Oracle® Fusion Middleware

Oracle GoldenGate Security Guide

19c (19.1.0)
E98065-06
October 2022

Oracle Fusion Middleware Oracle GoldenGate Security Guide, 19c (19.1.0)

E98065-06

Primary Author: Oracle Corporation

Contents

Audience

The *Oracle GoldenGate Security Guide* is intended for database administrators (DBAs), security administrators, application developers, and others tasked with performing the following operations securely and efficiently:

- Designing and implementing security policies to protect the data of an organization, users, and applications from accidental, inappropriate, or unauthorized actions
- Creating and enforcing policies and practices of auditing and accountability for inappropriate or unauthorized actions
- Creating, maintaining, and terminating user accounts, passwords, roles, and privileges
- Developing applications that provide desired services securely in a variety of computational models, leveraging database and directory services to maximize both efficiency and ease of use

To use this document, you need a basic understanding of how and why a database is used, and basic familiarity with SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, such as "From the File menu, select **Save**." Boldface also is used for terms defined in text or in the glossary.
italic *italic*	Italic type indicates placeholder variables for which you supply particular values, such as in the parameter statement: `TABLE` *`table_name`*. Italic type also is used for book titles and emphasis.
`monospace` `MONOSPACE`	Monospace type indicates code components such as user exits and scripts; the names of files and database objects; URL paths; and input and output text that appears on the screen. Uppercase monospace type is generally used to represent the names of Oracle GoldenGate parameters, commands, and user-configurable functions, as well as SQL commands and keywords.
UPPERCASE	Uppercase in the regular text font indicates the name of a utility unless the name is intended to be a specific case.
{ }	Braces within syntax enclose a set of options that are separated by pipe symbols, one of which must be selected, for example: `{`*`option1`* `|` *`option2`* `|` *`option3`*`}`.
[ ]	Brackets within syntax indicate an optional element. For example in this syntax, the `SAVE` clause is optional: `CLEANUP REPLICAT` *`group_name`* `[, SAVE` *`count`*`]`. Multiple options within an optional element are separated by a pipe symbol, for example: `[`*`option1`* `|` *`option2`*`]`.

Related Information

The Oracle GoldenGate Product Documentation Libraries are found at

https://docs.oracle.com/en/middleware/goldengate/index.html

Additional Oracle GoldenGate information, including best practices, articles, and solutions, is found at:

Oracle GoldenGate A-Team Chronicles

1

About Oracle GoldenGate Security

Oracle GoldenGate has integrated security features and understanding the security features and the use cases they cover are important first steps when setting up a secure environment.

There are two different architectures offered with Oracle GoldenGate:

Microservices Architecture (MA)
This is a REST API-based services architecture that allows you to configure, monitor, and manage Oracle GoldenGate services using a web interface or through REST API calls. Oracle recommends implementing MA to ensure the highest levels of security with Oracle GoldenGate.
You can use MA to deploy, monitor, manage, and perform Extract and Replicat operations on trail data within your MA implementation. To learn more about MA see Components of Oracle GoldenGate Microservices Architecture.

Classic Architecture (CA)
This is the original Oracle GoldenGate architecture to effectively move data across numerous topologies. To know more about Classic Architecture, see Components of Classic Architecture and the Oracle GoldenGate user guide for your database.

Oracle GoldenGate Microservices Architecture (MA) is most secure. This guide addresses MA-specific topics in the main chapters, while security aspects of the Classic Architecture are addressed in the appendix.

- Overview of Security Options
  You can use these security features to protect your Oracle GoldenGate environment and the data that is being processed.

1.1 Overview of Security Options

You can use these security features to protect your Oracle GoldenGate environment and the data that is being processed.

What to Secure	Security Features	Supported Databases	Supported Architecture	Description
Master Encryption Keys	Managing Data Encryption using Oracle Key Vault.	All databases	Classic and Microservices	Manages the encryption of trail files by storing the master keys.

What to Secure	Security Features	Supported Databases	Supported Architecture	Description
• Data in the trails or an Extract file • Data sent across TCP/IP networks	Encrypting Data with the Master Key and Wallet Method	Master key and wallet method is the preferred method on platforms that support it. Not valid for NonStop platforms.	X	Encrypts the data in files, across data links, and across TCP/IP. Use one of the following: • Any Advanced Encryption Security (AES) Advanced Encryption Standard (AES) is a symmetric-key encryption standard that is used by governments and other organizations that require a high degree of data security. It offers three 128-bit block-ciphers: a 128-bit key cipher, a 192-bit key cipher, and a 256-bit key cipher. The `LD_LIBRARY_PATH` value is set to `$ORACLE_HOME/lib` with a default configuration. Use the `export` command to modify this value. • Blowfish: Blowfish encryption: A keyed symmetric-block cipher. The Oracle GoldenGate implementation of Blowfish has a 64-bit block size.
User IDs and passwords (credentials) assigned to Oracle GoldenGate processes to log into a database.	Credential Store Identity Management Managing Identities in a Credential Store	Credential store is the preferred password management method on platforms that support it. Not valid for NonStop platforms.	Microservices	User credentials are maintained in secure wallet storage. Aliases for the credentials are specified in commands and parameters.
Passwords specified in commands and parameter files that are used by Oracle GoldenGate processes to log into a database.	Password Encryption See Encrypting a Password in a Command or Parameter File.	Valid for all Oracle GoldenGate-supported databases and platforms. Blowfish must be used on the DB2 for i, DB2 z/OS, and NonStop platforms. On other platforms, the credential store is the preferred password-management method.	Classic	Encrypts a password and then provides for specifying the encrypted password in the command or parameter input. Use any of the following: • AES-128 • AES-192 • AES-256 • Blowfish

What to Secure	Security Features	Supported Databases	Supported Architecture	Description
Oracle GoldenGate commands issued through GGSCI.	Command Authentication See Configuring GGSCI Command Security.	Valid for all Oracle GoldenGate-supported databases and platforms.	X	Stores authentication permissions in an operating-system-secured file. Configure a `CMDSEC` (Command Security) file.
TCP/IP connection to untrusted Oracle GoldenGate host machines that are outside a firewall.	Trusted Connection See Using Target System Connection Initiation.	Valid for all Oracle GoldenGate-supported databases and platforms.	X	Use any of the following: • AES-128 • AES-192 • AES-256 • Blowfish
Access rules for Manager.	Manager Security Securing Manager	Valid for all Oracle GoldenGate-supported databases and platforms.	Classic	You can secure the following: • `GGSCI`: Secures access to the GGSCI command-line interface. • `MGR | MANAGER`: Secures access to all inter-process commands controlled by Manager, such as `START`, `STOP`, and `KILL` • `REPLICAT`: Secures connection to the Replicat process. • `COLLECTOR | SERVER`: Secures the ability to dynamically create a Collector process.
Select the cryptographic library that better suits your needs: Portability (Classic), Portability and compliance with FIPS-140 standard (FIPS140), or enhanced throughput (Native).	CryptoEngine	Valid for all Oracle GoldenGate-supported databases and platforms (Classic and FIPS140). Valid for all Oracle GoldenGate-supported databases on Linux.x64 and Windows.x64 (Native).	Classic and Microservices	Selects which cryptographic library the Oracle GoldenGate processes will use.
MA REST Service Interface	Authentication	Valid for all Oracle GoldenGate-supported databases and platforms	Microservices	X
Communication Security	TLS and Secure Network Protocols	Valid for all Oracle GoldenGate-supported databases and platforms	Microservices	X

What to Secure	Security Features	Supported Databases	Supported Architecture	Description
MA REST User Authorization	Authorization	Valid for all Oracle GoldenGate-supported databases and platforms	Microservices	X
Target-initiated Trails	Target-initiated trails for trusted environments	Valid for all Oracle GoldenGate-supported databases and platforms	Microservices	See Using Target-Initiated Distribution Paths.
Reverse Proxy	The reverse proxy only uses one port. See Configure Reverse Proxy with NGINX to Access Oracle GoldenGate Microservices	Valid for all Oracle GoldenGate-supported databases and platforms	Microservices	X

Part I
Securing the Microservices Architecture

Use this part to secure your Microservices Architecture (MA) environment.

With Microservices, each server (Administration, Distribution, Performance Monitoring, Receiver Server and Service Manager) runs its own process and communicates with REST. As REST is a style that uses secure HTTP, all the security related concerns and solutions applied to HTTPS apply to REST interfaces also. This includes ensuring general security related to HTTPS-based requests, responses, sessions, cookies, headers and content as well as addressing issues such as Cross Site Request Forgery, UI Redressing and delegated authentication. TLS 1.2 (Transport Layer Security) provides both confidentiality and integrity with optional Authentication. Server authentication, which verifies the identity of the server used by the client for communication. Client authentication verifies the identity of the client that the server is communicating. A typical configuration enforces server authentication while client authentication is optional. Additional security configurations can specify the level of security strength and revocation options.

Inbound and Outbound Security Configuration

An **inbound configuration** defines the security characteristics used for requests being *received* by the server *from* a client; an inbound request.

An **outbound configuration** defines the security characteristics used for requests being *sent* from the server *to* a client; an outbound request.

A **server** is generally considered to be operating secured when security is enabled and the inbound configuration is valid.

All MA servers support inbound security configurations. Only the Distribution Server and Receiver Server support outbound configurations.

The Distribution Server and Receiver Server use the Outbound security configuration to secure request between them. When the Distribution Server issues a request to a Receiver Server or when a Receiver Server issues a request to a Distribution server, each uses their outbound configurations.

Topics:

- TLS and Secure Network Protocols
  Communication security is the confidentiality and integrity of the information sent over communications channels, such as TCP/IP-based networks.
- Using Target-Initiated Distribution Paths
  Learn about target-initiated distribution paths in MA, the need to set it up, and various use cases where it is helpful to use target-initiated distribution paths.

Microservices Security Concepts

Learn about these MA security features:

Connection Filtering
This is responsible for qualifying and filtering a candidate connection based on connection policy specifications.

Certificate Filtering
Similar to connection filtering, this feature enables qualifying certificates as part of accepting or denying a connection request.

Fall-back Constraints
Network security configuration within MA servers enables you to configure and constrain the protocol version negotiation fall-back behavior allowing them to control if and how the protocol versions are negotiated.

Session Management
MA Service Interfaces requests are REST and stateless, which implies that no client application context it stored on the server between requests. The application session state is entirely held by the client. Session management includes:

Logical state-tracking of the clients authorization status
The Authorization Cookie used by WebApps and available to other clients is an opaque token that allows secured client authorization information sent to the server with each REST request. The client state encoded in the Authorization Cookie is transferred automatically by the browser with each request, The client's effective authorization is not maintained by the server.

Secured TLS session caching and reuse
The secured communication sub-system supports TLS session caching and reuse. This reduces the computational load on the server by allowing cryptographic session established in a prior require to be reused and skip the high-cost handshake and cipher negotiation processing. TLS-session caching and reuse does not reuse any MA service request information.

User Credential Storage
User credentials are stored in a cryptographically secure persistent and fault-tolerant store. When you add credentials from the Admin Client, they are stored locally to the executable. This allows the Admin Client to run scripts securely from the local site.

Single Page Applications (SPAs) and WebApp Security
All popular web browsers support both HTTP and HTTPS protocol. MA supports running WebApps including SPAs and JavaScript-based applications in either HTTPS (secured) or HTTP (unsecured) mode.

Cipher-Suites
The MA configuration allows you to select the set of allowed cipher-suites if necessary. Generally, the MA default cipher-suite set is appropriate.

Encryption Profile
The encryption profile allows you to use Oracle Key Vault, which is a full-stack, security-hardened software appliance built to centralize the management of MA security objects.

2
Securing Deployments

Microservices REST-based Service Interfaces are agnostic with regard to which underlying HTTP or HTTPS protocol is used. Their behavior is the same whether issued over a secure or an unsecure protocol.

Securing deployments involves enabling security through the security configuration when setting up a deployment for the first time using Oracle GoldenGate microservices. Administrators who are assigned the security role can change the details of the default MA security profile to control various aspects of secure operation. See How to Add Users 19c in *Using the Oracle GoldenGate Microservices Architecture* to know more about the security user role.

MA's default security configuration is ranked at a high-medium security level. By enabling security for an MA deployment, the default coordinated security profile from both inbound and outbound communications is enabled (excluding the client and server wallet location (WRL).

To secure a deployment, you can use your existing wallets and certificates, or create new ones. See Setting Up Secure and Non-Secure Deployments in *Using the Oracle GoldenGate Microservices Architecture* for more information.

- Creating a Self-Signed Root Certificate
  In a secure mode, communication with Oracle GoldenGate MA including administrative calls and data transport is secured using TLS certificates, which you purchase or create your own for testing purposes.
- Creating Server Certificates
- Creating a Distribution Server User Certificate
  You have the option of using a client certificate or a username and password that is common to the Distribution Server deployment and the Receiver Server deployment.
- Trusted Certificates
  The `wss` communication protocol is used in the Distribution Server for the Distribution Path to meet the needs of secure communication using TLS in Oracle GoldenGate Microservices Architecture.

2.1 Creating a Self-Signed Root Certificate

In a secure mode, communication with Oracle GoldenGate MA including administrative calls and data transport is secured using TLS certificates, which you purchase or create your own for testing purposes.

In production environments, it is strongly recommended to use commercial certificates. In test environments, you may create your own self signed certificates using `orapki` or OpenSSL.

Each secure Oracle GoldenGate deployment requires two certificates: Server certificate for the Oracle GoldenGate services and a client certificate used by the distribution and/or receiver server to securely communicate with other remote deployments It is key that all certificates are signed by the same root certificate authority (rootCA). rootCA is the trustpoint. You use only one root certificate authority (rootCA) for all certificates across the deployment.

For each server where Oracle GoldenGate is deployed, you have one specific Server Certificate.

You may apply your existing root certificate or use the `orapki` in the `OGG_HOME/bin` directory, see About the orapki Utility in the *Oracle Database Security Guide*.

Here's an example of how you can create a root certificate using `orapki`:

1. Create a directory to store your wallets and certificates. For example, `~/wallet_directory`.
2. Create an automatic login wallet. This example uses `root_ca` for the wallet name.

   ```
   orapki wallet create -wallet ~/wallet_directory/root_ca -auto_login
   -pwd welcome123
   ```

3. In the `orapki` command to create self-signed (root user) certificate, specify the `-sign_alg sha256` option.
4. In `orapki` wallet:

   ```
   add -wallet ~/wallet_directory/root_ca -dn "CN=RootCA" -keysize
   2048 -self_signed -validity 7300 -pwd welcome123 -sign_alg sha256
   ```

5. Export the certificate to a `.pem` file.

   ```
   orapki wallet export -wallet ~/wallet_directory/root_ca -dn
   "CN=RootCA" -cert ~/wallet_directory/rootCA_Cert.pem -pwd welcome123
   ```

The wallet creation is complete.

2.2 Creating Server Certificates

The following steps are an example of how you can create a sever certificate using a root certificate named `root_ca`.

1. Create a directory to store your wallets and certificates. For example, `~/wallet_directory`.
2. Create an automatic login server wallet.

   ```
   orapki wallet create -wallet ~/wallet_directory/$(hostname) -
   auto_login -pwd welcome123*
   ```

   Enter the password for the server when prompted.
3. Add a Certificate Signing Request (CSR) to the server's wallet.

   ```
   orapki wallet add -wallet ~/wallet_directory/$(hostname) -dn "CN=$
   (hostname)" -keysize 2048 -pwd welcome123
   ```

4. Export the CSR to a `.pem` file.

```
orapki wallet export -wallet ~/wallet_directory/$(hostname) -dn "CN=$
(hostname)" -request ~/wallet_directory/servername_req.pem -pwd welcome123
```

5. Using the CSR, create a signed server or client certificate and sign it using the root certificate. Assign a unique serial number to each certificate.

```
orapki cert create -wallet ~/wallet_directory/root_ca -request ~/
wallet_directory/servername_req.pem -cert ~/wallet_directory/
servername_Cert.pem -serial_num 20 -validity 375  -sign_alg sha256
```

6. Add the root certificate into the client's or server's wallet as a trusted certificate.

```
orapki wallet add -wallet ~/wallet_directory/$(hostname) -trusted_cert -
cert ~/wallet_directory/rootCA_Cert.pem -pwd welcome123
```

7. Add the server or client certificate as a user certificate into the client's or server's wallet.

```
orapki wallet add -wallet ~/wallet_directory/$(hostname) -user_cert -cert
~/wallet_directory/servername_Cert.pem  -pwd welcome123
```

The wallet creation is complete.

2.3 Creating a Distribution Server User Certificate

You have the option of using a client certificate or a username and password that is common to the Distribution Server deployment and the Receiver Server deployment.

This certificate is also signed by the root certificate. It provides a common trust point because the server considers any certificate signed by the same root certificate as the server's certificate. To create the certificate, use the `orapki` in the `OGG_HOME/bin` directory. For more information about `orapki`, see About the orapki Utility in the *Oracle Database Security Guide*.

The following steps are an example of how you can create a distribution sever user certificate:

1. Create a directory to store your wallets and certificates. For example, `~/wallet_directory`.
2. Create an automatic login client wallet. This example uses `dist_client` for the wallet name.

```
orapki wallet create -wallet ~/wallet_directory/dist_client -auto_login -
pwd welcome123
```

3. Add a CSR to the wallet.

```
orapki wallet add -wallet ~/wallet_directory/dist_client -dn
"CN=dist_client" -keysize 2048 -pwd welcome123
```

4. Export the CSR to a `.pem` file.

```
orapki wallet export -wallet ~/wallet_directory/dist_client -dn
"CN=dist_client" -request ~/wallet_directory/dist_client_req.pem -
pwd welcome123
```

5. Using CSR, create a signed server or client certificate and sign it using the root certificate. Assign a unique serial number to each certificate.

```
orapki cert create -wallet ~/wallet_directory/root_ca -request ~/
wallet_directory/dist_client_req.pem -cert ~/wallet_directory/
dist_client_Cert.pem -serial_num 30 -validity 375 -pwd welcome123
```

6. Add the root certificate as a trusted certificate into the client's or server's wallet.

```
orapki wallet add -wallet ~/wallet_directory/dist_client -
trusted_cert -cert ~/wallet_directory/rootCA_Cert.pem -pwd
welcome123
```

7. Add the server or client certificate as a user certificate into the client's or server's wallet.

```
orapki wallet add -wallet ~/wallet_directory/dist_client -user_cert
-cert ~/wallet_directory/dist_client_Cert.pem -pwd welcome123
```

The wallet creation is complete.

2.4 Trusted Certificates

The `wss` communication protocol is used in the Distribution Server for the Distribution Path to meet the needs of secure communication using TLS in Oracle GoldenGate Microservices Architecture.

There are two types of TLS connections and to use TLS, there are certain requirement for the certificate trust chain.

Distribution Server and Receiver Server

Both the Distribution Server and Receiver Server need certificates. The Distribution Server uses the certificate in the client wallet location under outbound section. The location of that wallet can be found in the `deploymentConfiguration.dat` file under *deployment_home*`/etc/conf`.

The certificates in both wallets need to be trusted by each other, so either both need to have commercial certificates issued by Classic architecture, or they have to trust each other for self-signed certificates.

For self-signed certificates, you can choose from one of the following:

- Have both certificates signed by the same Root Certificate.
- The other side's certificate is added to the local wallet as trusted certificate

Here's an example that shows the Distribution Server and Receiver Server certificates.

```
"distsrvr": {
        "$schema": "ogg:service",
        "config": {
            "network": {
                "serviceListeningPort": 9102
            },
            "authorizationDetails": {
                "common": {
                    "allow": [
                        "Digest",
                        "x-Cert",
                        "Basic"
                    ]
                }
            },
            "authorizationEnabled": true,
            "workerThreadCount": 24,
            "legacyProtocolEnabled": true,
            "taskManagerEnabled": true,
            "security": false,
   //The following is the outbound communication setup.
            "securityDetails": {
                "network": {
                    "outbound": {
                        "authMode": "client_server",
                        "crlEnabled": false,
                        "role": "client",
                        "wrl": "file:/u02/ogg/dpora12c/etc/ssl/740977c539e7",
                        "wrlPassword": ""
}
```

In this example, the section that starts with securityDetails is for the outbound communication setup. The WRL values gives wallet location.

For the Receiver Server, the certificate is in the wallet for the inbound wallet location, which is in the same `deploymentConfiguration.dat` file, as shown in the following example.

```
…
  "recvsrvr" : {
     "$schema" : "ogg:service",
     "config" : {
        "authorizationDetails" : {
           "common" : {
              "allow" : [ "Basic", "x-Cert" ]
           }
         },
         "authorizationEnabled" : true,
         "legacyProtocolEnabled" : true,
         "network" : {
            "serviceListeningPort" : 10083
         },
         "security" : true,
```

```
          "securityDetails" : {
             "network" : {
                "common" : {
                  "authMode" : "clientOptional_server",
                   "blockSize" : 4096,
                   "cipherSuites" : [
                      "SSL_RSA_WITH_3DES_EDE_CBC_SHA",
                      "TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256",
                      "TLS_ECDHE_ECDSA_WITH_RC4_128_SHA",
                      "TLS_RSA_WITH_AES_256_GCM_SHA384"
                   ],
                   "crlEnabled" : false,
                   "crlStore" : "file:",
                   "id" : "OracleSSL",
                   "protocolVersion" : "1_2_Or_1_1_Or_1_0_Or_3_0"
                },
                "inbound" : {
                   "role" : "server",
                   "wrl" : "file:/home/oracle/apps/OlnxSRCSSL/etc/ssl/
OlnxSRC",
                   "wrlPassword" : ""
                },
                "outbound" : {
                   "role" : "client",
                   "wrl" : "file:/home/oracle/apps/OlnxSRCSSL/etc/ssl/
oggdistclient",
                   "wrlPassword" : ""
                }
```

On the Distribution Server, if the hostname used in the Receiver Server's certificate can't be routed correctly, `/etc/hosts` file should be updated with the correct IP address for that host. The Distribution Server will use this IP address to communicate with the Receiver Server once it accepts the certificate from the Receiver Server.

Using the Reverse Proxy (NGINX) with the Distribution Server and Receiver Server

You only need to add the Nginx certificate to the Distribution server's client wallet as a trusted certificate. Usually the certificate used by Nginx is self-signed. If it is issued by Classic architecture, then there is no need to perform this step.

The host name in the NGINX certificate should also be routable. If not, on the Distribution Server, `/etc/hosts` file needs to be updated to reflect the correct IP address for that host name.The Distribution Server will use the host name in the certificate to communicate to the target. If the NGINX certificate doesn't have a valid host name in it, but has a Subject Alternative Name record, then the host name is the DNS name there. For example:

```
…X509v3 Subject Alternative Name:
DNS:localhost,
DNS:oggmp0802iad,
DNS:oggmp0802iad.sub06261535551.wernervcnab.oraclevcn.com,
DNS:127.0.0.1, IP Address:127.0.0.1…
```

3
Authentication and Authorization

The MA security defines the communication authorization and authentication. Authentication includes tasks such as configuring the credential store and aliases for scripts in the AdminClient. Authorization includes tasks for network and server configuration.

All the security configurations and services are common to MA-based servers. These servers authenticate, authorize, and secure access to command and control, monitoring, data conveyance, and information service interfaces for the MA.

Oracle GoldenGate Microservices define an infrastructure for building service-aware applications to operate and integrate into global, cloud-based deployment environments. Oracle GoldenGate server programs are implemented using the microservices infrastructure. All security and configuration implementations provided by MA are common services.

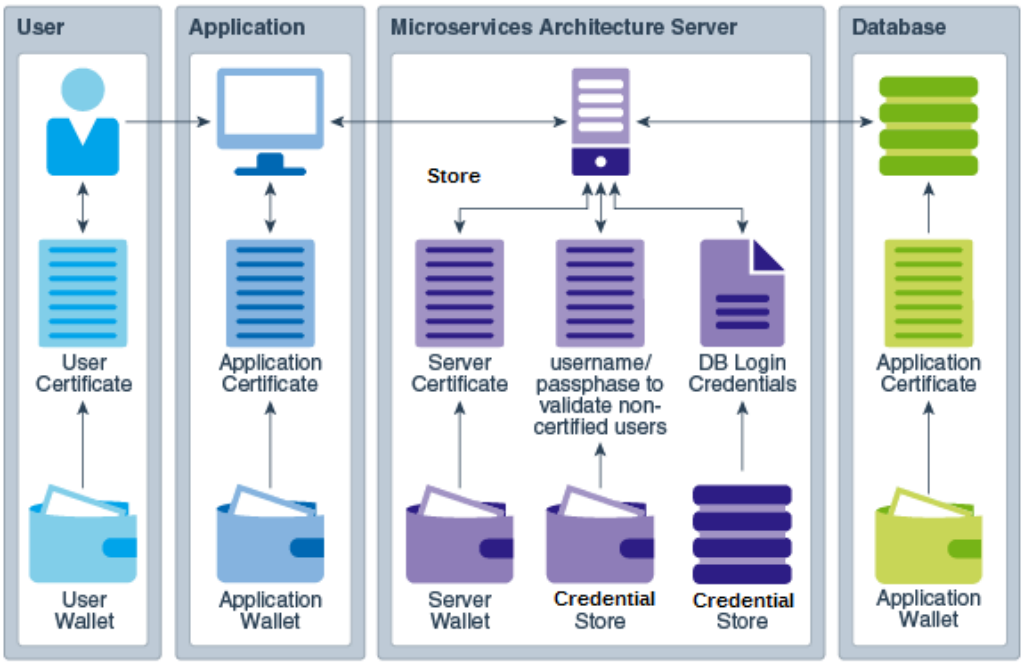
- Authentication
  Learn how you can use identity authentication.
- Authorization
  Learn how you can use authorization modes.
- Authentication and Authorization for WebSockets
  Learn how you can use WebSocket authentication and authorization.
- Response Status Codes
  A few of the MA HTTPS authentication and authorization error codes are:

3.1 Authentication

Learn how you can use identity authentication.

The goal of the authenticated identity design is to establish identity authentication between users, an MA server or application, and an MA server. The authentication design relies on either the validity of a certificate or of a user credential (`username` and `password` pair).

The MA servers publish REST service interfaces that enable users and applications to request services including operational control over one or more MA deployments, service administration, status, and performance monitoring.



The following types of certificates are used for authentication:

- **User Certificate**: A User Certificate is a certificate issued to a specific user. Oracle GoldenGate client applications store the User Certificate in a user Oracle Wallet. The default location of the user Oracle Wallet is under the user's home directory. Service requests issued with User Certificates include the user name and group information acquired from the host environment. This information identifies the real user executing the application.
- **Application Certificate**: An Application Certificate is a certificate issued to a specific application. The Application Certificate is stored by the application. Oracle GoldenGate client applications store the Application Certificate in an application Oracle Wallet designated by the Application configuration. The default location of the application Oracle Wallet is in the *$OGG_SSL_HOME* directory.
- **Server Certificate**: A Server Certificate is a certificate issued to a specific MA server. The Server Certificate is stored by the MA server in the server's Oracle Wallet. The default location of the server Oracle Wallet is under the server's installation directory. An MA server is authenticated to applications as the identity described in the Server Certificate.
- **User's or Application's Database Authentication**: MA servers support Service Interface request whose fulfillment requires logging into a source or target database. Database actions from an MA Server are limited to specific operations required to fulfill service request requirements. The following table describes the type of authentication that are supported by MA servers:

Type of Authentication	Description
OS Authentication	This configuration sets the OS server to establish connections to the database using its own credentials as the only authenticated user. All service requests requiring database access use the MA server database session.

	Database operations are logged as originating from the MA
OS authentication with database proxy support	This type sets the MA server to establish connections to the database using its own credentials but support proxy user sessions, through an MA server authenticated connection. Proxy support is configured using: User Name or Distinguished Name.
Pass-thru database authentication	This configuration sets the MA server to establish a session or connection to the database using the client provided user name and password.
User-alias database authentication	This configuration sets the MA server to establish a session or connection to the database using a client provided alias ID that is mapped to a credential, held by the MA server, to establish a session or connection to the database.

Example: Using Oracle UTL_HTTPS Authentication as a Client

In database sharding, the user and application authentication model also applies to database packages that support issuing REST Server Interface requests to MA servers. Depending on the security configuration of the MA server, packages or procedures that use the `UTL_HTTPS` Oracle Database package may need to configure the client database security environment to enable the use of Client-side certificates for authentication in UTL_HTTPS. Self-written applications can similarly use UTL_HTTPS for the authentication as a client.

To enable `UTL_HTTPS` to use client-side certificates:

1. Configure the database client Oracle Wallet, see Creating the Wallet and Adding a Master Key.
2. Configure `UTL_HTTPS` with TLS (SSL) for client-side authentication, see Using UTL_HTTPS.

Certificate Revocation List Authentication Support

MA servers supports Certificate Revocation List (CRL) checks as part of the authentication process. Although MA servers do not automatically query for updated CRLs, the MA infrastructure supports updating server CRL information at runtime without requiring the MA servers to restart, see TLS Certificate Revocation List Handling.

3.2 Authorization

Learn how you can use authorization modes.

Security Authentication Modes

The following is the list of supported security authentication modes that establish the authenticity of the entity presenting the authorization information. These are the available values that may be used when setting the `/config/securityDetails/network/common/authMode` security setting. This configurattion is available from the REST APIs to the metadata catalog. This mode is set when configuring an Oracle GoldenGate MA deployment.

See the Update Service Properties in the *Oracle GoldenGate REST API* guide.

Authorization Mode ID	Notes
`server_only`	Only validate Server certificates. The Server certificates are required. The Client certificates are ignored.
`client_server`	This is the default. Validate both Client and Server certificates. Both certificates are required.
`clientOptional_server`	Validate the client certificate if it is present, as it is optional. Validate the server certificate (it's mandatory).

User Privileges

You can configure these security roles for users from the Administration Server, see Setting Up Secure or Non-Secure Deployments.

Role ID	Privilege Level
Security	Grants administration of security related objects and invoke security related service requests. This role has full privileges.
Administrator	Grants full access to the user, including the ability to alter general, non-security related operational parameters and profiles of the server.
Operator	Allows users to perform only operational actions, like starting and stopping resources. Operators cannot alter the operational parameters or profiles of the MA server.
User	Allows information-only service requests, which do not alter or affect the operation of either the MA. Examples of query and read-only information include performance metric information and resource status and monitoring information.

> **Note:**
>
> These are authorization privileges and are not directly related to authentication.

3.3 Authentication and Authorization for WebSockets

Learn how you can use WebSocket authentication and authorization.

REST API calls are made using standard HTTPS request and take advantage of the authorization mechanism described in the Hypertext Transfer Protocol (RFC2616). The WebSocket protocol (RFC6455) is different because it is a streaming-like interface

so does not need authorization or require special handling. WebSockets can be governed with the standard HTTPS authentication and authorization mechanism.

Native HTTPS Authorization

The WebSocket handshake uses the HTTP upgrade header to change from the HTTP protocol to the WebSocket protocol. The MA server checks the authorization header to approve or deny the request based on whether the role associated with the requesting user is equal to or greater than the role assigned for WebSockets establishment requests.

Example 3-1

```
GET /chat HTTPS/1.1
Host: myserver.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: HTTPS://myserver.com
Sec-WebSocket-Protocol: ogg
Sec-WebSocket-Version: 13
Authorization: Basic xgfDE24sDwrasdbliop875ty=
```

3.4 Response Status Codes

A few of the MA HTTPS authentication and authorization error codes are:

The following response status codes are used:

- 1xx: Informal
- 2xx: Success
- 3xx: Redirect
- 4xx: Client Error
- 5xx: Server Error

The 4xx client response status is described as follows:

401 Unauthorized
Returned in all cases when the presented credential is poorly formed or missing when required. This includes incorrectly spelled or unregistered user names when presented as part of an authorization credential. It does not apply to authorization resources (404 errors).

403 Forbidden
Returned in all cases when the presented credential is well-formed, but is invalid or does not have sufficient authorization (permissions) to grant access to the underlying resource.

404 Not Found
Returned in cases where the presented credential is well-formed, but the server-side resource cannot be located.
For example, when attempting to retrieve user information using `/services/v2/authorizations/all/james` and the user james is not a registered user. Without a proper registration, no `james` resource exists so this error code is returned.

The full list is found in the Internet Engineering Task Force RFC 7231 standard.

4
Network

Learn how to secure your network for Oracle GoldenGate.

This chapter describes endpoint protection such as Network Access Control and Network Connection Adapters along with the steps to configure and use Reverse Proxy.

Topics:

4.1 Network Access Control

The MA configuration of the network connection takes the form of an array or network access control list (ACL).

All configurable ACLs are represented as an array of ACL specifications. In the JSON configuration this array takes the form:

```
aclArray := '[' <aclSpec> [, <aclSpec>] ']'
```

There might be cases where access from specific sides is excluded or access from specific sides is allowed. Within Oracle GoldenGate, you are able to adjust the Application network setting so that you can create black or white lists for access points. Similarly, there might be cases, that you want to have the distribution path working on a specific network adapter only. This is feasible within Oracle GoldenGate.

Each ACL specification minimally consists of a permission statement indicating whether the ACL specification allows or denies client connections from the specified address. ACL specifications are processed in order and terminate when the specified address is qualified. If the specified address does not qualify, processes continue with the next ACL specification. Once the address of the client requesting connection is qualified, the ACLs permissions dictate whether the connection is 'allowed' or 'denied'. If the ACL specifications qualify

address of the client requesting connection, a default resolution of 'allow is assumed and the client is allowed to connect. The ACL in the configuration take the following syntactic forms:

```
ipACL := '[' aclSpec [, aclSpec] ']'


aclSpec := "permission" : [ "deny" | "allow" ] [, "address":
[ ipv4Address | ipv4MappedAddress | ipv6Address ] ]
```

The ipACLs can use IPv4 addresses, ipv6 addresses and IPv4 mapped addresses as described in RFC 4291.

Inbound connection requests are processed uniformly after they are received over a network interface. The network interface configuration dictates the form of addressing. For example, addresses appearing on an IPv6 interface appears as IPv6 addresses. If the IPv6 configuration specifies IPv4 mapping, then the IPv4 client's address is mapped into the IPv6 addressing space. An address appearing on an IPv4 interface appears as an unmapped IPv4 address. Since the ACL qualification focuses on qualifying addresses and all adapters within the host environment have unique addresses, no additional interface information is required.

For hosts that support hot-fail over network interfaces, the fail-over and reassignment of network IP address to adapter MAC addresses is transparent to the application.

Example 4-1 Examples

Deny client connections originating from 192.0.2.254.

```
"ipACL" : [ { "permission" : "deny",  "address" : "192.0.2.254" } ]
```

Explicitly allow all client connections. The first ACP by default qualifies all addresses. The second ACL is never processed.

```
"ipACL" : [ { "permission" : "allow" },
            { "permission" : "deny",  "address" : "192.0.2.254" } ]
```

Allow client connections originating from 127.0.0.1, but deny connection originating from 192.0.2.254 appearing on an interface configured for IPv6 addressing.

```
"ipACL" : [ { "permission" : "allow", "address" : "127.0.0.1" },
            { "permission" : "deny",  "address" :
"ff::192.0.2.254" } ]
```

Example 4-2 Example

Allow client connections originating from and IPv6 loopback address (127.0.0.1 represented as ::1 in IPV6 addressing), allow client connections originating from the unmapped IPv4 address 192.0.2.253, allow client connections originating from IPv6

address 2001:db8:85a3:0:0:8a2e:370:7334 and deny client connections originating from mapped IPv4 address ff::192.0.2.254.

```
"ipACL" : [ { "permission" : "allow", "address" : "::1" },
             { "permission" : "allow", "address" : "192.0.2.254" },
             { "permission" : "allow", "address" :
"2001:db8:85a3:0:0:8a2e:370:7334" },
             { "permission" : "deny",  "address" : "ff::192.0.2.254" } ]
```

Example 4-3 Example

Using REST API Calls

```
{
    "$schema":"api:standardResponse",
    "links":[
        {
            "href":"http://localhost:11000/services/v2/deployments/Local/
services/adminsrvr",
            "mediaType":"application/json",
            "rel":"canonical"
        },
        {
            "href":"http://localhost:11000/services/v2/deployments/Local/
services/adminsrvr",
            "mediaType":"application/json",
            "rel":"self"
        }
    ],
    "messages":[
    ]
}
```

Example 4-4 Example

Using cURL:

Check initial configuration:

```
curl -s -k -u ggsca:ggsca -X GET https://abc.us.oracle.com:9100/services/v2/
deployments/depl_01/services/adminsrvr | json_reformat
```

Modify service properties:

```
curl -s -k -u ggsca:ggsca -X PATCH https://abc.us.oracle.com:9100/
services/v2/deployments/depl_01/services/adminsrvr' -H Cache-Control: no-
cache' -d @"admin_2.json" |
        json_reformat
```

Admin.json file:

```
-- 8< -- File Content from admin.json -------
    {"config": {
            "network": {
                "ipACL": [
                    {
                        "address": "10.196.9.33 ",
                        "permission": "allow"
                    },
                    {
                        "address": "10.90.136.97",
                        "permission": "allow"
                    },
                    {
                        "address": "10.209.243.80",
                        "permission": "deny"
                    }
                ],
                "serviceListeningPort": 9101
            }
        }
    }
```

Example 4-5 Example

Using `oggServiceConfig`:

```
-- Check initial configuration
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca --path /network
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca --path /network/ipACL


-- Modify Service Properties
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca --path /network/ipACL --value
'[{ "permission" : "allow", "address" : "10.196.9.33 " },
{ "permission" : "allow", "address" : "10.90.136.97" },
{ "permission" : "deny",  "address" : "10.209.243.80" } ]'

    Current value of "/network/ipACL" for "depl_01/adminsrvr" is <not
defined>.
    Setting new value and restarting service.
    New value of "/network/ipACL" for "depl_01/adminsrvr" is

-- Check final configuration
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --
user ggsca --password ggsca --path /network
```

```
oggServiceConfig  https://abc.us.oracle.com:9100 depl_01 adminsrvr  --user
ggsca --password ggsca --path /network/ipACL

    [
    {
        "address": "10.196.9.33 ",
        "permission": "allow"
    },
    {
        "address": "10.90.136.97",
        "permission": "allow"
    },
    {
        "address": "10.209.243.80",
        "permission": "deny"
    }
    ]
ipACL   := '[' aclSpec [, aclSpec] ']'
aclSpec :=   "permission" : [ "deny" | "allow" ]
          [, "address": [ ipv4Address | ipv4MappedAddress | ipv6Address ] ]
```

4.2 Network Connection Adapter

Learn about how to specify your network connection configuration.

Network Interface Use Control as Network Connection Adapter is the name of an internal implementation class. When there are more than one network interface that are configured in an environment where the host is multi-homed, then it is known as multiple networks. For example, handling connection requests on different addresses through different network interface adapters.

The `NetworkConnectionSpecs` themselves are members of an array associated with the `serviceListeningPort` configuration element. For example, using the `serviceListeningPort` configuration entry, a network specification may take any of the following syntactic forms:

```
1. portValue |  portValueString
2. networkSpec
3. '[' networkSpec [, networkSpec ...] ']'
```

You can use the following syntax in your network specification:

```
  portValue         := [1234567890]+
  portValueString   := '"' portValue '"'
  networkSpec       := '{' portSpec [, ipaddressSpec | nameSpec] [,
interfaceSpec] [, networkOptionSpec] '}'
  portSpec          := "port" : portValue |  portValueString
  ipaddressSpec     := "address" : ipv4Address | ipv6Address | "ANY"
  nameSpec          := '"' :alphanum: '"'
  interfaceSpec     := "interface" : '"' :alphanum: '"'
  networkOptionSpec := "options" : IPV4_ONLY | IPV6_ONLY
```

Regardless of the form your specification takes, the internal representation is normalized into the 3rd form:

```
1. portValue | portValueString == networkSpec
2. portValue                    == '{' "port" : portValue '}'
3. portValueString              == '[' '{' "port" : portValueString
'}' ']'
```

The first form retains compatibility with existing network port specifications where only the *portValue* or *portValueString* is provided.

The second form assigns the *networkSpec* as a single value. This form still only defines a single network specification and allows greater control and flexibility in identifying network values and options.

The third form defines an array of *networkSpec* instances. It allows you to specify different network configurations based upon either address or network interface.

Example 4-6 Example

With the following simplified host network interface configuration:

```
$/sbin/ip addr show
lo: LOOPBACK,UP,LOWER_UP mtu 16436 qdisc noqueue state UNKNOWN
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
eth0: BROADCAST,MULTICAST,UP,LOWER_UP mtu 1500 qdisc pfifo_fast state
UP qlen 1000
      link/ether 00:16:3e:52:6e:27 brd ff:ff:ff:ff:ff:ff
      inet 192.0.2.39/21 brd 10.240.111.255 scope global eth0
      inet6 2001:db8:85a3:0:0:8a2e:370:6666 brd ff02::1 scope link eth0
eth1: BROADCAST,MULTICAST mtu 1500 qdisc pfifo_fast state UP qlen 1000
      link/ether 00:16:3e:1f:99:bc brd ff:ff:ff:ff:ff:ff
      inet 192.0.2.98/21 brd 10.100.99.98 scope link eth1
      inet6 2001:db8:85a3:0:0:8a2e:370:7334 brd ff02::1 scope link eth1
      inet6 2001:db8:85a3:0:0:8a2e:370:6666
```

The following specification is derived:

```
1. "serviceListeningPort: "9000"


2. "serviceListeningPort:  9000


3. "serviceListeningPort: { "port" :  9000  }



4. "serviceListeningPort: { "port" : "9000" }


5. "serviceListeningPort: { "port" : "9000",   "address" : "192.0.2.254" }


6. "serviceListeningPort: { "port" : "9000", "name" : "server1" }


7. "serviceListeningPort: { "port" : "9000", "interface" : "eth1"}


8. "serviceListeningPort: [
      { "port" : "9000", "interface" :
"lo"                                    }
      { "port" : "9000",   "address" : "192.0.2.39", "option" : "IPV4_ONLY" }
      { "port" : "9000", "interface" : "eth1",       "option" : "IPV6_ONLY" }
```

These forms are described as:

Form 1 - 4
Listens on port 9000 on `ANY` address over `ALL` interfaces.

Form 5
Listens on port 9000 on address 192.0.2.254 only.

Form 6
Listens on port 9000 on the address associates with `server1`.

Form 7
Listens on port 9000 on the address associates with interface `eth1` and accepts IPV4 address connections using the mapped IPV4.

Form 8
Listens on port 9000 on the address associates with interface `lo`, on port 9000 address 192.0.2.39 accepting only IPV4 addresses, and on port 9000 with addresses associated with interface `eth1` accepting onlyIPV6 addresses.

Most of this logic handles selecting the network interface adapter based on the network interface adapter's identifying name or the address. The interface can be searched for based on the requested address.

Specifying multiple adapters means that each network specification resolves to only a subset of adapters. Precedence processing allows the specification of `ANY` address and `ALL` interfaces for the last network specification as a pool specification when the platform networking interfaces support mapping subset interface matches.

4.3 Configure Reverse Proxy with NGINX to Access Oracle GoldenGate Microservices

Learn how to configure reverse proxy service using NGINX for accessing Oracle GoldenGate Microservices without using port numbers.

Reverse proxy enables accessing microservices using one single port (443) in a deployment. This enables encapsulation of the URL for microservices over an unsecure deployment.

> **Note:**
>
> Reverse proxy is optional, however, Oracle recommends that you ensure easy access to microservices and provide enhanced security.

You can run microservices in an unsecure deployment on loopback address and front it with an HTTP reverse proxy using the NGINX installation.

When sending trail files from Oracle GoldenGate Classic to Microservices environment that is configured with a reverse proxy, use a pump Extract from Oracle GoldenGate Classic with `SOCKSPROXY` option. When sending trail files from Oracle GoldenGate Microservices to Classic Architecture use the `ogg` protocol in the Distribution Service configuration.

See Connecting Classic to MA and Connecting MA to Classic in *Administering Oracle GoldenGate*.

Reverse Proxy Support

You can configure Oracle GoldenGate Microservices Architecture to use a reverse proxy. Oracle GoldenGate MA includes a script called `ReverseProxySettings` that generates configuration file for only the NGINX reverse proxy server.

For example, the Administration Service is available on `http://goldengate.example.com:9001` and the Distribution Service is on `http://goldengate.example.com:9002`. With reverse proxy, each of the microservices can simply be accessed from the single address. For example, `http://goldengate.example.com/distsrvr` for the Distribution Service. The URL is different for each service and is by name instead of by port.

You can use these options by running the `ReverseProxySettings` utility. Here are the options available with this utility:

`-o` or `--output`
The output file name. The default file name is `ogg.conf`.

`-P` or `--password`
A password for a Service Manager account.

`-l` or `--log`
Log file name and initiates logging. The default is no logging.

`--trailOnly`
Configure only for inbound trail data.

`-t` or `--type`
The proxy server type. The default is Nginx.

`-s` or `--no-ssl`
Configure without SSL.

`-h` or `--host`
The virtual host name for reverse proxy.

`-p` or `--port`
The reverse proxy port number. The defaults are 80 or 443.

`-?` or `--help`
Display usage information.

`-u` or `--user`
Name of the Service Manager account to use.

`-v` or `--version`
Displays the version.

These values are used when connecting to the Service Manager and are required when authentication is enabled.

You can use any reverse proxy service with MA. The following example provides a process that you can follow to configure other reverse proxy services in conjunction with the documentation for your proxy server.

Prerequisites

The following prerequisites provide details on the minimum requirements to configure an NGINX Reverse Proxy. Similar requirements may be required for your environment and reverse proxy if not using NGINX. Consult the documentation for your reverse proxy.

1. Install NGINX, see Install the NGINX Web Server and Proxy on Oracle Linux. For Oracle Linux, the command to install NGINX is:

   ```
   yum –y install NGINX
   ```

2. Check the JRE version to be JRE 8 or higher.
3. Install Oracle GoldenGate MA.
4. Create one or more active MA deployments.
5. Ensure that the Oracle user has `sudo` permissions.
6. Configure the `PATH` environment variable to include the NGINX installation directory path.

Configuring NGINX Reverse Proxy

An Oracle GoldenGate MA installation includes the `ReverseProxySettings` utility. The `ReverseProxySettings` utility is located in the *`$OGG_HOME`*`/lib/utl/reverseproxy` directory. To identify additional commands that can be used with the `ReverseProxySettings` utility, run the utility with the `--help` option:

```
$OGG_HOME/lib/utl/reverseproxy/ReverseProxySettings --help
```

To add the NGINX certificate to the Distribution Service's client wallet as a trusted certificate, see Trusted Certificates.

1. To generate a configuration file for NGINX Reverse Proxy, navigate to the location of the ReverseProxySettings utility:

   ```
   cd $OGG_HOME/lib/utl/reverseproxy
   ```

2. Run the ReverseProxySetting utility:

   ```
   ReverseProxySettings -u adminuser -P adminpwd -o ogg.conf http://
   localhost:9100
   ```

   In this code snippet, *`adminuser`* is the deployment user name and *`adminpwd`* is the deployment user password used to login to the deployment.

3. Replace the existing NGINX configuration with the configuration that was generated using the `ReverseProxySetting` utility for your MA deployment:

   ```
   sudo mv ogg.conf /etc/nginx/conf.d/nginx.conf
   ```

   However, this NGINX configuration isn't complete without the `events` section, and enclosing the `map` and `server` sections in `http`.

   Optionally, you can use the default `nginx.conf` file and add the generated `ogg.conf` by adding an `include` statement similar to this:

   ```
   include /etc/nginx/conf.d/ogg.conf;
   ```

   In this case, you must comment out the other `servers` section.

4. Generate a self-signed certificate for NGINX:

   ```
   sudo sh /etc/ssl/certs/make-dummy-cert /etc/nginx/ogg.pem
   ```

   For distribution paths to go through the reverse proxy, you need to use a valid certificate. It's better to specify the same certificate that the deployment is using to process incoming requests, otherwise, starting the path will fail with the next error in Distribution Service:

   ```
   2019-03-26T11:26:00.324-0700 ERROR| ERROR   OGG-10351  Oracle
   GoldenGate Distribution
    Service for Oracle:  Generic error -1 noticed. Error description -
   Certificate validation
   ```

```
 error: Unacceptable certificate from test00abc: application verification
failure. (A4)
```

5. Validate the NGINX configuration:

   ```
   sudo NGINX -t
   ```

   The output would show the following, if the command is successful:

   ```
   NGINX: the configuration file /etc/NGINX/NGINX.conf syntax is ok
   NGINX: configuration file /etc/NGINX/NGINX.conf test is successful
   ```

6. Reload NGINX with the new configuration:

   ```
   sudo NGINX -s reload
   ```

   If the changes for the configuration file are not loaded, stop and restart the proxy.

7. To test if you can access the microservices after NGINX is set up successfully, open the web browser.
8. Enter the proxy URL for the Service Manager using port number 443, similar to the following:

   http://dc.example.com:443

   This would open the Service Manager login page, from where you can access the other microservices also. If you want to directly access a microservice, you can enter the proxy URL for that microservice, as given in the `ogg.conf` file, generated previously.

Also see this video on configuring the NGINX reverse proxy.

SSL Termination

When there is an unsecure connection between the reverse proxy, which uses a TLS-based connection, and the origin server, it is referred to as reverse proxy SSL-termination.

> **Note:**
>
> In SSL-Termination the connections between the reverse proxy and the origin servers are unsecure.

However, SSL-bridging is also supported where the connections between the client and reverse proxy is secured and the connection between the reverse proxy and the origin server is also secured.

4.4 Network Communication

An MA server is the originator of all the response messages sent to the client when a request is sent to the server.

An MA server neither serves as a proxy nor supports tunneling of response messages generated by other applications. Secured network communications use TLS 1.2 or DTLS (Datagram Transport Layer Security) libraries. MA Oracle platforms uses the Oracle SSL toolkit (NZ), which includes Oracle Wallet integration.

For heterogeneous platforms, the Oracle SSL toolkit is used where available.

5

TLS and Secure Network Protocols

Communication security is the confidentiality and integrity of the information sent over communications channels, such as TCP/IP-based networks.

Secure communication implies confidentiality and integrity of data sent over communications channels, such as TCP/IP-based networks. It uses cryptographic protocols to provide communication security over the network.The TLS protocol provides privacy and data integrity when:

- Routing data using the distribution path between the Distribution and Receiver Server.
- Communicating between Oracle GoldenGate and the client applications (web browser, Admin Client or any other Rest API calls)

When secured by TLS, connections between a client and a server should have one or more of the following properties:

- The connection is private (or secure) because cryptography is used to encrypt the data transmitted
- The identity of the communicating parties can be authenticated using public-key cryptography The connection is reliable because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission.

Topics:

5.1 Certificate Access Control List

Learn how you can refine communication security.

The communication security accepts a valid certificate during the connection handshake process. The certificate must be signed by the server or for CA it must trusted by the server. However, you may need to filter and reject otherwise valid certificates based on internal policies. To support this additional validation, the MA extends the standard certificate validation by adding a post-verification certificate Access Control List (ACL) management. This certificate ACL follows the general model used for network ACLs where the ACL is a map with the key identifying the governed element and a value indicating whether the element is allowed or denied. The `certACL` entry has a `scope`specification that allows the ACL entry to be applied to specific identification elements within a certificate.

The configuration of a certificate ACL takes the form of an array of `certACL` entry configuration specification. Each specification minimally contains a permission statement indicating whether it allows or denies client connections from the specified address. The `certACL` entry specifications are processed in order and terminate as soon as the specified address is qualified. If the specified address does not qualify, processing continues with the next specification. Once a certificate is qualified, the `certACL` permissions dictate whether the certificate is allowed or denied. If a no `certACL` entry specification qualify the certificate of the client requesting connection, a default resolution of 'allow is assumed and the certificate is accepted.

`CertACL` Entry Syntax

```
certACL := '[' aclSpec [, aclSpec] ']'
  aclSpec    := '{' perm [',' name [',' scope '}'
  perm       := "permission" ':' [ "deny" | "allow" ]
  name       := "name"       ':' regex
  scope      := "scope"      ':' [ "subject-name" | "issuer-name" ]
  regex      :=  ** Uses the dynamic regular expression syntax.
```

The `regex` syntax follows the ECMAScript definition. Defining a regular expression as a JSON node value requires that the any meta symbols used (like \s) have the \character escaped. You should take care when specifying name regular expression patterns to ensure that only the full match with the intended target pattern is matched. In the syntax, the patterns only full match with the intended target pattern `CN=AdminClnt` not `CN=AdminClnt1`, `CN=AdminClntOther`, `CN=OtherAdminClnt`, or `CCN=OtherAdminClnt` because the match pattern includes delimiter specifications that bound the pattern. These patterns assume a standard distinguished name format that allows no whitespace between the keyname and the value. The `CN = AdminClnt` non-standard pattern would not match.

Example 5-1 Allow All Certificates Example

```
 "CertACL" : [ { "name" : "^(?:(?:\\s*,?)|.*[\\s,]+)(CN=AdminClnt)(?:(?:\\s*(,+\
\s*.*))$|\\s$)", "permission" : "deny" } ]
```

Or

```
"CertACL" : [ { "name" : "^(?:(?:\\s*,?)|.*[\\s,]+)(CN=AdminClnt)(?:(?:\\s*(,+\
\s*.*))$|\\s$)", "scope" : "subject-name", "permission" : "deny" } ]
```

Example 5-2 Deny certificates issued from `Deploy2`

```
 "CertACL" : [ { "name" : "^(?:(?:\\s*,?)|.*[\\s,]+)(CN=Deploy2)(?:(?:\\s*(,+\
\s*.*))$|\\s$)", "scope" : "issuer-name", "permission" : "deny" } ]
```

Example 5-3 Certificates Issued to `Suspect` or Any Certificate Issued By`Deploy2`

```
 "CertACL" : [ { "name" : "^(?:(?:\\s*,?)|.*[\\s,]+)(CN=Suspect)(?:(?:\\s*(,+\
\s*.*))$|\\s$)", "scope" : "subject-name", "permission" : "deny" }, { "name" :
"^(?:(?:\\s*,?)|.*[\\s,]+)(CN=Deploy2)(?:(?:\\s*(,+\\s*.*))$|\\s$)", "scope" :
"issuer-name",  "permission" : "deny" } ]
```

5.2 Transport Layer Security Protocols and Ciphers

Review the supported security protocols.

TLS 1.2 is the default version used with Oracle GoldenGate. See the RFC 5246 for details about the TLS protocol version 1.2.

TLS Security Cipher Suites

The following are the supported security cipher suites and these are the available values that you can use when setting the `/config/securityDetails/network/common/cipherSuites` security setting.

TLS v1.1

```
TLS_RSA_WITH_AES_128_CBC_SHA
    TLS_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
    TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
    TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
    TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
    TLS_ECDH_ECDSA_WITH_RC4_128_SHA
    TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
    TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
    TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDH_RSA_WITH_RC4_128_SHA
```

TLS v1.2

```
TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_128_GCM_SHA256
    TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_RSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
    TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
    TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
    TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
    TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
    TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
```

ECC ciphers are based on the algebraic structure of elliptic curves over finite fields. The elliptic curve discrete logarithm problem (ECDLP) assumes that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible. The benefit of ECC ciphers is that generally the key sizes are smaller compared to non-ECC cipher equivalents.

5.3 TLS Certificate Revocation List Handling

Learn how to configure a revocation list.

A Certificate Revocation List (CRLs) is a Privacy Enhance Mail (PEM) formatted file that contains information identifying the issuer of the revocation list followed by zero or more entries identifying certificate that have been revoked. A secured server is part of establishing a secure channel with a peer and will initiate a handshake with the peer. During this handshake security information and capabilities are negotiated and exchanged, which

includes the one or both certificates of the participants. Depending on security configurations, one, both, or neither of the participants may present or require the presentation of the peer's certificate.

After receiving and verifying the validity of a peer's X.509 certificate, the receiving participant consults the currently configured CRL. The presence of an entry identifying the just-validated peer certificate causes the receiving participant to consider the remote participant's certificate as having been revoked. A revoked certificate is considered invalid for the purposes of authenticating the identity of the remote participant. A revoked certificate fails the integrity-check portion of the secure channel handshake and terminates the channel. Depending on the implementation that remote peer detects that an error occurred during certificate validation, but may not be informed of the specific cause.

The actual CRL consists of prolog and identifies the issuer of the CRL followed by zero or more entries. Each entry identifies a specific certificate by serial number along with security information relating to the date of revocation, the signature algorithm, and finger-print information.

Typically, the CRL in compact form only includes the contents between the `-----BEGIN X509 CRL-----` and `-----END X509 CRL-----` delimiters. All other data outside these delimiters is ignored. You can embed a textual representation of the CRL in the CRL file without affecting the function of the CRL.

The use of CRLs is configured for each MA server individually The CRL configuration is composed of two properties:

`/config/security/common/crlEnabled`
Enables or disables CRL processing.
If, however, `/config/security/common/crlEnabled` is enabled (`true`), then the `/config/security/common/crlStore` property must refer to a valid and well formed CRL.

`/config/security/common/crlStore`
When CRL processing is disabled (`false`), the remote participant's certificate is not checked against a CRL. When this is the case, you don't need to set the `/config/security/common/crlStore` property.

A valid and well formed CRL file is either a PEM encoded CRL file that conforms to the RFC2380 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile or an empty file.

The following is a sample excerpt declaring and defining CRL processing for a secured server.

```
{
    "config" : {
        "security: {
            "common" : {
                "crlEnabled" : true,
                "crlStore"   : "file:/scratch/Tests/unittests/etc/ssl/RootCA/CAs/
Deploy1/CRLs/empty_CRL.pem"
            }
        }
    }
}
```

The CRL file may be updated or replace by other, presumably more current, versions while the server is running. Replacing the CRL file causes the next request CRL lookup to use the newly updated file.

Regardless of how the `/config/security/common/crlEnabled` property is set, CRL processing is disabled if the general security configuration of the server is disabled. For example, the value of the `/config/security` property is `false`).

One other configure setting that indirectly effects CRL processing is the `/config/securityDetails/network/common/authMode` property. This property controls whether the server requires the client to authenticate using a certificate or whether the server accepts optionally presented certificate or whether the server will ignore any presented client certificates. If a certificate is not required, not presented, or ignored by the server, then CRL processing is not used.

5.4 HTTPS Security and Cache Headers

Review the supported security and cache headers.

The MA server accepts and returns HTTPS envelopes that contain a set of headers that govern how the server, the client, and proxies handle the HTTPS contents. For HTTPS information, see:

RFC 7034 - HTTP Header Field X-Frame-Options https://tools.ietf.org/html/rfc7034
RFC 7762 - Initial Assignment for the Content Security Policy https://tools.ietf.org/html/rfc7762
RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1 https://tools.ietf.org/html/rfc2616

Security Headers

The security headers that can be issued are:

Content Security Policy (CSP)
The CSP is included as a header in server responses and defines how the client should handle the content sent by the server.
The default CSP header statement is:

```
Content-Security-Policy: script-src 'self' 'unsafe-eval' 'unsafe-inline'
```

The options are:

- `script-src:`
- `unsafe-eval:`
- `unsafe-inline:`

X-Frame-Options
The X-Frame-Options is included as headers in server responses and signals the client whether or not a user-agent should be allowed to render the content in an `<frame>`, `<iframe>`, or `<object>`. Websites use`<frame>` and `<iframe>` to create mash-ups or to embed part of one site. However, this exposes the embedded site to clickjacking (classified as a user interface redress) attacks. This directive disallows the client from rendering the content as embedded unless the content is from the same site (origin).
The default X-Frame-Options statement is:

```
X-Frame-Options: SAMEORIGIN
```

The option is `SAMEORIGIN`.

X-XSS-Protection
The X-XSS-Protection is included as a header in server responses and configure the user-agent's built in XSS (Cross-Site-Security)protection. The options are to enable, disable and can be combined with block and report.
The default X-XSS-Protection statement is:

```
X-XSS-Protection: 1; mode=block
```

The options are:

- `1`: Enable the user-agent's protection mode.
- `2`: Disable the user-agent's protection mode.
- `mode=block`: Block the server's response if the content script was injected as user input.
- `mode-report=`*`url`*: Report the potential XSS attack to the designated URL. Only supported by Chrome and WebKit.

X-Content-Type-Options
The default `X-Content-Type-Options` statement is:

```
  X-Content-Type-Options: nosniff
```

The option is `nosniff`.

Cache Headers

The supported cache headers are:

Cache-Control
The default `Cache-Control` statement is:

```
Cache-Control: no-cache, no-store, must-revalidate
```

Pragma
The default `Pragma` statement is:

```
Pragma: no-cache
```

Expires
The default `Expires` statement is:

```
Expires: 0
```

HTTP Strict-Transport-Security
The default HTTP Strict-Transport-Security (HSTS) statement is:

```
Strict-Transport-Security: max-age=expire-time; includeSubDomains
```

The configured default for max-age is 31536000 and `includeSubDomains` specifies that the HSTS applies the requesting domain and all subdomains. The default configuration is controlled by:

```
{ "config" : { "hstsEnabled": true, "hstsDetails": "max-age=31536000 ;
includeSubDomains" }}
```

The options are:

`hstsEnable` controls whether or not the HSTS header is included in responses.
`hstsDetails` defines the value of the HSTS header, see RFC 6797 HTTP Strict Transport Security (HSTS).

6
Using Target-Initiated Distribution Paths

Learn about target-initiated distribution paths in MA, the need to set it up, and various use cases where it is helpful to use target-initiated distribution paths.

Topics:

6.1 Overview of Target-Initiated Paths

Target-initiated paths for microservices enable the Receiver Server to initiate a path to the Distribution Service on the target deployment and pull trail files. This feature allows the Receiver Server to create a target initiated path for environments such as Demilitarized Zone Paths (DMZ) or Cloud to on-premise, where the Distribution Server in the source Oracle GoldenGate deployment cannot open network connections in the target environment to the Receiver Server due to network security policies.

If the Distribution Server cannot initiate connections to the Receiver Server, but Receiver Server can initiate a connection to the machine running the Distribution Server, then the Receiver Server establishes a secure or non-secure target initiated path to the Distribution Server through a firewall or Demilitarized (DMZ) zone using Oracle GoldenGate and pull the requested trail files.

The Receiver Server end-points display that the retrieval of the trail files was initiated by the Receiver Server, see Quick Tour of the Receiver Server Home Page.

You can enable this option from the Configuration Assistant wizard Security options, see How to Create Deployments. For steps to create a target-initiated distribution path, see How to Add a Target-Initiated Distrbution Path in *Using the Oracle GoldenGate Microservices Architecture*.

6.2 How Do Target-Initiated Distribution Paths Work?

Oracle GoldenGate has been using the passive alias configuration to initiate passive and alias connection between source and target systems. With MA, this functionality has been enhanced and is not available with target-initiated distribution paths, which can be managed from the Receiver and Distribution server and the Admin Client commands.

The path is created from the Receiver Server of the target deployment and has the property `TARGET_INITIATED`. This is read-only. This path is accessible from the Distribution Server also. The path information is stored on the target system.

If the communication is lost, then the Receiver Server on the target host needs the path definition to restart the connection. This information is shared with the Distribution Server when the path is running.

The path is **ephemeral** on the source deployment. Ephemeral paths help with consolidation of path configuration and with reinforcement of target-to-source connection initiation.

When the path is stopped or disconnected, the Distribution Server removes all the path information including the path definition. However, the checkpoint file is retained because the checkpoint is used to decide whether old trails can be purged or not. It is recommended that old trails are not purged unless the path is intentionally deleted.

Security Configuration between Secure and Non-Secure Source and Target Deployments

The communication channel can optionally be secured using SSL and the same authentication mechanisms. See the Security Options in How to Create Deployments.

Part II
Common Security Features

Use this part to implement security features that are common to both the Microservices Architecture and theClassic Architecture environments.

Topics:

- Managing Encryption Using a Key Management Service in Oracle GoldenGate
  Oracle Key Vault, a Key Management Service (KMS) is supported for both Classic Architecture and Microservices Architecture. This chapter describes the benefits, system requirements, processes, and parameters for configuring Oracle Key Vault with Oracle GoldenGate.
- Encrypting Data with the Master Key and Wallet Method
  To use this method of data encryption, you create a master key wallet and add a master key to the wallet. This method works as follows, depending on whether the data is encrypted in the trails or across TCP/IP:
- Managing Identities in a Credential Store
  Learn how to use an Oracle GoldenGate credential store to maintain encrypted database passwords and user IDs and associate them with an alias.

7

Managing Encryption Using a Key Management Service in Oracle GoldenGate

Oracle Key Vault, a Key Management Service (KMS) is supported for both Classic Architecture and Microservices Architecture. This chapter describes the benefits, system requirements, processes, and parameters for configuring Oracle Key Vault with Oracle GoldenGate.

Topics:

7.1 What is a Key Management Service?

A Key Management Service (KMS) is a utility that centralizes the management of encryption keys.

Oracle GoldenGate Microservices Architecture supports KMS to provide scalability in managing encryption keys and credentials along with security such that the key isn't stored or managed by Oracle GoldenGate.

The Oracle GoldenGate key uses the encapsulation approach to encrypt trail files. It generates a data encryption key (DEK) for each trail file, known as local key. An encrypted version of the local key is included in the trail file header and a master key is used to encrypt the data encryption key. This process is called encapsulation encryption.

In Oracle GoldenGate, a KMS can be used to manage cryptographic keys within an enterprise.

Topics:

7.1.1 Why Use KMS to Store Oracle GoldenGate Encryption Keys?

Oracle GoldenGate encryption of trail files is enhanced by using Oracle Key Vault as the Key Management Service (KMS) to store master keys.

Key management refers to managing cryptographic keys within an enterprise. It deals with generating, exchanging, storing, using, and replacing keys as required. A KMS also includes key servers, user procedures, and protocols. The security of the enterprise is dependent upon successful key management.

The advantages of using KMS with Oracle GoldenGate are:

- Centralized lifecycle management of master keys. You'll be able to generate and upload master keys to Oracle Key Vault directly using custom attributes and perform lifecycle maintenance tasks within the KMS directly.
- Oracle GoldenGate doesn't need to store the master keys locally and is not involved in the lifecycle management of the master keys.
- Oracle GoldenGate can leverage from the specialized KMS features that provide key management with several layers of security.

7.1.2 Oracle Key Vault Capabilities

Oracle GoldenGate supports Oracle Key Vault.

The following table provides the behavior and capabilities of Oracle Key Vault.

For more information about configuring Oracle Key Vault, see Installing and Configuring Oracle Key Vault .

KMS Name	KMS Type	Support Tags	Support Importing of Keys
Oracle Key Vault	Keyname and custom attributes for versioning	Yes	Yes

7.2 Managing Encryption Using a Key Management Service in Oracle GoldenGate Microservices Architecture

This chapter describes the benefits of using a Key Management Services with Oracle GoldenGate Microservices Architecture. It also describes the system requirements, processes and parameters available with Oracle GoldenGate for configuring Oracle Key Vault with Oracle GoldenGate.

Topics:

- What is an Encryption Profile?
  An encryption profile is the configuration information that is used to retrieve a masterkey from a KMS. This includes all the information necessary to connect and authenticate to the KMS server, together with all the details necessary to retrieve a particular masterkey that will be used for encryption and decryption.
- Prerequisites for Configuring OKV on Oracle GoldenGate
  Learn the prerequisites for setting up OKV with Oracle GoldenGate.
- How to Configure an Encryption Profile in MA?
  This topic describes the steps to configure an encryption profile for different KMS options available with Oracle GoldenGate MA.

- Client Behavior Against Different Key States for Oracle Key Vault
  This topic describes the relative behavior of the of the reader or writer client processes depending on the different encryption key states.

7.2.1 What is an Encryption Profile?

An encryption profile is the configuration information that is used to retrieve a masterkey from a KMS. This includes all the information necessary to connect and authenticate to the KMS server, together with all the details necessary to retrieve a particular masterkey that will be used for encryption and decryption.

Any Key Management Service uses an authentication token to access their APIs. Oracle GoldenGate Microservices Architecture stores this access token as a credential. This credential is created using the encryption profile in Microservices Architecture. Encryption profile configuration only available with Microservices Architecture. For Classic Architecture, see Managing Encryption Using a Key Management Service in Oracle GoldenGate Classic Architecture.

An encryption profile is used by the writer and reader clients. A writer client encrypts information, while a reader client decrypts information. In the Microservices Architecture, this is defined by the following roles assigned to each component:

- Extract: Writer client.
- Replicat: Reader client.
- Distribution Server Path: Writer and Reader client.
- LogDump: Reader client.

The clients use the encryption profile that you choose when setting up the encryption MA. The Distribution Server has both the roles of a writer and a reader and only one encryption profile is used. However, if a Distribution Server is operating in PASSTHRU mode then it does not require any encryption profile. Decryption is only needed when column filtering is used. You can create different encryption profiles and all the clients can access the required encryption profile. Clients access their associated encryption profile whenever they need it. A reader will access the encryption profile every time a new trail is being read. The TTL parameter is used to keep the key on memory until time to live (TTL) has been reached.

In MA, each Extract and Replicat process is associated with an encryption profile. The default encryption profile is **Local Wallet** if you haven't specified any other encryption profile as the default.

Already created Extracts, Replicats and Distribution Paths use their associated encryption profile and not a newly created one. Only processes created after the default encryption profile has been changed, will use the newly created encryption profile. So, the **Local Wallet** profile is not used if you specify any other encryption profile for the Extract, Replicat, and Distribution path processes.

A distribution path will use the encryption profile when:

1. the source trail is not encrypted and you have specified the algorithm property in the encryption object:

```
"target": {
            "details": {
                "encryption": {
                    "algorithm": "AES256"
                }
```

```
            },
            "uri": "ogg://localhost:13101/services/v2/targets?
trail=b4"
        }
```

2. The source trail is encrypted and there is a defined filter of type `COLUMNVALUES`.

The Administration Server in microservices allows you to manage your encryption profiles. You cannot modify an encryption profile. If you need to change it, you must delete and add a new profile using the Administration Server.

7.2.2 Prerequisites for Configuring OKV on Oracle GoldenGate

Learn the prerequisites for setting up OKV with Oracle GoldenGate.

The following steps belong to the OKV configuration on the machine where the Oracle GoldenGate instance is running:

1. Download the `okvrestservices.jar` from the OKV server, where Oracle GoldenGate is deployed as the same system user as the deployment.
2. Download and install the endpoint file, `okvclient.jar` from the OKV server, where Oracle GoldenGate is deployed as the same system user as the deployment. For example,

   ```
   OS> java -jar okvclient.jar -d /u01/app/oracle/OKV
   ```

3. Create the key. The name of the wallet is provided by the OKV administrator. The following example show how the key is created:

   ```
   OS> java -jar okvrestservices.jar kmip
           --config /u01/app/oracle/OKV/conf/okvclient.ora
           --service create_key
           --algorithm AES
           --length 256
           --mask
   "ENCRYPT,DECRYPT,TRANSLATE_ENCRYPT,TRANSLATE_DECRYPT,TRANSLATE_WRAP,
   TRANSLATE_UNWRAP"
           --wallet OKV_WALLET76876ABA-B06D-4F35-BF7C-D9306D29764B
   ```

   Alternatively, you can register your own key, as shown in the following example:

   ```
   OS>java -jar okvrestservices.jar kmip
               --config ./conf/okvclient.ora --service reg_key -
   ENCRYPT,DECRYPT,TRANSLATE_ENCRYPT,TRANSLATE_DECRYPT,TRANSLATE_WRAP,T
   RANSLATE_UNWRAP
               --wallet OGG_WALLET
               --object /u01/key.txt64B3AAD0-BE77-1821-
   E053-0100007FD178
   ```

4. Set the `OKV_HOME` environment variable.

   ```
   OS> setenv OKV_HOME /u01/app/oracle/OKV
   ```

   The sub-directory structure contains the necessary libraries, binaries, and configuration files for the OKV environment. See Install OKV Software Onto

Endpoint in the *Oracle Key Vault Administration Guide* for details about the configuration within the OKV server.

5. Activate the key as shown in the following example:

```
OS> java -jar okvrestservices.jar kmip
            --config /u01/app/oracle/OKV/conf/okvclient.ora
            --service activate
            --uid 76876ABA-B06D-4F35-BF7C-D9306D29764B
INFO: Success
```

6. Add the Oracle GoldenGate related key attributes (KeyName, KeyVersion) to the configuration. The key name must match the master keyname in the KMS encryption profile created within Oracle GoldenGate. The key value must match the version number of the masterkey.

```
OS> java -jar okvrestservices.jar kmip
            --config /u01/app/oracle/OKV/conf/okvclient.ora
            --service add_custom_attr
            --uid 76876ABA-B06D-4F35-BF7C-D9306D29764B
            --attribute x-OGG-KeyName
            --type TEXT
            --value OGG_Masterkey
INFO: Success


OS> java -jar okvrestservices.jar kmip
            --config /u01/app/oracle/OKV/conf/okvclient.ora
            --service add_custom_attr
            --uid 76876ABA-B06D-4F35-BF7C-D9306D29764B
            --attribute x-OGG-KeyVersion
            --type TEXT
            --value 1
INFO: Success
```

7. Use `okvutil` to list the configuration setting and check the endpoint status. As shown in the following example:

```
OS>okvutil list -v 4
okvutil version 18.2.0.0.0
Endpoint type: Oracle (non-database)
Configuration file: /u01/app/oracle/OKV/conf/okvclient.ora
Server: 10.245.64.45:5696 10.245.64.46:5696
Standby Servers:Read Servers: 10.245.64.48:5696
Auto-login wallet found, no password needed
Trying to connect to 10.245.64.45:5696 ...
Connected to 10.245.64.45:5696.
Unique ID Type Identifier
72B673E8-840B-4AD6-8400-CB77B68D74B5 Template Default template for OGG_EP
76876ABA-B06D-4F35-BF7C-D9306D29764B Symmetric Key -
```

The next steps are managed within Oracle GoldenGate and are shown as an implementation from the Admin Client.

7.2.3 How to Configure an Encryption Profile in MA?

This topic describes the steps to configure an encryption profile for different KMS options available with Oracle GoldenGate MA.

You can configure encryption profiles from the Administration Server or the AdminClient. To configure the encryption profile using the Administration Server, see Administration Server: Key Management tab.

The Admin Client commands used to set up the encryption profile for Extract, Replicat, and Distribution Path, include `ADD ENCRYPTIONPROFILE`, `ALTER ENCRYPTIONPROFILE`, `DELETE ENCRYPTIONPROFILE`, `INFO ENCRYPTIONPROFILE`. In addition, the `ADD` or `ALTER` the Extract, DISTPATH, or Replicat commands have been modified to include the parameter `ENCRYPTIONPROFILE` *`encryption-profile-name`*.

To know more, see AdminClient Command Line Interface Commands in *Command Line Interface Reference for Oracle GoldenGate*.

There are two options for managing masterkeys:

- Local Wallets
- KMS, which is OKV.

Local Wallet Encryption Profile

The default encryption profile is set to Local Wallet after you install Oracle GoldenGate MA or upgrade to Oracle GoldenGate 19c (19.1.0). For Extract, Replicat, and Distribution Path, the Profile Name field displays the value as Local Wallet.

Oracle Key Vault Encryption Profile

For Oracle Key Vault, the encryption profile credentials require the following inputs:

- Name: Specify the name of the Oracle Key Vault encryption profile.
- Type: Specify the KMS type as `OKV`.
- Home Path: Specify the directory location where Oracle Key Vault is installed. In Admin Client, this is the OKV path. In the web interface, this is the KMS library path.
- Key Name Attribute: Specify the name of the encryption key using this custom attribute. This value must match the key name in the KMS parameter in Oracle GoldenGate and cannot be changed once replication has started.
- Key Version Attribute: Specify the version of the encryption key using this custom attribute. This value must be numeric.
- MasterKey Name: Specify the name of the master key.
- MasterKey Version: Specify the version of Oracle Key Vault. Default value is LATEST or you can specify the version number such as 18.1.
- Time to live: Time to live (TTL) for the key retrieved by Extract from KMS. When encrypting the next trail, Extract checks if TTL has expired. If so, it retrieves the latest version of the master key. The default is 24 hours.

> **Note:**
>
> Do not upload keys with duplicate values of Key Name and Key Version. At the time of startup, restart, or rollover, Oracle GoldenGate processes retrieve the highest Key Version value.

7.2.4 Client Behavior Against Different Key States for Oracle Key Vault

This topic describes the relative behavior of the of the reader or writer client processes depending on the different encryption key states.

A key can be in the following states:

- Active: Trail writer choose the highest version number (unless _Version is specified) with Active state for encryption. Trail reader can use this (key, version number) to decrypt the trail.
- Preactive: Trail writer ignores the key and version number with this state.
- Deactivated: Trail writer ignores the key and version number with this state. Trail file reader retrieves and uses this key and version number to decrypt the trail if it is deactivated or compromised.
- Compromised: Trail writer ignores the key and version number with this state. Trail file reader retrieves and uses this key and version number to decrypt the trail if it is deactivated or compromised.
- Destroyed: Trail writer ignores the key and version number with this state. Trail file reader generates an error and abends if the key and version number required to decrypt is in this state.
- Destroyed-Compromised: Trail writer ignores the key and version number with this state. Trail file reader generates an error and abends if the key and version number required to decrypt is in this state.

8

Encrypting Data with the Master Key and Wallet Method

To use this method of data encryption, you create a master key wallet and add a master key to the wallet. This method works as follows, depending on whether the data is encrypted in the trails or across TCP/IP:

- Each time Oracle GoldenGate creates a trail file, it generates a new encryption key automatically. This encryption key encrypts the trail contents. The master key encrypts the encryption key. This process of encrypting encryption keys is known as **key wrap** and is described in standard ANS X9.102 from American Standards Committee.
- For the Classic Architecture, to encrypt data across the network, Oracle GoldenGate generates a session key using a cryptographic function based on the master key. However, the Distribution Server `ogg` protocol doesn't support this method.

Oracle GoldenGate uses an auto-login wallet (file extension `.sso`), which is an obfuscated container that does not require human intervention to supply the necessary passwords.

Encrypting data with a master key and wallet is not supported on the NonStop platforms.

Topics:

8.1 Creating the Wallet and Adding a Master Key

The wallet is created in a platform-independent format. The wallet can be stored on a shared file system that is accessible by all systems in the Oracle GoldenGate environment. Alternatively, you can use an identical wallet on each system in the Oracle GoldenGate environment. If you use a wallet on each system, you must create the wallet on one system, typically the source system, and then copy it to all of the other systems in the Oracle GoldenGate environment. This must also be done every time you add, change, or delete a master key.

This procedure creates the wallet on the source system and then guides you through copying it to the other systems in the Oracle GoldenGate environment.

1. (Optional) To store the wallet in a location other than the `dirwlt` subdirectory of the Oracle GoldenGate installation directory, specify the desired location with the `WALLETLOCATION` parameter in the `GLOBALS` file.

   ```
   WALLETLOCATION directory_path
   ```

2. Create a master-key wallet with the `CREATE WALLET` command in GGSCI.
3. Open the wallet after it has been created with the `OPEN WALLET` command.

4. Add a master key to the wallet with the `ADD MASTERKEY` command.
5. Issue the `INFO MASTERKEY` command to confirm that the key you added is the current version. In a new installation, the version should be 1.
6. Issue the `INFO MASTERKEY` command with the `VERSION` option, where the version is the current version number. Record the version number and the AES hash value of that version.

   ```
   INFO MASTERKEY VERSION version
   ```

7. Copy the wallet to all of the other Oracle GoldenGate systems.
8. Issue the `INFO MASTERKEY` command with the `VERSION` option on each system to which you copied the wallet, where the version is the version number that you recorded. For each wallet, make certain the `Status` is `Current`. All wallets must show identical key versions.

   ```
   INFO MASTERKEY VERSION version
   ```

8.2 Specifying Encryption Parameters in the Parameter File

This procedure adds the parameters that are required to support data encryption in the trails and across the network with the master key and wallet method.

1. In the following parameter files, add the following:
   - To encrypt trail data: In the parameter file of the primary Extract group and the data pump, add an `ENCRYPTTRAIL` parameter statement before any parameter that specifies a trail or file that you want to be encrypted. Parameters that specify trails or files are `EXTTRAIL`, `RMTTRAIL`, `EXTFILE`, and `RMTFILE`. The syntax is:

     ```
     ENCRYPTTRAIL {AES128 | AES192 | AES256 | BLOWFISH}
     ```

   - To encrypt data across TCP/IP: You can either modify the parameters file using the `RMTHOSTOPIONS ENCRYPT` option or use SOCKS5 proxy to deliver data over the network via a SOCKS5 Proxy. See Using SOCKS5 Proxy to Deliver Encrypted Data.

     In the parameter file of the data pump (or the primary Extract, if no pump is being used), use the `ENCRYPT` option of the `RMTHOSTOPTIONS` parameter. The syntax is:

     ```
     RMTHOSTOPTIONS host, MGRPORT port, ENCRYPT {AES128 | AES192 | AES256 |
     BLOWFISH}

     RMTHOSTOPTIONS ENCRYPT {AES128 | AES192 | AES256 | BLOWFISH}
     ```

   Where:

   - `RMTHOSTOPTIONS` is used for Extract including passive extracts. See Using Target System Connection Initiation for more information about passive Extract.
   - `ENCRYPTTRAIL` without options specifies 256-key byte substitution. This format is not secure and should not be used in a production environment. Use only for backward compatibility with earlier Oracle GoldenGate versions.
   - `AES128` encrypts with the AES-128 encryption algorithm.
   - `AES192` encrypts with AES-192 encryption algorithm.

- `AES256` encrypts with AES-256 encryption algorithm.
- `BLOWFISH` uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use AES if supported for the platform. Use `BLOWFISH` for backward compatibility with earlier Oracle GoldenGate versions, and for DB2 z/OS and DB2 for i. AES is not supported on those platforms.

2. Use the `DECRYPTTRAIL` parameter for a data pump if you want trail data to be decrypted before it is written to the output trail. Otherwise, the data pump automatically decrypts it, if processing is required, and then reencrypts it before writing to the output trail. (Replicat decrypts the data automatically without any parameter input.) Also see How to Configure an Encryption Profile in MA?.

   ```
   DECRYPTTRAIL
   ```

> **Note:**
>
> You can explicitly decrypt incoming trail data and then re-encrypt it again for any output trails or files. First, enter `DECRYPTTRAIL` to decrypt the data, and then enter `ENCRYPTTRAIL` and its output trail specifications. `DECRYPTTRAIL` must precede `ENCRYPTTRAIL`. Explicit decryption and re-encryption enables you to vary the AES algorithm from trail to trail, if desired. For example, you can use AES 128 to encrypt a local trail and AES 256 to encrypt a remote trail. Alternatively, you can use the master key and wallet method to encrypt from one process to a second process, and then use the `ENCKEYS` method to encrypt from the second process to the third process.

- Using SOCKS5 Proxy to Deliver Encrypted Data

8.2.1 Using SOCKS5 Proxy to Deliver Encrypted Data

The SOCKS5 protocol routes packets between a server and a client using a proxy server. The protocol establishes a TCP connection to another server on behalf of the client and then routes the traffic between the client and server, while hiding the identity of the client from the public network.

In Oracle GoldenGate, you can use SOCKS5 proxy to deliver data over the network with the `RMHOSTOPTIONS` parameter. See `RMTHOSTOPTIONS` for details.

To create a SOCKS5 proxys with SSH tunneling to securely transmit data over the network, perform the following steps:

1. On Linux, a SOCKS5 proxy can be set up along with SSH tunneling using the following SSH command:

   ```
   ssh -i private_key file -v -N -f -D listening IP Address:listening IP
   port GGCS
         Oracle User@GGCS IP Address socksproxy output file
   -N: No execution command on remote system
   -D: Dynamic Port Forwarding
   -i: Private Key File
   -f: Run the proxy process in the background
   ```

```
-v: Verbose Mode
-C: Compression
```

The following example shows the SOCKS5 proxy with SSH tunnel connecting to a GoldenGate Cloud Service (GGCS) instance with IP address 129.145.2.34:

```
ssh -N -f -i opc_rsa.ppk -D 127.0.0.1:1080 opc@129.145.2.34 /tmp/
ogg_socksproxy.log
```

After setting up the SOCKS5 proxy, you can set up Oracle GoldenGate on-premises Pump to deliver data to GGCS using the SOCKSPROXY parameter from the proxy server, as shown in the following example:

```
RMTHOST 129.145.2.34, COMPRESS, MGRPORT 1021, SOCKSPROXY 127.0.0.1:1080
```

8.3 Renewing the Master Key

This procedure renews the master encryption key in the encryption-key wallet. Renewing the master key creates a new version of the key. Its name remains the same, but the bit ordering changes. As part of your security policy, you should renew the current master key regularly so that it does not get stale.

All renewed versions of a master key remain in the wallet until they are marked for deletion with the `DELETE MASTERKEY` command and then the wallet is purged with the `PURGE WALLET` command, see Deleting Stale Master Keys.

Unless the wallet is maintained centrally on shared storage (as a shared wallet), the updated wallet must be copied to all of the other systems in the Oracle GoldenGate configuration that use that wallet. To do so, the Oracle GoldenGate processes need to be stopped. This procedure includes steps for performing those tasks in the correct order.

1. Stop Extract. You need to stop the `TRANLOG` Extract group, which is the Extract capturing from the transaction logs.

   > **Note:**
   >
   > If the `TRANLOG` Extract group is also acting as an Extract pump, then you need to stop the applicable and stop all activity on the database as well.

   ```
   STOP EXTRACT group
   ```

2. Check the read RBA of the Extract data pump against the size of the Extract trail by performing the following:

   ```
   INFO EXTRACT SHELL ls -l ./dirdat/[trail identifier]*
   ```

   When the read RBA matches the trail size, then there is no more data to process. When the Extract pump or Extract sends all the trail files to the target, then you should start checking the Replicat to determine when it is at EOF.

3. On the source system, stop the data pumps.

   ```
   STOP EXTRACT group
   ```

4. Check the read RBA of the Replicat against the size of the remote trail.
5. On the target systems, stop the Replicat when the read RBA matches the size of the remote trail.

   ```
   STOP REPLICAT group
   ```

6. On the source system, issue the following command to open the wallet.

   ```
   OPEN WALLET
   ```

7. On the source system, issue the following command to confirm the version of the current key. Make a record of the version.

   ```
   INFO MASTERKEY
   ```

8. On the source system, issue the following command to renew the master key.

   ```
   RENEW MASTERKEY
   ```

9. On the source system, issue the following command to confirm that a new version is current.

   ```
   INFO MASTERKEY
   ```

   > **Note:**
   >
   > If you are using a shared wallet, go to step 12. If you are using a wallet on each system, continue to the next step.

10. On the source system, issue the following command, where *version* is the new version of the master key. Make a record of the hash value.

    ```
    INFO MASTERKEY VERSION version
    ```

11. Copy the updated wallet from the source system to the same location as the old wallet on all of the target systems.
12. On each target, issue the following command, where *version* is the new version number of the master key. For each wallet, make certain the `Status` is `Current` and compare the new hash value with the one that you originally recorded. All wallets must show identical key versions and hash values.

    ```
    INFO MASTERKEY VERSION version
    ```

13. Restart Extract.

    ```
    START EXTRACT group
    ```

14. Restart the data pumps.

    ```
    START EXTRACT group
    ```

15. Restart Replicat.

    ```
    START REPLICAT group
    ```

8.4 Deleting Stale Master Keys

This procedure deletes stale versions of the master key. Deleting stale keys should be part of the overall policy for maintaining a secure Oracle GoldenGate wallet. It is recommended that you develop a policy for how many versions of a key you want to keep in the wallet and how long you want to keep them.

> **Note:**
>
> For Oracle GoldenGate deployments using a shared wallet, the older versions of the master key should be retained after the master key is renewed until all processes are using the newest version. The time to wait depends on the topology, latency, and data load of the deployment. A minimum wait of 24 hours is a conservative estimate, but you may need to perform testing to determine how long it takes for all processes to start using a new key. To determine whether all of the processes are using the newest version, view the report file of each Extract immediately after renewing the master key to confirm the last SCN that was mined with the old key. Then, monitor the Replicat report files to verify that this SCN was applied by all Replicat groups. At this point, you can delete the older versions of the master key.

If the wallet is on central storage that is accessible by all Oracle GoldenGate installations that use that wallet, you need only perform these steps once to the shared wallet. You do not need to stop the Oracle GoldenGate processes.

If the wallet is not on central storage (meaning there is a copy on each Oracle GoldenGate system) you can do one of the following:

- If you can stop the Oracle GoldenGate processes, you only need to perform the steps to change the wallet once and then copy the updated wallet to the other systems before restarting the Oracle GoldenGate processes.
- If you cannot stop the Oracle GoldenGate processes, you must perform the steps to change the wallet on each system, making certain to perform them exactly the same way on each one.

  These steps include prompts for both scenarios.

1. On the source system, issue the following command to determine the versions of the master key that you want to delete. Typically, the oldest versions should be the ones deleted. Make a record of these versions.

   ```
   INFO MASTERKEY
   ```

2. On the source system, issue the following command to open the wallet.

   ```
   OPEN WALLET
   ```

3. Issue the following command to delete the stale master keys. Options are available to delete a specific version, a range of versions, or all versions including the current one. To delete all of the versions, transaction activity and the Oracle GoldenGate processes must be stopped.

   ```
   DELETE MASTERKEY {VERSION version | RANGE FROM begin_value TO end_value}
   ```

> **Note:**
>
> `DELETE MASTERKEY` marks the key versions for deletion but does not actually delete them.

4. Review the messages returned by the `DELETE MASTERKEY` command to ensure that the correct versions were marked for deletion. To unmark any version that was marked erroneously, use the `UNDELETE MASTERKEY VERSION` *`version`* command before proceeding with these steps. If desired, you can confirm the marked deletions with the `INFO MASTERKEY` command.
5. When you are satisfied that the correct versions are marked for deletion, issue the following command to purge them from the wallet. This is a permanent deletion and cannot be undone.

   ```
   PURGE WALLET
   ```

   Next steps:

   - If the wallet resides on shared storage, you are done with these steps.
   - If there is a wallet on each system and you cannot stop the Oracle GoldenGate processes, repeat the preceding steps on each Oracle GoldenGate system.
   - If there is a wallet on each system and you can stop the Oracle GoldenGate processes, continue with these steps to stop the processes and copy the wallet to the other systems in the correct order.
6. Stop Extract.

   ```
   STOP EXTRACT group
   ```
7. In GGSCI, issue the following command for each data pump Extract until each returns `At EOF`, indicating that all of the data in the local trail has been processed.

   ```
   SEND EXTRACT group STATUS
   ```
8. Stop the data pumps.

   ```
   STOP EXTRACT group
   ```
9. On the target systems, issue the following command for each Replicat until it returns `At EOF`.

   ```
   SEND REPLICAT group STATUS
   ```
10. Stop the Replicat groups.

    ```
    STOP REPLICAT group
    ```
11. Copy the updated wallet from the source system to the same location as the old wallet on all of the target systems.
12. Restart Extract.

    ```
    START EXTRACT group
    ```
13. Restart the data pumps.

    ```
    START EXTRACT group
    ```
14. Restart Replicat.

```
START REPLICAT group
```

9
Managing Identities in a Credential Store

Learn how to use an Oracle GoldenGate credential store to maintain encrypted database passwords and user IDs and associate them with an alias.

It is the alias, not the actual user ID or password, that is specified in a command or parameter file, and no user input of an encryption key is required. The credential store is implemented as an autologin wallet within the Oracle Credential Store Framework (CSF).

Another benefit of using a credential store is that multiple installations of Oracle GoldenGate can use the same one, while retaining control over their local credentials. You can partition the credential store into logical containers known as **domains**, for example, one domain per installation of Oracle GoldenGate. Domains enable you to develop one set of aliases (for example `ext` for Extract, `rep` for Replicat) and then assign different local credentials to those aliases in each domain. For example, credentials for user `ogg1` can be stored as `ALIAS ext` under `DOMAIN system1`, while credentials for user `ogg2` can be stored as `ALIAS ext` under `DOMAIN system2`.

The credential store security feature is not supported on the DB2 for i, DB2 z/OS, and NonStop platforms. For those platforms and any other supported platforms, see Encrypting a Password in a Command or Parameter File.

Topics:

9.1 Creating and Populating the Credential Store

1. (Optional) To store the credential store in a location other than the `dircrd` subdirectory of the Oracle GoldenGate installation directory, specify the desired location with the `CREDENTIALSTORELOCATION` parameter in the `GLOBALS` file.
2. From the Oracle GoldenGate installation directory, run GGSCI.
3. Issue the following command to create the credential store.

   ```
   ADD CREDENTIALSTORE
   ```

4. Issue the following command to add each set of credentials to the credential store.

   ```
   ALTER CREDENTIALSTORE ADD USER userid,
     [PASSWORD password]
     [ALIAS alias]
     [DOMAIN domain]
   ```

   Where:

   - *`userid`* is the user name. Only one instance of a user name can exist in the credential store unless the `ALIAS` or `DOMAIN` option is used.

- *password* is the password. The password is echoed (not obfuscated) when this option is used. For security reasons, it is recommended that you omit this option and allow the command to prompt for the password, so that it is obfuscated as it is entered.
- *alias* is an alias for the user name. The alias substitutes for the credential in parameters and commands where a login credential is required. If the ALIAS option is omitted, the alias defaults to the user name. If you do not want user names in parameters or command input, use ALIAS and specify a different name from that of the user.
- *domain* is the domain that is to contain the specified alias. The default domain is *Oracle GoldenGate*.

9.2 Specifying the Alias in a Parameter File or Command

The following commands and parameters accept an alias as substitution for a login credential.

Table 9-1 Specifying Credential Aliases in Parameters and Commands

Purpose of the Credential	Parameter or Command to Use
Oracle GoldenGate database login.	USERIDALIAS *alias*
Oracle GoldenGate database login for Oracle ASM instance.	TRANLOGOPTIONS ASMUSERALIAS *alias*
Oracle GoldenGate database login for a downstream Oracle mining database.	TRANLOGOPTIONS MININGUSERALIAS *alias*
Password substitution for {CREATE | ALTER} USER *name* IDENTIFIED BY *password*.	DDLOPTIONS DEFAULTUSERPASSWORDALIAS *alias*
Oracle GoldenGate database login from GGSCI.	DBLOGIN USERIDALIAS *alias*
Oracle GoldenGate database login to a downstream Oracle mining database from GGSCI.	MININGDBLOGIN USERIDALIAS *alias*

Part III
Securing the Classic Architecture

Use this part to secure your Classic Architecture environments.

Topics:

- Securing Manager
  You can use the Manager parameter, `ACCESSRULE`, to set security access rules for Manager. It allows GGSCI access from a remote host if you are using passive Extract or Director.
- Configuring GGSCI Command Security
  You can establish command security for Oracle GoldenGate to control which users have access to which Oracle GoldenGate functions.
- Using Target System Connection Initiation
  Learn how to allow Oracle GoldenGate to replicate into a more secure network or server where communication must be established from the target back to the source system..
- Managing Encryption Using a Key Management Service in Oracle GoldenGate Classic Architecture
  This chapter describes the benefits of using a Key Management Service with Oracle GoldenGate Classic Architecture. It also describes the system requirements, processes and parameters available with Oracle GoldenGate for configuring Oracle Key Vault with Oracle GoldenGate.
- Configuring SSL Support for PostgreSQL

Managing Encryption Using a Key Management Service in Oracle GoldenGate Classic Architecture

This chapter describes the benefits of using a Key Management Service with Oracle GoldenGate Classic Architecture. It also describes the system requirements, processes and parameters available with Oracle GoldenGate for configuring Oracle Key Vault with Oracle GoldenGate.

Topics:

- Registering Oracle GoldenGate Endpoint in Oracle Key Vault
  To retrieve trail file encryption from Oracle Key Vault, you must register the Oracle GoldenGate endpoints.
- Uploading Master Keys in Oracle Key Vault
  You decide the method you want to use to upload the master keys to Oracle Key Vault.
- Configuring Oracle GoldenGate
  You need to configure the Key Management Services (KMS) global parameters in Oracle GoldenGate for Oracle Key Vault.

- Oracle GoldenGate Trail Writer and Reader Behavior for Different Master Key States
  Oracle GoldenGate behaves differently depending on the key states.

Registering Oracle GoldenGate Endpoint in Oracle Key Vault

To retrieve trail file encryption from Oracle Key Vault, you must register the Oracle GoldenGate endpoints.

To configure Oracle GoldenGate:

1. Register the Oracle GoldenGate in Oracle Key Vault 18.1 or later:
   Download and install the endpoint file, `okvclient.jar`, where Oracle GoldenGate is deployed as the same system user as the deployment.

   ```
   java -jar okvclient.jar -d OKV_HOME
   ```

   > **Note:**
   >
   > Ensure that wallet created while installing the endpoint is an auto-login wallet.

2. Specify the following Oracle Key Vault details with the *GLOBALS* parameter using these options:
   - Location of `OKV_HOME`
   - Name of the master key *master_key_name*.
   - `KMS TYPE` with the value set as `OKV`
   - `KMS_VERSION` with the value set as `18.1`.
   - Time to live `TTL`.

   See Configuring Oracle GoldenGate for details about KMS globals parameter values.

Uploading Master Keys in Oracle Key Vault

You decide the method you want to use to upload the master keys to Oracle Key Vault.

There are some prerequisites when uploading master keys to Oracle Key Vault.

- Generate symmetric master keys of 256 bits and upload to Oracle Key Vault.
- Add custom attributes to the uploaded key:
  - `x-OGG-KeyName` *key_name*: This value must match the key name in the KMS parameter in Oracle GoldenGate and cannot be changed once replication has started.
  - `x-OGG-KeyVersion` *version#*: This value must be numeric.

- Do not upload keys with duplicate values of `x-OGG-KeyName` and `x-OGG-KeyVersion`. At the time of startup, restart, or rollover, Oracle GoldenGate processes retrieve the highest `x-OGG-KeyVersion` value

Topics:

- Register and Upload Master Keys in Oracle Key Vault
  Advanced Encryption Standard (AES) 256 master keys can be generated externally and uploaded to the Oracle Key Vault.
- Create Oracle GoldenGate Master Keys in Oracle Key Vault
  You can create an AES 256 master key for Oracle GoldenGate instead of registering it.

Register and Upload Master Keys in Oracle Key Vault

Advanced Encryption Standard (AES) 256 master keys can be generated externally and uploaded to the Oracle Key Vault.

Use the Oracle Key Vault REST utility to register or create the master keys, which Oracle GoldenGate can retrieve for trail file encryption and decryption.

For details, see Oracle Key Vault Automation with RESTful Services.

Here are the steps to register the master key using the REST utility:

1. Register an AES 256 master key. The following is an example:

```
  java  -jar okvrestservices.jar kmip
          --config ./conf/okvclient.ora
          --service reg_key -
ENCRYPT,DECRYPT,TRANSLATE_ENCRYPT,TRANSLATE_DECRYPT,TRANSLATE_WRAP,TRANSLA
TE_UNWRAP
          --wallet OGG_WALLET
          --object /u01/key.txt
  ØReturns a UID, eg: 64B3AAD0-BE77-1821-E053-0100007FD178
```

   See Oracle Key Vault Use Case Scenarios for more information about registering and uploading master keys.

2. Activate the master key. The following is a example:

```
    java  -jar okvrestservices.jar kmip
          --config ./conf/okvclient.ora
          --service activate
          --uid A9917590-4F7C-4F5B-BF62-E7872C797638
```

3. Add the Oracle GoldenGate master key name and version attributes to the key, as shown in the following example:

```
    java  -jar okvrestservices.jar kmip
          --config ./conf/okvclient.ora
          --service  add_custom_attr
          --uid 64B3AAD0-BE77-1821-E053-0100007FD178
          --attribute x-OGG-KeyName
```

```
        --type TEXT
        --value OGG_MASTER_KEY_NAME
```

```
java  -jar okvrestservices.jar kmip
       --config ./conf/okvclient.ora
       --service  add_custom_attr
        --uid 64B3AAD0-BE77-1821-E053-0100007FD178
       --attribute x-OGG-KeyVersion
       --type TEXT
       --value 201
```

Oracle GoldenGate identifies the master key for a particular deployment using the custom attributes, `x-OGG-KeyName` and `x-OGG-KeyVersion`.

4. Specify the following Oracle Key Vault values in the new KMS global parameter. See Configuring Oracle GoldenGate :
   - The location of Oracle Key Vault home directory (*OKV_HOME*)
   - Name of the master key
   - Time-to-live

   An example with the Oracle Key Vault values is:

   ```
   KMS TYPE OKV KMS_VERSION 18.1 HOME /u01/app/okv_home
   MASTER_KEY_NAME OGG1 TTL 60mins
   ```

> **Note:**
>
> Do not register multiple keys with the same *x-OGG-KeyName* and *x-OGG-KeyVersion*.

Create Oracle GoldenGate Master Keys in Oracle Key Vault

You can create an AES 256 master key for Oracle GoldenGate instead of registering it.

Here are the steps to create the Oracle GoldenGate master key in Oracle Key Vault:

1. Create an AES 256 Oracle GoldenGate master key, as shown in the following example.

   ```
   java  -jar okvrestservices.jar kmip --config ./conf/okvclient.ora --
   service create_key
    --algorithm AES --length 256 --mask "ENCRYPT,DECRYPT" --wallet
   OGG_WALLET
   ```

   This code returns a UID similar to `64B3AAD0-BE77-1821-E053-0100007FD177`.

2. Activate the master key, as shown in the following example.

```
java  -jar okvrestservices.jar kmip --config ./conf/okvclient.ora --
service activate
    --uid 64B3AAD0-BE77-1821-E053-0100007FD177
```

3. Add the Oracle GoldenGate master key name and version attributes to the key, as shown in the following example.

```
    java  -jar okvrestservices.jar kmip --config ./conf/okvclient.ora
      --service  add_custom_attr --uid 64B3AAD0-BE77-1821-
E053-0100007FD178
      --attribute x-OGG-KeyName --type TEXT --value OGG_MASTER_KEY_NAME
```

```
    java  -jar okvrestservices.jar kmip --config ./conf/okvclient.ora
      --service  add_custom_attr --uid 64B3AAD0-BE77-1821-
E053-0100007FD178
      --attribute x-OGG-KeyVersion --type TEXT --value 201
```

Oracle GoldenGate identifies the master key for a particular deployment using the custom attributes, *x-OGG-KeyName* and *x-OGG-KeyVersion*.

Configuring Oracle GoldenGate

You need to configure the Key Management Services (KMS) global parameters in Oracle GoldenGate for Oracle Key Vault.

The syntax for specifying the KMS global parameter is:

```
KMS TYPE kms_type KMS_VERSION kms_version HOME okv_home MASTER_KEY_NAME
master_key_name [ TTL  time ]
```

For example:

```
KMS TYPE OKV  KMS_VERSION 18.1 HOME /u01/OKV MASTER_KEY_NAME OGG1 [ TTL  10 ]
```

These are the KMS global parameter values.

Option	Mandatory	Value	Description
TYPE	Yes	OKV	Specifies type of KMS. Only OKV is supported.
HOME	Yes	KMS endpoint software installation directory.	Directory location of the OKV_HOME where the Oracle Key Vault endpoint software was installed.
KMS_VERSION	Yes	Version of the KMS	Specifies the version of the KMS software.

Option	Mandatory	Value	Description
MASTER_KEY_NAME	Yes	Master key name	Master key name that needs to the same on source and target and cannot be changed without quiescing replication.
TTL	No	Time	Time to live (TTL) for the key retrieved by Extract from KMS. When encrypting the next trail, Extract checks if TTL has expired. If so, it retrieves the latest version of the master key. The default is 24 hours.

Oracle GoldenGate Trail Writer and Reader Behavior for Different Master Key States

Oracle GoldenGate behaves differently depending on the key states.

These are the behaviors of the trail writer (encryption) and trail reader (decryption) for each key state.

Key State	Trail Writer (encryption)	Trail Reader (decryption)
Active	Trail writer chooses the highest version number with Active state for encryption.	Trail reader can use this key and version number to decrypt the trail.
Preactive	Trail writer ignores and does not consider the key version number with these states.	Not Applicable
Deactivated	None	Trail file reader retrieves and decrypts the trail if the key and version number is deactivated or compromised.
Compromised	None	Trail file reader retrieves and decrypts the trail if the key and version number is deactivated or compromised.
Destroyed	Non	Trail file reader generates an error and abends if the key and version number required to decrypt is in the destroyed or destroyed-compromised state.

Key State	Trail Writer (encryption)	Trail Reader (decryption)
Destroyed-Compromised	None	Trail file reader raises an error and abends if the key and version number required to decrypt is in the destroyed or destroyed-compromised state.

Configuring SSL Support for PostgreSQL

SSL can be enabled by setting the configuration parameter `SSL` to `on` in the PostgreSQL configuration file (`$PGDATA/postgresql.conf`). If SSL is enabled, the corresponding `hostssl` entry must be present or added in the `pg_hba.conf` file.

When SSL is enabled, Oracle GoldenGate uses the root certificate, root certification revocation list (CRL), server client certificate, and key from the default locations, as shown in the following snippet:

```
~/.postgresql/root.crt
~/.postgresql/root.crl
~/.postgresql/postgresql.crt
~/.postgresql/postgresql.key
```

You need to create the desired entities from this list, and store them in appropriate locations.

If the SSL configuration is setup using non-default locations, then the following environment variables should be set up as per the environment.

```
PGSSLROOTCERT
PGSSLCRL
PGSSLCERT
PGSSLKEY
```

Changes required in $ODBCINI file

The SSL support can be enabled by setting the `EncryptionMethod DSN` attribute to 1 or 6 in the `$ODBCINI` file.

If set to 0 (No Encryption), data is not encrypted.

If set to 1 (SSL), data is encrypted using the SSL protocols specified in the Crypto Protocol Version connection option. If the specified encryption method is not supported by the database server, the connection fails and the driver returns an error.

If set to 6 (RequestSSL), the login request and data are encrypted using SSL if the server is configured for SSL. If the server is not configured for SSL, an unencrypted connection is established. The SSL protocol used is determined by the setting of the Crypto Protocol Version connection option.

If the database server/client certificates also need to be validated, then the corresponding KeyStore file needs to be created and the below mentioned ODBC DSN attributes should be setup accordingly in `$ODBCINI`.

```
KeyStore=<path to .p12 keystore file>
KeyStorePassword=<keystore-passwd>
TrustStore=<path to root certificate>
ValidateServerCertificate=1
```

10
Securing Manager

You can use the Manager parameter, ACCESSRULE, to set security access rules for Manager. It allows GGSCI access from a remote host if you are using passive Extract or Director.

The ACCESSRULE parameter controls connection access to the Manager process and the processes under its control. You can establish multiple rules by specifying multiple ACCESSRULE statements in the parameter file and control their priority. To establish priority, you can either list the rules in order from most important to least important, or you can explicitly set the priority of each rule with the PRI option.

You must specify one of the following options:

IPADDR, login_ID, or PROGRAM

For example, the following access rules have been assigned explicit priority levels through the PRI option. These rules allow any user to access the Collector process (the SERVER program), and in addition, allow the IP address 122.11.12.13 to access GGSCI commands. Access to all other Oracle GoldenGate programs is denied.

```
ACCESSRULE, PROG *, DENY, PRI 99
ACCESSRULE, PROG SERVER, ALLOW, PRI 1
ACCESSRULE, PROG GGSCI, IPADDR 122.11.12.13, PRI 1
```

Another example, the following access rule grants access to all programs to the user JOHN and designates an encryption key to decrypt the password. If the password provided with PASSWORD matches the one in the ENCKEYS lookup file, connection is granted.

```
ACCESSRULE, PROG *, USER JOHN, PASSWORD OCEAN1, ENCRYPTKEY lookup1
```

11
Configuring GGSCI Command Security

You can establish command security for Oracle GoldenGate to control which users have access to which Oracle GoldenGate functions.

> **Note:**
>
> The `GGSCI` program is only available in the Oracle GoldenGate CA.

For example, you can allow certain users to issue `INFO` and `STATUS` commands, while preventing their use of `START` and `STOP` commands. Security levels are defined by the operating system's user groups.

To implement security for Oracle GoldenGate commands, you create a `CMDSEC` file in the Oracle GoldenGate directory. Without this file, access to all Oracle GoldenGate commands is granted to all users.

> **Note:**
>
> The security of the `GGSCI` program is controlled by the security controls of the operating system.

Topics:

11.1 Setting Up Command Security

1. Open a new ASCII text file.
2. Referring to the following syntax and the example on , create one or more security rules for each command that you want to restrict, one rule per line. List the rules in order from the most specific (those with no wildcards) to the least specific. Security rules are processed from the top of the `CMDSEC` file downward. The first rule satisfied is the one that determines whether or not access is allowed.

   Separate each of the following components with spaces or tabs.

   ```
   command_name command_object OS_group OS_user {YES | NO}
   ```

   Where:

   - *`command_name`* is a GGSCI command name or a wildcard, for example `START` or `STOP` or *.

- *command_object* is any GGSCI command object or a wildcard, for example EXTRACT or REPLICAT or MANAGER.
- *OS_group* is the name of a Windows or UNIX user group. On a UNIX system, you can specify a numeric group ID instead of the group name. You can use a wildcard to specify all groups.
- *OS_user* is the name of a Windows or UNIX user. On a UNIX system, you can specify a numeric user ID instead of the user name. You can use a wildcard to specify all users.
- YES | NO specifies whether access to the command is granted or prohibited.

3. Save the file as CMDSEC (using upper case letters on a UNIX system) in the Oracle GoldenGate home directory.

The following example illustrates the correct implementation of a CMDSEC file on a UNIX system.

Table 11-1 Sample CMDSEC File with Explanations

File Contents	Explanation
`#GG command security`	Comment line
`STATUS REPLICAT * Smith NO`	STATUS REPLICAT is denied to user Smith.
`STATUS * dpt1 * YES`	Except for the preceding rule, all users in dpt1 are granted all STATUS commands.
`START REPLICAT root * YES`	START REPLICAT is granted to all members of the root group.
`START REPLICAT * * NO`	Except for the preceding rule, START REPLICAT is denied to all users.
`* EXTRACT 200 * NO`	All EXTRACT commands are denied to all groups with ID of 200.
`* * root root YES`	Grants the root user any command.
`* * * * NO`	Denies all commands to all users. This line covers security for any other users that were not explicitly granted or denied access by preceding rules. Without it, all commands would be granted to all users except for preceding explicit grants or denials.

The following *incorrect* example illustrates what to avoid when creating a CMDSEC file.

Table 11-2 Incorrect CMDSEC Entries

File Contents	Description
`STOP * dpt2 * NO`	All STOP commands are denied to everyone in group dpt2.
`STOP * * Chen YES`	All STOP commands are granted to Chen.

The order of the entries in Table 11-2 causes a logical error. The first rule (line 1) denies all `STOP` commands to all members of group `dpt2`. The second rule (line 2) grants all `STOP` commands to user `Chen`. However, because Chen is a member of the `dpt2` group, he has been denied access to all `STOP` commands by the second rule, even though he is supposed to have permission to issue them.

The proper way to configure this security rule is to set the user-specific rule before the more general rule(s). Thus, to correct the error, you would reverse the order of the two `STOP` rules.

11.2 Securing the CMDSEC File

The security of the GGSCI program and that of the `CMDSEC` file is controlled by the security controls of the operating system. Because the `CMDSEC` file is a source of security, it must be secured. You can grant read access as needed, but Oracle recommends denying write and delete access to everyone except for system administrators.

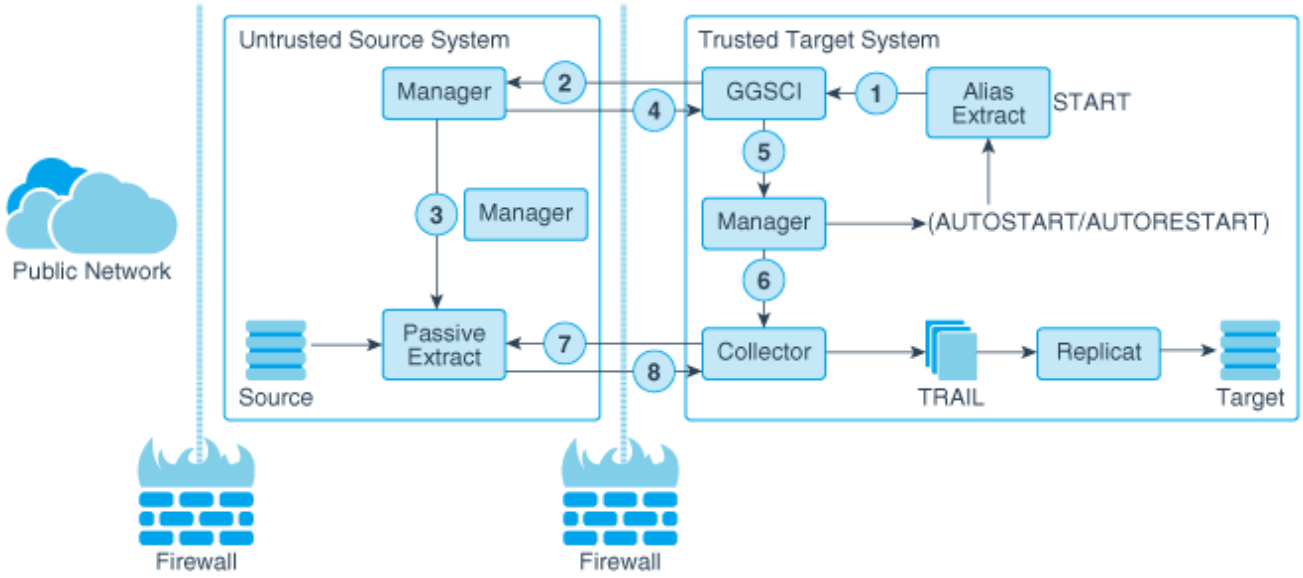
12

Using Target System Connection Initiation

Learn how to allow Oracle GoldenGate to replicate into a more secure network or server where communication must be established from the target back to the source system..

When a target system resides inside a trusted intranet zone, initiating connections from the source system (the standard Oracle GoldenGate method) may violate security policies if the source system is in a less trusted zone. It also may violate security policies if a system in a less trusted zone contains information about the ports or IP address of a system in the trusted zone, such as that normally found in an Oracle GoldenGate Extract parameter file.

In this kind of intranet configuration, you can use a **passive-alias Extract** configuration. Connections are initiated from the target system inside the trusted zone by an **alias Extract** group, which acts as an alias for a regular Extract group on the source system, known in this case as the **passive Extract**. Once a connection between the two systems is established, data is processed and transferred across the network by the passive Extract group in the usual way.



1. An Oracle GoldenGate user starts the alias Extract on the trusted system, or an `AUTOSTART` or `AUTORESTART` parameter causes it to start.
2. GGSCI on the trusted system sends a message to Manager on the less trusted system to start the associated passive Extract. The host name or IP address and port number of the Manager on the trusted system are sent to the less trusted system.
3. On the less trusted system, Manager starts the passive Extract, and the passive Extract finds an open port (according to rules in the `DYNAMICPORTLIST` Manager parameter) and listens on that port.
4. The Manager on the less trusted system returns that port to GGSCI on the trusted system.

5. GGSCI on the trusted system sends a request to the Manager on that system to start a Collector process on that system.
6. The target Manager starts the Collector process and passes it the port number where Extract is listening on the less trusted system.
7. Collector on the trusted system opens a connection to the passive Extract on the less trusted system.
8. Data is sent across the network from the passive Extract to the Collector on the target and is written to the trail in the usual manner for processing by Replicat.

Topics:

12.1 Configuring the Passive Extract Group

The passive Extract group on the less trusted source system will be one of the following, depending on which one is responsible for sending data across the network:

- A solo Extract group that reads the transaction logs and also sends the data to the target, or:
- A data pump Extract group that reads a local trail supplied by a primary Extract and then sends the data to the target. In this case, there are no special configuration requirements for the primary Extract, just the data pump.

> **Note:**
>
> The passive Extract group is only available in the Oracle GoldenGate CA.

To create an Extract group in passive mode, use the standard `ADD EXTRACT` command and options, but add the `PASSIVE` keyword in any location relative to other command options. Examples:

```
ADD EXTRACT fin, TRANLOG, BEGIN NOW, PASSIVE, DESC 'passive Extract'
ADD EXTRACT fin, PASSIVE, TRANLOG, BEGIN NOW, DESC 'passive Extract'
```

To configure parameters for the passive Extract group, create a parameter file in the normal manner, except:

- *Exclude* the `RMTHOST` parameter, which normally would specify the host and port information for the target Manager.
- Use the optional `RMTHOSTOPTIONS` parameter to specify any compression and encryption rules. For information about the `RMTHOSTOPTIONS` options, see *Reference for Oracle GoldenGate*.

12.2 Configuring the Alias Extract Group

The alias Extract group on the trusted target does not perform any data processing activities. Its sole purpose is to initiate and terminate connections to the less trusted source. In this capacity, the alias Extract group does not use a parameter file nor does it write processing checkpoints. A checkpoint file is used only to determine whether the passive Extract group is running or not and to record information required for the remote connection.

> **Note:**
>
> The alias Extract group is only available in the Oracle GoldenGate CA.

To create an Extract group in alias mode, use the `ADD EXTRACT` command without any other options except the following:

```
ADD EXTRACT group
, RMTHOST {host_name | IP_address}
, MGRPORT port
[, RMTNAME name]
[, DESC 'description']
```

The `RMTHOST` specification identifies this group as an alias Extract, and the information is written to the checkpoint file. The *`host_name`* and *`IP_address`* options specify the name or IP address of the source system. `MGRPORT` specifies the port on the source system where Manager is running.

The alias Extract name can be the same as that of the passive Extract, or it can be different. If the names are different, use the optional `RMTNAME` specification to specify the name of the passive Extract. If `RMTNAME` is not used, Oracle GoldenGate expects the names to be identical and writes the name to the checkpoint file of the alias Extract for use when establishing the connection.

Error handling for TCP/IP connections is guided by the `TCPERRS` file on the target system. It is recommended that you set the response values for the errors in this file to `RETRY`. The default is `ABEND`. This file also provides options for setting the number of retries and the delay between attempts. For more information about error handling for TCP/IP and the `TCPERRS` file.

12.3 Starting and Stopping the Passive and Alias Processes

To start or stop Oracle GoldenGate extraction in the passive-alias Extract configuration, you must start or stop the alias Extract group from GGSCI on the target.

```
START EXTRACT alias_group_name
```

or,

```
STOP EXTRACT alias_group_name
```

The command is sent to the source system to start or stop the passive Extract group. Do not issue these commands directly against the passive Extract group. You can issue a `KILL EXTRACT` command directly for the passive Extract group.

When using the Manager parameters `AUTOSTART` and `AUTORESTART` to automatically start or restart processes, use them on the target system, not the source system. The alias Extract is started first and then the start command is sent to the passive Extract.

12.4 Managing Extraction Activities

Once extraction processing has been started, you can manage and monitor it in the usual manner by issuing commands against the passive Extract group from GGSCI on the source system. The standard GGSCI monitoring commands, such as `INFO` and `VIEW REPORT`, can be issued from either the source or target systems. If a monitoring command is issued for the alias Extract group, it is forwarded to the passive Extract group. The alias Extract group name is replaced in the command with the passive Extract group name. For example, `INFO EXTRACT` *`alias`* becomes `INFO EXTRACT` *`passive`*. The results of the command are displayed on the system where the command was issued.

12.5 Other Considerations when using Passive-Alias Extract

When using a passive-alias Extract configuration, these rules apply:

- In this configuration, Extract can only write to one target system.
- This configuration can be used in an Oracle RAC installation by creating the Extract group in the normal manner (using the `THREADS` option to specify the number of redo threads).
- The `ALTER EXTRACT` command cannot be used for the alias Extract, because that group does not do data processing.
- To use the `DELETE EXTRACT` command for a passive or alias Extract group, issue the command from the local GGSCI.
- Remote tasks, specified with `RMTTASK` in the Extract parameter file and used for some initial load methods, are not supported in this configuration. A remote task requires the connection to be initiated from the source system and uses a direct connection between Extract and Replicat.

A

Encrypting a Password in a Command or Parameter File

Learn how to encrypt a database password that is to be specified in a command or parameter file. This method should only be used for HP NonStop platforms. All other platforms should use the Oracle Credential store to create an alias for using commands or parameter files.

This method takes a clear-text password as input and produces an obfuscated password string and a lookup key, both of which can then be used in the command or parameter file. This encryption method supports all of the databases that require a login for an Oracle GoldenGate process to access the database.

Oracle recommends that you use the `USERIDALIAS`, `ASMUSERALIAS`, or `MININGUSERALIAS` options before using this method.

Depending on the database, you may be able to use a credential store as an alternative to this method. See Managing Identities in a Credential Store.

Topics:

A.1 Encrypting the Password

1. Run GGSCI.
2. Issue the `ENCRYPT PASSWORD` command.

   ```
   ENCRYPT PASSWORD password algorithm ENCRYPTKEY {key_name | DEFAULT}
   ```

   Where:

   - *password* is the clear-text login password. Do not enclose the password within quotes. If the password is case-sensitive, type it that way.
   - *algorithm* specifies the encryption algorithm to use:
     - `AES128` uses the AES 128 cipher, which has a key size of 128 bits.
     - `AES192` uses the AES 192 cipher, which has a key size of 192 bits.
     - `AES256` uses the AES 256 cipher, which has a key size of 256 bits.
     - `BLOWFISH` uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32-bits to 128-bits. Use AES if supported for the platform. Use `BLOWFISH` for backward compatibility with earlier Oracle GoldenGate versions, and for DB2 z/OS and DB2 for i. AES is not supported on those platforms.
   - `ENCRYPTKEY` *key_name* specifies the logical name of a user-created encryption key in the `ENCKEYS` lookup file. The key name is used to look up the actual key in the `ENCKEYS` file. Using a user-defined key and an `ENCKEYS` file is required for AES

encryption. To create a key and `ENCKEYS` file, see Populating an ENCKEYS File with Encryption Keys.

- `ENCRYPTKEY DEFAULT` directs Oracle GoldenGate to generate a predefined Blowfish key. This type of key is insecure and should not be used in a production environment if the platform supports AES. Use this option only for DB2 on /OS and DB2 for i when `BLOWFISH` is specified. `ENCRYPT PASSWORD` returns an error if AES is used with `DEFAULT`.

  If no algorithm is specified, AES 128 is the default for all database types except DB2 z/OS, where `BLOWFISH` is the default.

The following are examples of `ENCRYPT PASSWORD` with its various options.

```
ENCRYPT PASSWORD mypassword AES256 ENCRYPTKEY mykey1
ENCRYPT PASSWORD mypassword BLOWFISH ENCRYPTKEY mykey1
ENCRYPT PASSWORD mypassword BLOWFISH ENCRYPTKEY DEFAULT
```

3. The encrypted password is output to the screen when you run the `ENCRYPT PASSWORD` command. Copy the encrypted password and then see Specifying the Encrypted Password in a Parameter File or Command for instructions on pasting it to a command or parameter.

A.2 Specifying the Encrypted Password in a Parameter File or Command

Copy the encrypted password that you generated with the `ENCRYPT PASSWORD` command (see Encrypting a Password in a Command or Parameter File), and then paste it into the appropriate Oracle GoldenGate parameter statement or command as in the following table. Option descriptions follow the table.

Table A-1 Specifying Encrypted Passwords in Parameters and Commands

Purpose of the Password	Parameter or Command to Use
Oracle GoldenGate database login Syntax elements required for `USERID` vary by database type. See *Reference for Oracle GoldenGate* for more information.	`USERID` *user*, `PASSWORD` *encrypted-password*, & *algorithm* `ENCRYPTKEY {`*keyname* `| DEFAULT}`
Oracle GoldenGate database login for Oracle ASM instance	`TRANLOGOPTIONS ASMUSER SYS@`*ASM_instance_name*, & `ASMPASSWORD` *encrypted-password*, & *algorithm* `ENCRYPTKEY {`*keyname* `| DEFAULT}`

Oracle GoldenGate database login for a downstream Oracle mining database	`[MININGUSER {/ |` *user*`}[, MINGPASSWORD`
	encrypted-password`]&`
	`[`*algorithm* `ENCRYPTKEY {`*key_name* `| DEFAULT}]&`
	`[SYSDBA]]`

Password substitution for `{CREATE	ALTER} USER` *name* `IDENTIFIED BY` *password*	`DDLOPTIONS DEFAULTUSERPASSWORD` *encrypted-password* & *algorithm* `ENCRYPTKEY {`*keyname* `| DEFAULT}`

Table A-1 (Cont.) Specifying Encrypted Passwords in Parameters and Commands

Purpose of the Password	Parameter or Command to Use
Oracle TDE shared-secret password	`DBOPTIONS DECRYPTPASSWORD encrypted-password`[1] `algorithm &` `ENCRYPTKEY {keyname | DEFAULT}`
Oracle GoldenGate database login from GGSCI	`DBLOGIN USERID user, PASSWORD encrypted-password, &` `algorithm ENCRYPTKEY {keyname | DEFAULT}`
Oracle GoldenGate database login to a downstream Oracle mining database from GGSCI	`MININGDBLOGIN USERID user, PASSWORD encrypted-password,&` `algorithm ENCRYPTKEY {keyname | DEFAULT}`

1 This is the shared secret.

Where:

- *user* is the database user name for the Oracle GoldenGate process or (Oracle only) a host string. For Oracle ASM, the user must be `SYS`.
- *encrypted-password*is the encrypted password that is copied from the `ENCRYPT PASSWORD` command results. Do not enclose the password within quotes. Do not use commas in passwords. If the password is case-sensitive, type it that way.
- *algorithm* specifies the encryption algorithm that was used to encrypt the password: `AES128`, `AES192`, `AES256`, or `BLOWFISH`. AES128 is the default if the default key is used and no algorithm is specified.
- `ENCRYPTKEY` *keyname* specifies the logical name of a user-created encryption key in the `ENCKEYS` lookup file. Use if `ENCRYPT PASSWORD` was used with the `KEYNAME` *keyname* option.
- `ENCRYPTKEY DEFAULT` directs Oracle GoldenGate to use a random key. Use if `ENCRYPT PASSWORD` was used with the `KEYNAME DEFAULT` option.

The following are examples of using an encrypted password in parameters and command:

> **Note:**
>
> In the following example, comma is used as a separator and is not part of the password.

```
SOURCEDB db1 USERID ogg,&
PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
BLOWFISH, ENCRYPTKEY securekey1

USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
```

```
BLOWFISH, ENCRYPTKEY DEFAULT

TRANLOGOPTIONS ASMUSER SYS@asm1, &
ASMPASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

DBLOGIN USERID ogg, PASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES128, ENCRYPTKEY securekey1

DDLOPTIONS DEFAULTUSERPASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey

DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey

DDLOPTIONS PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey
```

B
Avoiding Security Attacks

Learn about security attacks and ways to mitigate them.

Topics:

B.1 Cross Site Request Forgery

Learn how to avoid client-side attacks.

Oracle GoldenGate has CSRF mitigation support that is controlled by the server's configuration. The default configuration is to enforce CSRF-token based protection.

Cross Site Request Forgery (CSRF) protection when enabled applies to any request issued from a web browser that's originating from a script or programmatic interface. CSRF protection is only checked for requests that intend to modify resources at the origin server. This means that PUT, POST, and DELETE are the only HTTP request verbs where CSRF protection will be enforced (if enabled).

CSRF protection will not be enforced regardless of CSRF being enabled for requests issued from non-browser clients such as `curl`, `wget`, or `netcat`. CSRF is also not enforced for request from Admin Client as none of these clients are web browsers.

For more information, see Open Web Application Security Project Cross-Site Request Forgery (CSRF) page for further details.

C

Encrypting Data with the ENCKEYS Method

To use this method of data encryption, you configure Oracle GoldenGate to generate an encryption key and store the key in a local `ENCKEYS` file.

> **Note:**
>
> Oracle only recommends the use of this method for platforms where master key and wallet support is not available. You should not use this method if wallet-based support is available.

The method secures the date in the trails or an Extract file and data sent across TCP/IP networks.

The `ENCKEYS` method is valid for all Oracle GoldenGate-supported databases and platforms. Blowfish must be used on the DB2 for i, DB2 z/OS, and NonStop platforms.

Encrypts the data in files, across data links, and across TCP/IP. Use any of the following:

- Any Advanced Encryption Security (AES) cipher: Advanced Encryption Standard (AES) is a symmetric-key encryption standard that is used by governments and other organizations that require a high degree of data security. It offers three 128-bit block-ciphers: a 128-bit key cipher, a 192-bit key cipher, and a 256-bit key cipher. To use AES for any database other than Oracle on a 32-bit platform, the path to the lib sub-directory of the Oracle GoldenGate installation directory must be set with the library path variable. Bug 27523872 For different platforms the library path variable is different. For Linux it is LD_LIBRARY_PATH. For IBM i and AIX it is LIBPATH, SHLIB_PATH variable for Solaris and the PATH variable on Windows. Not required for 64-bit platforms.

  AES-128

  AES-192

  AES-256

- Blowfish encryption: A keyed symmetric-block cipher. The Oracle GoldenGate implementation of Blowfish has a 64-bit block size with a variable-length key size from 32 bits to 256 bits.

This method makes use of a permanent key that can only be changed by regenerating the algorithm, see Populating an ENCKEYS File with Encryption Keys.

The `ENCKEYS` file must be secured through the normal method of assigning file permissions in the operating system.

This procedure generates an AES encryption key and provides instructions for storing it in the `ENCKEYS` file. `ENCKEYS` file for microservices is stored in the *`deployment_dir`*`/etc/conf/ogg` directory. In classic architecture, it's in the install location (same location as GGSCI).

Topics:

- Populating an ENCKEYS File with Encryption Keys
  Learn how to use an ENCKEYS file.

C.1 Setting Up the Data Encryption

1. Generate an encryption key and store it in the ENCKEYS file, see Populating an ENCKEYS File with Encryption Keys. Make certain to copy the finished ENCKEYS file to the Oracle GoldenGate installation directory on any intermediary systems and all target systems.
2. In the following parameter files, add the following:
   - To encrypt trail data: In the parameter file of the primary Extract group and the data pump, add an ENCRYPTTRAIL parameter before any parameter that specifies a trail or file that you want to be encrypted. Parameters that specify trails or files are EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. The syntax is one of the following:

     ```
     ENCRYPTTRAIL {AES128 | AES192 | AES256 | BLOWFISH}

     ENCRYPTTRAIL AES192, KEYNAME keyname
     ```

   - To encrypt data across TCP/IP: In the RMTHOSTOPTIONS parameter in the parameter file of the data pump (or the primary Extract, if no pump is being used), add the ENCRYPT option with the KEYWORD clause. The syntax is one of the following:

     ```
     RMTHOSTOPTIONS host, MGRPORT port, ENCRYPT {AES128 | AES192 | AES256 |
     BLOWFISH} KEYNAME keyname

     RMTHOSTOPTIONS ENCRYPT {AES128 | AES192 | AES256 | BLOWFISH} KEYNAME
     keyname
     ```

   Where:

   - RMTHOSTOPTIONS is used for passive Extract, see Populating an ENCKEYS File with Encryption Keys.
   - ENCRYPTTRAIL without options uses AES 128 as the default for all database types except the DB2 for i, DB2 z/OS, and NonStop platforms, where BLOWFISH is the default.
   - AES128 encrypts with the AES 128 encryption algorithm. Not supported for iDB2 for i, DB2 z/OS, and NonStop platforms.
   - AES192 encrypts with AES 192 encryption algorithm. Not supported for DB2 for i, DB2 z/OS, and NonStop platforms.
   - AES256 encrypts with AES 256 encryption algorithm. Not supported for iSeries, z/OS, and NonStop platforms.
   - BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32-bits to 128-bits. Use AES if supported for the platform. Use BLOWFISH for backward compatibility with earlier Oracle GoldenGate versions, and for DB2 for I and DB2 z/OS. AES is not supported on those platforms.
   - KEYNAME *keyname* specifies the logical look-up name of an encryption key in the ENCKEYS file. Not an option of ENCRYPTTRAIL.

> **Note:**
>
> RMTHOST is used unless the Extract is in a passive configuration.

3. If using a static Collector with data encrypted over TCP/IP, append the following parameters in the Collector startup string:

   ```
   -KEYNAME keyname
   -ENCRYPT algorithm
   ```

   The specified key name and algorithm must match those specified with the KEYNAME and ENCRYPT options of RMTHOST.

- Decrypting the Data with the ENCKEYS Method
- Examples of Data Encryption using the ENCKEYS Method

C.1.1 Decrypting the Data with the ENCKEYS Method

Data that is encrypted over TCP/IP connections is decrypted automatically at the destination before it is written to a trail, unless trail encryption also is specified.

Data that is encrypted in the trail remains encrypted unless the DECRYPTTRAIL parameter is used. DECRYPTTRAIL is required by Replicat before it can apply encrypted data to the target. A data pump passes encrypted data untouched to the output trail, unless the DECRYPTTRAIL and ENCRYPTTRAIL parameters are used. If the data pump must perform work on the data, decrypt and encrypt the data as follows.

To Decrypt Data for Processing by a Data Pump

Add the DECRYPTTRAIL parameter to the parameter file of the data pump. The decryption algorithm and key must match the ones that were used to encrypt the trail, see Setting Up the Data Encryption.

```
DECRYPTTRAIL {AES128 | AES192 | AES256 | BLOWFISH}
```

To Encrypt Data After Processing by a Data Pump

To encrypt data before the data pump writes it to an output trail or file, use the ENCRYPTTRAIL parameter before the parameters that specify those trails or files. Parameters that specify trails or files are EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. The ENCRYPTTRAIL parameter and the trail or file specifications must occur after the DECRYPTTRAIL parameter.

> **Note:**
>
> The algorithm specified with ENCRYPTTRAIL can vary from trail to trail. For example, you can use AES 128 to encrypt a local trail and AES 256 to encrypt a remote trail.

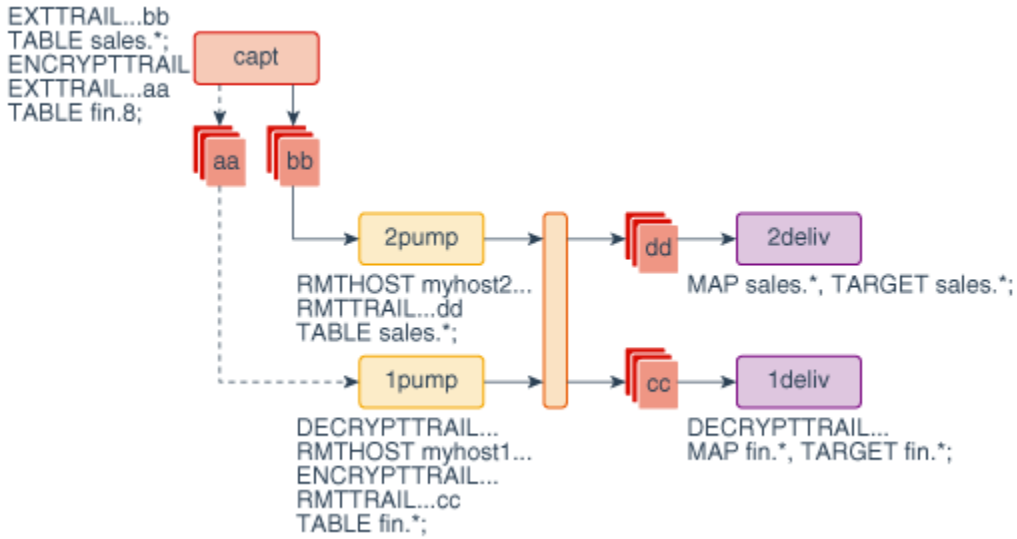
To Decrypt Data for Processing by Replicat

If a trail that Replicat reads is encrypted, add a DECRYPTTRAIL parameter statement to the Replicat parameter file. The decryption algorithm and key must match the ones that were used to encrypt the trail.

C.1.2 Examples of Data Encryption using the ENCKEYS Method

The following example shows how to turn encryption on and off for different trails or files. In this example, Extract writes to two local trails, only one of which must be encrypted.

In the Extract configuration, trail `bb` is the non-encrypted trail, so its `EXTTRAIL` parameter is placed before the `ENCRYPTTRAIL` parameter that encrypts trail `aa`. Alternatively, you can use the `NOENCRYPTTRAIL` parameter before the `EXTTRAIL` parameter that specifies trail `bb` and then use the `ENCRYPTTRAIL` parameter before the `EXTTRAIL` parameter that specifies trail `aa`.



In this example, the encrypted data must be decrypted so that data pump `1pump` can perform work on it. Therefore, the `DECRYPTTRAIL` parameter is used in the parameter file of the data pump. To re-encrypt the data for output, the `ENCRYPTTRAIL` parameter must be used after `DECRYPTTRAIL` but before the output trail specifications. If the data pump did not have to perform work on the data, the `DECRYPTTRAIL` and `ENCRYPTTRAIL` parameters could have been omitted to retain encryption all the way to Replicat.

Example C-1 Extract Parameter File

```
EXTRACT capt
USERIDALIAS ogg
DISCARDFILE /ogg/capt.dsc, PURGE
-- Do not encrypt this trail.
EXTTRAIL /ogg/dirdat/bb
TABLE SALES.*;
-- Encrypt this trail with AES-192.
ENCRYPTTRAIL AES192
EXTTRAIL /ogg/dirdat/aa
TABLE FIN.*;
```

Example C-2 Data Pump 1 Parameter File

```
EXTRACT 1pump
USERIDALIAS ogg
```

```
DISCARDFILE /ogg/1pmp.dsc, PURGE
-- Decrypt the trail this pump reads. Use encryption key mykey1.
DECRYPTTRAIL AES192
-- Encrypt the trail this pump writes to, using AES-192.
RMTHOSTOPTIONS myhost1, MGRPORT 7809
ENCRYPTTRAIL AES192
RMTTRAIL /ogg/dirdat/cc
TABLE FIN.*;
```

Example C-3 Data pump 2 Parameter File

```
EXTRACT 2pump
USERIDALIAS ogg
DISCARDFILE /ogg/2pmp.dsc, PURGE
RMTHOST myhost2, MGRPORT 7809
RMTTRAIL /ogg/dirdat/dd
TABLE SALES.*;
```

Example C-4 Replicat1 (on myhost1) Parameter File

```
REPLICAT 1deliv
USERIDALIAS ogg
ASSUMETARGETDEFS
DISCARDFILE /ogg/1deliv.dsc, PURGE
-- Decrypt the trail this Replicat reads. Use encryption key mykey2.
DECRYPTTRAIL AES192
MAP FIN.*, TARGET FIN.*;
```

Example C-5 Replicat 2 (on myhost2) parameter file

```
REPLICAT 2deliv
USERIDALIAS ogg
ASSUMETARGETDEFS
DISCARDFILE /ogg/2deliv.dsc, PURGE
MAP SALES.*, TARGET SALES.*;
```

C.2 Populating an ENCKEYS File with Encryption Keys

Learn how to use an `ENCKEYS` file.

You must generate and store encryption keys when using the security features:

- `ENCRYPTTRAIL` (see Setting Up the Data Encryption)
- `ENCRYPT PASSWORD` with `ENCRYPTKEY` *`keyname`* (see Encrypting a Password in a Command or Parameter File)
- `RMTHOST` or `RMTHOSTOPTIONS` with `ENCRYPT` (see Setting Up the Data Encryption)

You can define your own key or run the Oracle GoldenGate `KEYGEN` utility to create a random key.

Topics:

- Defining Your Own Key
- Using KEYGEN to Generate a Key
- Creating and Populating the ENCKEYS Lookup File

C.2.1 Defining Your Own Key

Use a tool of your choice. The key value can be up to 256-bits (32 bytes) as either of the following:

- a quoted alphanumeric string (for example `"Dailykey"`)
- a hex string with the prefix `0x` (for example `0x420E61BE7002D63560929CCA17A4E1FB`)

C.2.2 Using KEYGEN to Generate a Key

Change directories to the Oracle GoldenGate home directory on the source system, and issue the following shell command. You can create multiple keys, if needed. The key values are returned to your screen. You can copy and paste them into the `ENCKEYS` file.

```
KEYGEN key_length n
```

Where:

- *`key_length`* is the encryption key length, up to 256-bits (32 bytes).
- *`n`* represents the number of keys to generate.

Example:

```
KEYGEN 128 4
```

C.2.3 Creating and Populating the ENCKEYS Lookup File

1. On the source system, open a new ASCII text file.
2. For each key value that you generated, enter a logical name of your choosing, followed by the key value itself.
   - The key name can be a string of 1 to 24 alphanumeric characters without spaces or quotes.
   - Place multiple key definitions on separate lines.
   - Do not enclose a key name or value within quotes; otherwise it is interpreted as text.

   Use the following sample `ENCKEYS` file as a guide.

Encryption key name	Encryption key value
`## Key name`	`Key value`
`superkey`	`0x420E61BE7002D63560929CCA17A4E1FB`
`secretkey`	`0x027742185BBF232D7C664A5E1A76B040`
`superkey1`	`0x42DACD1B0E94539763C6699D3AE8E200`
`superkey2`	`0x0343AD757A50A08E7F9A17313DBAB045`
`superkey3`	`0x43AC8DCE660CED861B6DC4C6408C7E8A`

3. Save the file as the name `ENCKEYS` in all upper case letters, without an extension, in the Oracle GoldenGate installation directory.

4. Copy the `ENCKEYS` file to the Oracle GoldenGate installation directory on every system. The key names and values in all of the `ENCKEYS` files must be identical, or else the data exchange will fail and Extract and Collector will abort with the following message:

```
GGS error 118 - TCP/IP Server with invalid data.
```