer functions will perform this cleanup if they return an error to the caller. The caller only needs to call the cleanup function if it is terminating for its own reason.

6. The signature area allocated and returned to the caller in the PGSN_SF_SIG_AREA@ parameter by SIGFINAL has the following format:

Table 23. PGSN_SF_SIG_AREA@ signature area format

Offset	Length	Description
0	4	Eyecatcher, “PSSD”.
4	4	Length of entire area, including the eyecatcher.
8	1	Subpool used to obtain the area storage.
9	3	Reserved.
12	4	Length of z/OS signature info area.
16	*	ZOSSignatureInfo structure to be included in the signed program object. See the next usage note for the format.

7. The ZOSSignatureInfo structure returned in the signature area is the signature data that is to be placed in the signed program object. It is DER encoded according to the following ASN.1 definition:

```
ZOSSignatureInfo ::= SEQUENCE {
   signDetails     SignatureDetails
   certs           SET OF Certificate -- In reverse hierarchy order, EE to root
   signature       BIT STRING         -- PKCS #1 format - Encrypted DigestInfo
}

SignatureDetails ::= SEQUENCE {        -- DER encoding included in data signed
   version         INTEGER(0)
   signatureAlg    AlgorithmIdentifier -- From PKCS #1
   signatureTime   OCTET STRING(12)    -- TIME DEC,ZONE=UTC,DATETYPE=YYYYMMDD
                                       -- format (EBCDIC)
}
```

8. The only supported algorithm for the signatureAlg field is sha256WithRSAEncryption with NULL parameters.
9. It is the caller’s responsibility to free the signature area when it is no longer needed.
10. The only supported message digest algorithm is SHA256.
11. The only supported certificate key type is RSA. The maximum RSA key size is 4096 bits.
12. The supported certificate signature algorithms are:
    - sha256WithRSAEncryption
    - sha1WithRSAEncryption
13. All numeric parameters are treated as unsigned.
14. All length parameters must be non-zero unless otherwise indicated.
15. On SIGINIT, if the key ring to use is not specified, the security manager determines the key ring based on security settings. See the *z/OS Security Server RACF Security Administrator’s Guide* for information on these security settings and on how to populate the key ring. There can be no more than 10 certificates within the trust chain, starting with the code signer and ending with the self-signed Certificate Authority certificate.
16. If no program data is ever passed in by the caller, a digital signature is generated solely for the SignatureDetails structure documented above.

Usage notes for program verification

1. For unauthorized callers, this service tracks the resources used for verification in a 'context' using a task related name/token pair. The 16–byte token name has the following format:

   IRRPVERF*program-name*

   Where program-name is one of the parameters provided by the caller. Consequently, for any given series of VERINIT, VERUPDAT, VERFINAL, and VERCLEAN calls used to verify the signature of a single program object, the program name must be the same.
2. Calls to this service using different program names are considered independent operations.
3. For a given program name, VERINIT must be called prior to calling any of VERUPDAT, VERFINAL (with the exception documented in the descriptions of the PGSN_VF_CONTEXT@ and PGSN_VF_PGM_NAME_LEN fields in the VERFINAL parameter list), or VERCLEAN.
4. For a given program name, VERINIT cannot be called a second time without terminating the first VERINIT with a call to VERFINAL or VERCLEAN.
5. For a given program name, it is the caller's responsibility to call the VERCLEAN function in the event that signature generation will not be completed by calling VERFINAL. Note that all R_PgmSignVer functions will perform this cleanup if they return an error to the caller. The caller only needs to call the cleanup function if it is terminating for its own reason.
6. If auditing is required, it is performed in the VERFINAL (or VERINTER) call. Auditing is only performed when the ICHSFENT is provided by an authorized caller, subject to the audit settings from the directive within and the outcome of the VERFINAL service.
7. Some signature generation and all verification functions allow, via a pointer in the function-specific parameter list, the specification of an array of ranges of data to be hashed. This is optional. If the address is 0, no data will be hashed. The ranges are defined using the structure mapped by PGSN_DATA_RANGE in the IRRPCOMP mapping macro. This structure must exist in storage within the primary address space. The structure consists of an ALET followed by a fullword specifying the number of ranges which follow (if the number of ranges is 0, no data will be hashed). This is followed by an array of pointer pairs. Each pointer is an 8-byte pointer. AMODE(31) callers must set the high order full word of the pointer fields to 0. The first pointer is the address of the first byte of the range, and the second pointer is the address of the last byte of the range (they can be the same, for a length of 1). The maximum number of ranges which can be specified per call is defined in the PGSN_DATA_NUM_RANGES_MAX constant.

Field	Attributes	Description
PGSN_DATA_RANGE	Structure	Ranges of data to verify.
PGSN_DATA_ALET	4 byte numeric	The ALET for the address space containing the data.
PGSN_DATA_NUM_RANGES	4 byte numeric	The number of data ranges in the following array, not to exceed PGSN_DATA_NUM_RANGES_MAX.
PGSN64_DATA_RANGE_LIST	Array	Repeating array of the following data items.
PGSN_DATA_START@	Address of	Address of the first byte in the range.
PGSN_DATA_END@	Address of	Address of the last byte in the range.

8. The default message digest algorithm is SHA256. This is the only supported message digest algorithm.
9. The ZOSSignatureInfo structure is DER encoded. It has the following ASN.1 definition:

```
ZOSSignatureInfo ::= SEQUENCE {
   signDetails     SignatureDetails
   certs           SET OF Certificate -- In reverse hierarchy order, EE to root
   signature       BIT STRING         -- PKCS #1 format - Encrypted DigestInfo
}

SignatureDetails ::= SEQUENCE {       -- DER encoding included in data signed
   version         INTEGER(0)
   signatureAlg    AlgorithmIdentifier -- From PKCS #1
   signatureTime  OCTET STRING(12)      -- TIME DEC,ZONE=UTC,DATETYPE=YYYYMMDD
                                        -- format (EBCDIC)
}
```

10. The only supported algorithm for the signatureAlg field is sha256WithRSAEncryption with NULL parameters.
11. The only supported certificate key type is RSA. The maximum RSA key size is 4096 bits.
12. The supported certificate signature algorithms are:
    - sha256WithRSAEncryption
    - sha1WithRSAEncryption
13. All numeric parameters are treated as unsigned.
14. All length parameters must be non-zero unless otherwise indicated.
15. The program signature-verification key ring is specified using the APPLDATA field of FACILITY class profile IRR.PROGRAM.SIGNATURE.VERIFICATION. See *z/OS Security Server RACF Security Administrator's Guide* for more information about creating profiles.
16. If there is no ICHSFENT, and thus no directive, supplied by the caller, the verification occurs on the signature, but there is no check for the root CA certificate being trusted, and no auditing performed.

Related services

None.

Chapter 10. Macros and interface considerations

A new event code, 86(56): R_PgmSignVer, has been added:

Updates to the table of event codes and event code qualifiers

EVENT dec(hex)	Command	Code qualifier dec(hex)	Description	Relocate type sections (possible SMF80DTP/SMF80DA2 values)
86(56)	R_PgmSignVer	0(0)	Successful signature verification	1, 15, 46, 49, 53, 66, 331, 332, 386, 392, 393, 394, 395, 409, 410, 411, 412, 413, 414
		1(1)	Signature appears valid but root CA certificate not trusted	
		2(2)	Module signature failed verification	
		3(3)	Module certificate chain incorrect	
		4(4)	Signature required but module not signed	
		5(5)	Signature required but signature has been removed	
		6(6)	Program verification module not loaded. Program verification was not available when attempt was made to load this program.	
		7(7)	The Algorithmic self test failed while verifying the program verification module.	

Table of extended-length relocate section variable data

Data type (SMF80TP2) dec(hex)	Data length (SMF80DL2)	Format	Audited by event code	Description (SMF80DA2)
409(199)	1-255	EBCDIC	86	Root signing certificate subject's distinguished name
410(19A)	1-255	EBCDIC	86	Program signer (End Entity) certificate subject's distinguished name
411(19B)	1	Binary	86	R_PgmSignVer flags byte **Bit** **Meaning** **0** 1 = Module allowed to be loaded
412(19C)	8	EBCDIC	86	Time module was signed
413(19D)	10	EBCDIC	86	Date module was signed
414(19E)	10	EBCDIC	86	Date when module certificate chain expires

Event codes

Event code name	Col ID	Event code	Description	Where described
R_PgmSignVer	PGMV	86	Signature verification	"The R_PgmSignVer record extension" on page 78

Updates to record extensions

The R_PgmSignVer record extension has been added.

The R_PgmSignVer record extension

Table 24. Format of the R_PgmSignVer record extension (event code 86)

			Position		
Field name	**Type**	**Length**	**Start**	**End**	**Comments**
PGMV_RES_NAME	Char	255	282	536	Name of program being verified.
PGMV_VOL	Char	6	538	543	Volume containing the program.
PGMV_LOGSTRING	Char	255	545	799	Logstring parameter.
PGMV_USER_NAME	Char	20	801	820	The name associated with the user ID.
PGMV_UTK_ENCR	Yes/No	4	822	825	Is the UTOKEN associated with this user encrypted?
PGMV_UTK_PRE19	Yes/No	4	827	830	Is this a pre-1.9 token?
PGMV_UTK_VERPROF	Yes/No	4	832	835	Is the VERIFYX propagation flag set?
PGMV_UTK_NJEUNUSR	Yes/No	4	837	840	Is this the NJE undefined user?
PGMV_UTK_LOGUSR	Yes/No	4	842	845	Is UAUDIT specified for this user?
PGMV_UTK_SPECIAL	Yes/No	4	847	850	Is this a SPECIAL user?
PGMV_UTK_DEFAULT	Yes/No	4	852	855	Is this a default token?
PGMV_UTK_UNKNUSR	Yes/No	4	857	860	Is this an undefined user?
PGMV_UTK_ERROR	Yes/No	4	862	865	Is this user token in error?
PGMV_UTK_TRUSTED	Yes/No	4	867	870	Is this user a part of the TCB?
PGMV_UTK_SESSTYPE	Char	8	872	879	The session type of this session.
PGMV_UTK_SURROGAT	Yes/No	4	881	884	Is this a surrogate user?
PGMV_UTK_REMOTE	Yes/No	4	886	889	Is this a remote job?
PGMV_UTK_PRIV	Yes/No	4	891	894	Is this a privileged user ID?
PGMV_UTK_SECL	Char	8	896	903	The security label of the user.
PGMV_UTK_EXECNODE	Char	8	905	912	The execution node of the work.
PGMV_UTK_SUSER_ID	Char	8	914	921	The submitting user ID.
PGMV_UTK_SNODE	Char	8	923	930	The submitting node.
PGMV_UTK_SGRP_ID	Char	8	932	939	The submitting group name.
PGMV_UTK_SPOE	Char	8	941	948	The port of entry.
PGMV_UTK_SPCLASS	Char	8	950	957	Class of the POE.
PGMV_UTK_USER_ID	Char	8	959	966	User ID associated with the record.
PGMV_UTK_GRP_ID	Char	8	968	975	Group name associated with the record.
PGMV_UTK_DFT_GRP	Yes/No	4	977	980	Is a default group assigned?
PGMV_UTK_DFT_SECL	Yes/No	4	982	985	Is a default security label assigned?
PGMV_PDS_DSN	Char	44	987	1030	Partitioned data set name containing the program.
PGMV_UTK_NETW	Char	8	1032	1039	The port of entry network name.
PGMV_X500_SUBJECT	Char	255	1041	1295	Subject's name associated with this event.
PGMV_X500_ISSUER	Char	255	1297	1551	Issuer's name associated with this event.
PGMV_SERV_POENAME	Char	64	1553	1616	SERVAUTH resource or profile name.
PGMV_CTX_USER	Char	510	1618	2127	Authenticated user name.
PGMV_CTX_REG	Char	255	2129	2383	Authenticated user registry name.
PGMV_CTX_HOST	Char	128	2385	2512	Authenticated user host name.

Table 24. Format of the R_PgmSignVer record extension (event code 86) (continued)

Field name	Type	Length	Position		Comments
			Start	End	
PGMV_CTX_MECH	Char	16	2514	2529	Authenticated user authentication mechanism object identifier (OID).
PGMV_ROOT_DN	Char	255	2531	2785	Root signing certificate subject's distinguished name.
PGMV_SIGNER_DN	Char	255	2787	3041	Program signing certificate subject's distinguished name.
PGMV_MOD_LOADED	Yes/No	4	3043	3046	Module loaded?
PGMV_SIGN_TIME	Time	8	3048	3055	Time module was signed.
PGMV_SIGN_DATE	Date	10	3057	3066	Date module was signed.
PGMV_EXPIR_DATE	Date	10	3068	3077	Date at which module signature certificate chain will expire.
PGMV_IDID_USER	Char	985	3079	4063	Authenticated distributed user name.
PGMV_IDID_REG	Char	1021	4065	5085	Authenticated distributed user registry name.

Table 25. Event qualifiers for R_PgmSignVer callable service records

Event qualifier	Event Qualifier Number	Event description
SUCCESS	00	Successful signature verification.
NOTRUST	01	Signature appears valid but root CA certificate not trusted.
INVALSIG	02	Module signature failed verification.
INCORCHN	03	Module certificate chain incorrect.
NOTSIGND	04	Signature required but module not signed.
SIGREMOV	05	Signature required but signature has been removed.
VERNOTLD	06	Program verification module not loaded. Program verification was not available when attempt was made to load this program.
SLFTSTFL	07	Algorithmic self test failed.

Updates to database unload

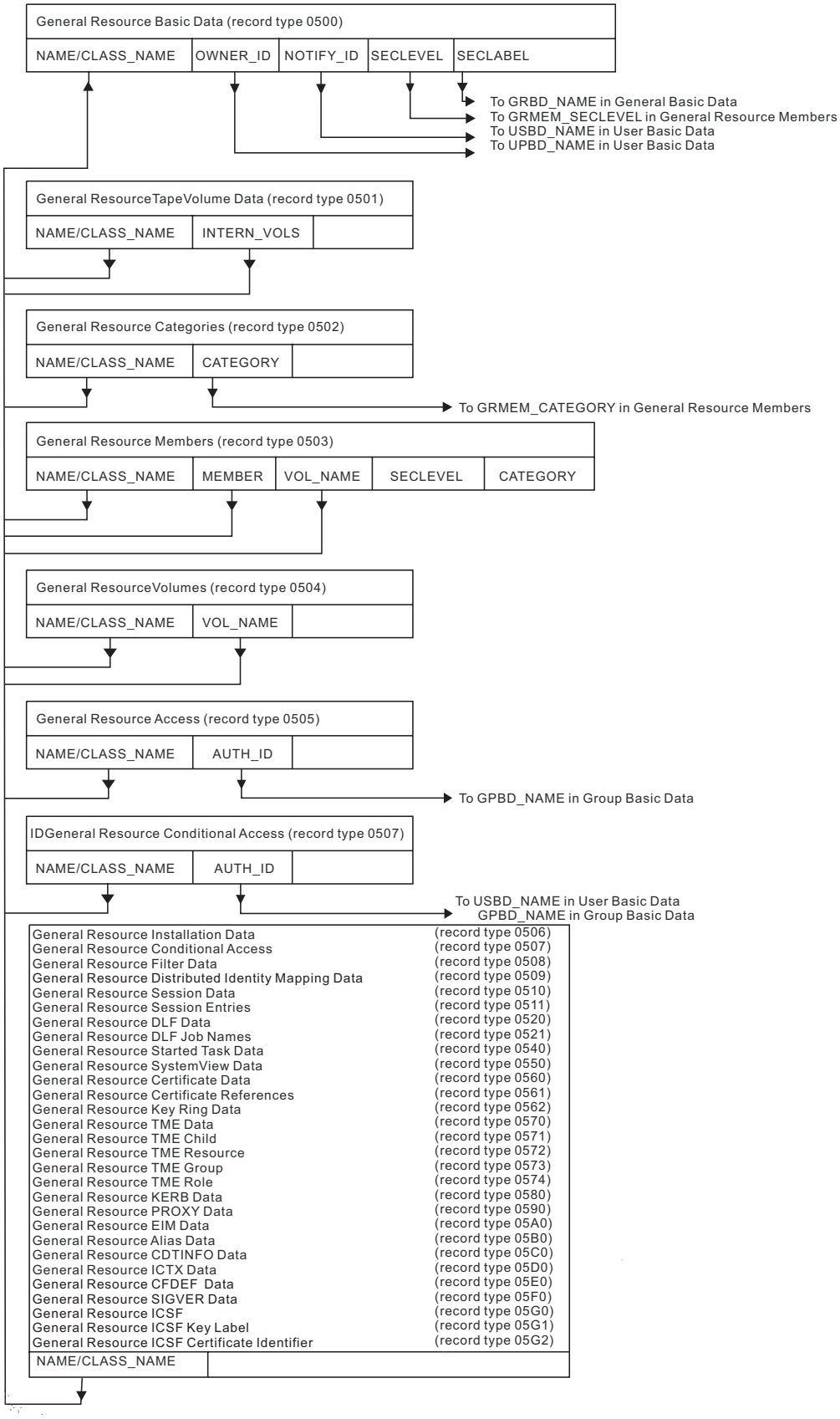
The SIGVER segment of the GENERAL template has been added:

Record Type	Record Name
05F0	General Resource SIGVER Data

Updates to the General resource records table:

Record Name	Record Type	Record Prefix
General Resource SIGVER Data	05F0	GRSIG

Updates to the Relationship among the General Resource Record Types figure:



Updates to the General resource record formats:

- General Resource SIGVER Data Record

General Resource SIGVER data record (05F0)

The General Resource SIGVER Data record (05F0) defines the settings that control program signature verification. There is one record per general resource profile that contains a SIGVER segment.

Table 26. General Resource SIGVER Data Record

		Position		
Field Name	**Type**	**Start**	**End**	**Comments**
GRSIG_RECORD_TYPE	Int	1	4	Record type of the general resource SIGVER data record (05F0).
GRSIG_NAME	Char	6	251	General resource name as taken from the profile name.
GRSIG_CLASS_NAME	Char	253	260	Name of the class to which the general resource profile belongs.
GRSIG_SIGREQUIRED	Yes/No	262	265	Signature required.
GRSIG_FAILLOAD	Char	267	276	Condition for which load should fail. Valid values are NEVER, BADSIGONLY, and ANYBAD.
GRSIG_AUDIT	Char	278	287	Condition for which RACF should audit. Valid values are NONE, ALL, SUCCESS, BADSIGONLY, and ANYBAD.

Updates to the RACF database templates

Template Field name (character data)	**Field ID**	**Flag 1**	**Flag 2**	**Field length decimal**	**Default value**	**Type**	**Field being described**
The following is the SIGVER segment of the GENERAL template.							
SIGVER	001	00	00	0	0		Start of segment fields
SIGREQD	002	00	00	1	0	Bin	Module must have a signature: **Value** **Meaning** **X'80'** Yes **X'00'** No
FAILLOAD	003	00	00	1	0	Bin	Loader failure conditions: **Value** **Meaning** **X'80'** Bad signature only **X'40'** Any failing signature condition **X'00'** Never
SIGAUDIT	004	00	00	1	0	Bin	RACF audit conditions: **Value** **Meaning** **X'80'** Bad signature only **X'40'** Any failing signature condition **X'20'** Success **X'01'** All **X'00'** None

Updates to event code qualifier descriptions

Event 86(56): R_PgmSignVer

86(56) **R_PgmSignVer**

0 Successful signature verification

1 Signature appears valid but root CA certificate not trusted

2 Module signature failed verification

3 Module certificate chain incorrect

4 Signature required but module not signed

5 Signature required but signature has been removed

6 Program verification module not loaded. Program verification was not available when attempt was made to load this program.

7 The Algorithmic self test failed while verifying the program verification module.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

DB2
IBM
MVS
RACF
UNIX
z/OS
zSeries

UNIX and X/OPEN are registered trademarks of The Open Group in the United States and other countries.

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

Program Number:

Printed in USA