Oracle® Fusion Middleware

Securing the Oracle GoldenGate Environment

18c (18.1.0)
E95980-03

Oracle Fusion Middleware Securing the Oracle GoldenGate Environment, 18c (18.1.0)

E95980-03

Primary Author: Oracle Corporation

Contents

Audience

This guide is intended for the person or persons who are responsible for operating Oracle GoldenGate and maintaining its performance. This audience typically includes, but is not limited to, systems administrators and database administrators.

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, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

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

Introducing Oracle GoldenGate Security

Oracle GoldenGate includes many security features that provide varying levels of security. Understanding the security features and the uses cases they cover are important first steps when learning how to secure your environment.

There are two different architectures offered with Oracle GoldenGate:

Microservices Architecture (MA)
This is a REST API Microservices-based architecture that allows you to configure, monitor, and manage Oracle GoldenGate services using a web-based UI.
You can use MA to deploy, monitor, manage, and perform Extract and Replicat operations on trail data within your MA implementation. To know 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.

Securing both architectures is detailed in these parts:

- Securing the Microservices Architecture
- Securing Oracle GoldenGate

Part I
Securing the Microservices Architecture

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

The MA service interfaces use the REST architectural style, within an HTTP environment. As REST is a style that uses HTTP and not a distinct transfer implementation, all the security related concerns and solutions applied to HTTP apply equally to REST interfaces. This includes ensuring general security related to HTTP-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/SSL when enabled, ensures confidentiality and optionally integrity, although typical configurations do not ensure bi-lateral integrity. Negotiating security configurations can further specify identity validation, renegotiation, and revocation requirements as allowed by Oracle security standards.

Communications Transport

All REST Service Interfaces and Data Conveyances may be conducted over the following network transport:

- TCP is used for network communication.
- UDT is an additional protocol used for data conveyance. It is a high-performance, UDP-based data transfer protocol, which transfers large datasets over high-speed WAN.
- WebSockets 2.0 is a not a transport protocol but a pseudo-transport that enables a server to send content to client without client solicitation, thereby enabling bi-directional messaging over a persistent connection. It operates over HTTPS ports simplifying network security management.

Communications Security

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 Oracle approved TLS (Transport Layer Security) or DTLS (Datagram Transport Layer Security) libraries. MA Oracle platforms uses the Oracle SSL toolkit (NZ), which includes Oracle Wallet integration.

For non-Oracle platforms, the Oracle SSL toolkit is used where available. Where the Oracle SSL Toolkit is not available, an alternate SSL toolkit is used.

All MA servers implement client and server authentication. However, client and server authentication is only available when network security is configured and enabled. MA servers can be configured with network security enabled but without using server or client authentication.

Inbound and Outbound Security Configuration

Security configuration can be inbound or outbound. Inbound configuration implies configuring specific behavior associated with a server. A server receives requests and

responds with information or messages. Outbound security configuration assumes that the specific behavior is associated with a client.

A client issues requests and receives the response information from the server. Only the Distribution Server acts as a client with outbound security requirements. All other servers are server-only. For example, in MA, the Distribution Server accepts service requests from clients through inbound configured secured connections, while it connects and sends trail data to Receiver Server through secure connections with Outbound configuration.

Topics:

- MA Security Features
  Learn about these MA security features:
- Network
  Learn how to secure your network for Oracle GoldenGate.
- Authentication and Authorization
  The MA security and authorization model declares and defines how communication security (confidentiality and Integrity) and Authorization (authentication and permissions) are configured and implemented.
- Communication Security
  Communication security is the confidentiality and integrity of the information sent over communications channels, such as TCP/IP-based networks.
- Server and Deployment Identities
  You must uniquely identify MA servers and deployments using schemes.
- Securing Deployments
  You can choose to set up a secure or non-secure deployment. A secure deployment involves making RESTful API calls and conveying trail data between the Distribution Server and Receiver Server, over SSL/TLS. You can use your existing wallets and certificates, or you can create new ones.

MA Security Features

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.
- **IPv6 Support**: Oracle GoldenGate network implementations support native IPv6 addressing standards.
- **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.
- **User Credential Storage**: MA implementations address this by using Oracle Wallets and related identity management services to store security information.

Approved encryption technologies are configured to secure both stored and in-flight user data. Stored data typically refers to file system files like capture data trail files while in-flight data typically refers to data transmitted between peers over a non-persistent communications channel.

- **Single Page Applications (SPAs) and WebApp Security**: If the initial connection to the Service Manager uses the HTTPS protocol, then the browser connects using SSL/TLS. If the server is configured to require the client to present a certificate, the browser needs to be configured to present the appropriate client certificate.
- **Cipher-suites**: The cipher-suites for MA are configured during deployment. You can change the value of the cipher-suite using the Server Manager REST interfaces for each server. Alternatively, you can update then using either the MA boostrap configuration override option or the command-line configuration override options. The list of cipher-suites available to a user differs based on the environment. This ensures that there is sufficient overlap to allow secure communication at the required security level.

  Both client and server platforms generally support more than one cipher-suite. This increases the probability that the client and server can negotiate and agree on a cipher-suite to use. The set of available cipher-suites on the server is dictated by the NZ Toolkit (or alternate TLS/SSL toolkit). There are several cipher-suites set as the default set and is dependent on the Java Runtime Environment distributed with Oracle GoldenGate. The default set attempts to specify the most common cipher-suites with the highest security protection and highest performance. However, in practice you need to choose between high security and high performance as these are competing attributes and there is a trade-off between security and performance.

2
Network

Learn how to secure your network for Oracle GoldenGate.

Topics:

Network Access Control

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

Each ACL specification minimally consists of a permission statement indicating whether the APC 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 no 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 form:

```
ipACL := '[' aclSpec [, aclSpec] ']'
aclSpec := "permission" : [ "deny" | "allow" ] [, "address": [ ipv4Address
| ipv4MappedAddress | ipv6Address ] ]
ipv4Address := '"' decimal '.' decimal '.' decimal '.' decimal '"'
ipv4MappedAddress := '"' 'ff::' decimal '.' decimal '.' decimal '.'
decimal '"'
ipv6Address := '"' hexadecimal ':' hexadecimal ':' hexadecimal ':'
hexadecimal ':' hexadecimal ':' hexadecimal ':' hexadecimal ':'
hexadecimal '"'
```

Inbound connection request 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 2-1 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" } ]
```

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" } ]
```

Network Connection Adapter

Learn about how to specify your network connection configuration.

The actual network connection information is captured in Network Connection Specification of the Software Communications Architecture (SCA). In the description of the `ScaNetworkSpec` class, instances of the `ScsNetworkSpec` represent the network configuration information acquired from the `ScaSharedContext`. The `ScaNetworkSpec` handles the discrete network specification. However, a complete SCA network specification takes any of three forms and can define more than one network configuration. Multiple networks is when more than one network interface is configured in an environment where the host be multi-homed, For example, handling connections 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, an SCA 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 normalize 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 2-2 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
```

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 describes as:

Form 1 - 4
Listens on port 9000 on all `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 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 `ScaNetworkSpec` resolves to only a subset of adapters. Precedence processing allows the specification of `ANY` address and `ALL` interfaces for the last `ScaNetworkSpec` as a pool specification when the platform networking interfaces support mapping sub-set interface matches

Proxy Support

Learn how to configure your proxy servers.

Proxy configuration mediates access with different MA servers within a network for a deployment.

MA requires you to exhibit proper and compliant behavior in a network environment where one or more proxy servers may mediate access to MA servers.

Configuration

The initial configuration is simply declaring whether proxy detection should be enabled or disabled. Typically, it is enabled by default though you can disable it in `/config/network/proxyDetails`. The enable clause is similar to:

```
{
    "network"             : {
        "proxyEnabled": true,
        "proxyDetails": {
            "proxyACLEnabled": true,
            "proxyACL": [
                { "permission": "deny",  "address":
"192.0.2.254"                    },
                { "permission": "allow", "address": "192.0.2.254",
"trusted": false },
                { "permission": "allow", "address": "ANY",
"trusted": true  }
            ],
            "urlMappingEnabled": true,
            "urlMapping": [
            ]
        }
    }
  }
```

Proxy ACL Specifications

The configuration of Proxy ACL specifications is similar to Network IP ACL specifications. The differences are that each entry defines the access control for a proxy server in your environment and includes a trust designator. Each ACL specification minimally consists of a permission statement indicating whether the ACL specification allows or denies client connections proxied through the proxy server's 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 no ACL specifications qualify address of the client requesting connection, a default resolution of allow is used and the client is allowed to connect. The ACL in the configuration may take the following form:

```
ipACL := '[' aclSpec> [, aclSpec] ']'
aclSpec := "permission" : [ "deny" | "allow" ] [, "address": [ ipv4Address
| ipv4MappedAddress | ipv6Address ] ]
ipv4Address := '"' decimal '.' decimal '.' decimal '.' decimal '"'
ipv4MappedAddress := '"' 'ff::' decimal '.' decimal '.' decimal '.'
decimal '"'
ipv6Address := '"' hexadecimal ':' hexadecimal ':' hexadecimal ':'
hexadecimal ':' hexadecimal ':' hexadecimal ':' hexadecimal ':'
hexadecimal '"'
```

Example 2-3 Proxy 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" } ]
```

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" } ]
```

Reverse Proxy Support

Learn how to configure your reverse proxy servers.

Reverse Proxy allows a single point of contact for various microservices associated with an Oracle GoldenGate MA deployment.

You can configure a proxy server depending on your environment setup and network requirements. Reverse proxy is optional, however, it is recommended to ensure easy access to microservices and provide enhanced security.

Reverse Proxy Support

Oracle GoldenGateMA can be configured to use a reverse proxy. Oracle GoldenGate MA includes an application called `ReverseProxySettings` that generates configuration file for a reverse proxy server. For example, the Administration Server is available on `https://goldengate.example.com:9001` and the Distribution Server is on `https://goldengate.example.com:9002`. With reverse proxy, all the microservices can be accessed from a single address, for example, `https://goldengate.example.com`.

The `ReverseProxySettings` application has two additional parameters in Oracle GoldenGate12*c* (12.3.0.1) and later:

- `-P`: Password for a Service Manager account.
- `-u`: Name of the Service Manager account to use.

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

Prerequisites

If you need to use a reverse proxy service with MA, use Nginx. Its a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server.Oracle GoldenGate MA is shipped with a utility to configure Nginx reverse proxy.

Here are the prerequisites for configuring Nginx-based reverse proxy:

- Install Nginx: For Oracle Linux, the command to install Nginx is:

  ```
  yum –y install nginx
  ```

  For more information about installing Nginx, see Installing Nginx Reverse Proxy
- Check the JRE version to be JRE 8.
- Install Oracle GoldenGate MA.
- Create one or more active MA deployments.

Configuring Nginx-based Reverse Proxy Example

An Oracle GoldenGate MA installation includes the `ReverseProxySettings` application in the `$OGG_HOME/lib/utl/reverseproxy/`directory. You can see the list of options available with the application, using the `–help` command.

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

  Usage: proxysettings [options] service-manager-url

  Options:

    -o, --output   Output file name (default is ogg.conf)
    -l, --log      Log file name (default is no logging)
    -t, --type     Proxy server type (default is nginx)
    -s, --no-ssl   Configure without SSL
    -h, --host     Virtual host name for reverse proxy
    -p, --port     Reverse proxy port number (defaults to 80 or 443)
    -?, --help     Display usage information
    -v, --version  Display version
```

Follow these steps to configure a reverse proxy:

1. To create the settings file for Nginx:

   ```
   $OGG_HOME/lib/utl/reverseproxy/ReverseProxySettings -u adminuser -P
   adminpwd -o ogg.conf http://localhost:9100
   ```

   If using a secure configuration (no -s is used), you have to use `https` instead of `http`.

2. Replace the existing Nginx configuration with the configuration required for MA deployment:

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

> **Note:**
>
> There shouldn't be any configuration file in the directory. If there is a `default.conf` file, you must rename it to `default.conf.bak`.

3. Create your self-signed certificate for Nginx, using the following command:

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

4. Test the new Nginx configuration using the following command:

```
sudo nginx -t

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

5. As root, reload Nginx and the new configuration:

```
sudo nginx -s reload
```

6. Use Curl to verify that reverse proxy is working:

```
curl -sv http://localhost/services/v2
{"$schema":"api:version","catalog":{"links":[
{"href":"http://localhost/service s/v2/metadata-
catalog","rel":"canonical"}]},"isLatest":true,"lifecycle":"active","link
s":[
{"href":"http://localhost/services/v2","mediaType":"application/js
on","rel":"canonical"},
{"href":"http://localhost/services/v2","mediaType":"app lication/
json","rel":"self"}],"version":"v2"}
```

> **Note:**
>
> If the deployments associated with the target Service Manager change, the Nginx configuration file must be re-generated and reloaded.

3
Authentication and Authorization

The MA security and authorization model declares and defines how communication security (confidentiality and Integrity) and Authorization (authentication and permissions) are configured and implemented.

All the security and authorization 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.

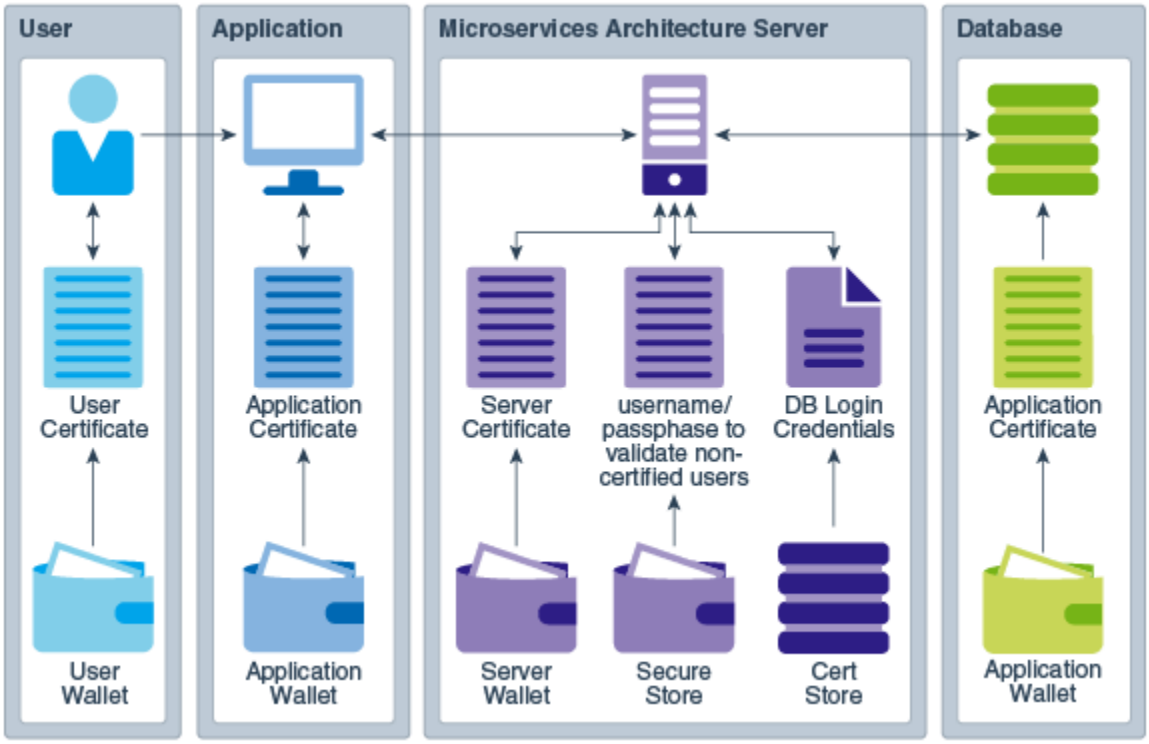
The MA defines a model and infrastructure for building service-aware applications. This model is not a generalized model, but one targeted at the current and future Oracle GoldenGate products that need to operate and integrate into global, cloud-based deployment environments. Oracle GoldenGate server programs are implemented using the MA infrastructure. All security and configuration implementations provided by the MA are common services.

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 passphrase 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 illustration depicts the relationship between the user, application, server, and database.



The following types of certificates are used for authentication:

- **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.
- **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.
- **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. MA Server database actions 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
MA server database authentication	This configuration sets the MA 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
MA server database 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.

Oracle UTL_HTTP Authentication

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_HTTP` Oracle Database package may need to configure the client database security environment to enable the use of Client-side certificates for authentication in UTL_HTTP.

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

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

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.

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 mode is set when configuring an Oracle GoldenGate MA deployment.

Authorization Mode ID	Notes
`server_only`	Only validate Server certificates. The Server certificates are required. The Client certificates are ignored.
`client_server`	Validate both Client and Server certificates. Both certificates are required.
`clientOptional_server`	This is the default. 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
User	Allows information-only service requests, which do not alter or effect the operation of either the MA. Examples of Query/Read-Only information include performance metric information and resource status and monitoring information.
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.
Administrator	Grants full access to the user, including the ability to alter general, non-security related operational parameters and profiles of the server.
Security	Grants administration of security related objects and invoke security related service requests. This role has full privileges.

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

Authorization for WebSockets

Learn how you can use WebSocket authorization.

REST API calls are made using standard HTTP request and take advantage of the authorization mechanism described in 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 HTTP authorization mechanism.

Native HTTP Authorization

The native HTTP authorization includes a header in the initial WebSocket establishment request. 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 Example

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

Error Codes

Review the MA HTTP error codes.

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

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 privileges 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 at:

https://tools.ietf.org/html/rfc7231

Cross Site Request Forgery

Learn how to avoid client-side attacks.

Cross Site Request Forgery (CSRF) is a client-side attack where a malicious or unauthorized website attempts to cause the client browser to perform or issue a compromising action or request against a protected server-side resource using a valid

user or client authorization object. The attack is limited to the actions and resources published by the attacked website.

Mode of Attack

A general mode of attack is for a malicious agent to cause a user's browser to be redirected to a malicious website. The malicious resource at this malicious site causes the user's browser to download a client-side script (JavaScript). This downloas causes the user's browser to issue a compromised request against a protected website that the user has obtained prior authorization. The browser issues the compromised request delivering both the malicious script's request payload along with any authorization cookies that are automatically conveyed with the request.
For example, the malicious website's script instructs the user's browser to request the addition of a new user with a high security clearance. The request is issued to the protected website along with current browser user's current authorization cookie. This cookie is delivered automatically and transparently with the malicious request. The request with the valid user authorization is forged by a script that is retrieved from different redirected malicious site and issues a malicious request under the authorization context of the current browser user.

Taking Defensive Measures

In response to the CSRF threats, the compliant browsers implement a mechanism so that cross-site information is limited and additional information regarding the requesting browsers environment is included.

When scripts are executed that have been retrieved from a site other than the script's request is targeting, then the browser only allows the following allowed methods to be explicitly defined:

```
GET
HEAD
POST
```

Other than the HTTP headers that are automatically set by the browser, the only HTTP headers allowed to be explicitly set are the CORS-safelisted request-header (simple header):

```
Accept
Accept-Language
Content-Language
Content-Type
Last-Event-ID
DPR
Save-Data
Viewport-Width
Width
```

The `Content-Type` header is only allowed to declare the following:

```
application/x-www-form-urlencoded
multipart/form-data
text/plain
```

No event listeners can be registered with a `XMLHttpRequestUpload` object nor are any `ReadableStream` instances allowed or used in the request.

CSRF Mitigation Strategy

Requests issued from scripts are not retried from the same site as the current target request includes one or more of the following:

`Origin` HTTP header – Always included in cross-site script requests.

`Referer` HTTP header – Included if the request is from a referred parent page. (Note that `Referer` is misspelled in the Remote Function Call).

If a proxy or reverse proxy is between the requesting client and the target website, then the proxy or reverse proxy must be configured to include the following extended HTTP headers:

`X-Forwarded-Host` – The original hostname the request to which the request was targeted (the proxy or reverse proxy host). The `X-Forwarded-Host` should replace the `Origin` header on propagated requests, but contain the same information.

`X-Forwarded-Server` – The hostname of the proxy or reverse proxy server.

This is the strategy in order of evaluation:

1. If the `Origin` HTTP header exists, then verify that the `Origin` hostname matches the origin server's hostname.
2. If the `Referer` HTTP header exists, then verify that the `Origin` HTTP header also exists and that the hostname value for both the `Origin` and `Referer` HTTP headers match.
3. If the `X-Forwarded-Host` HTTP header exists, then verify that the `X-Forwarded-Server` HTTP header also exists and that the hostname value for both the `X-Forwarded-Host` and `X-Forwarded-Server` HTTP headers match.
4. If neither the `Origin` header nor the `X-Forwarded-Host` HTTP headers exist, the request is presumed not to be originating as a Cross Site Request. This places a reliance on the compliance of the browser to support Cross Site Scripting (XSS) policies.

> **Note:**
>
> Because of the reliance on the XSS policy support in the client, malicious CSRF requests from general purpose non-browser clients (like cURL, Wget, Python, Perl, and eNetcat) can not be protected against.

4
Communication Security

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

Topics:

Certificate Access Control List

Learn how you can refine communication security.

The communication security accepts any valid certificate during the connection handshake process. However, you may need to filter and reject otherwise valid certificates based on internal policies. For example, Finance may want to reject certificates issued to Human Resources even though the Human Resources certificates are cryptographically valid. 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 4-1 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 4-2 Deny certificates issued from `Deploy2`

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

Example 4-3 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" } ]
```

Transport Layer Security Protocols and Ciphers

Review the supported security protocols.

Transport Layer Security (TLS) Protocols

The following are the supported security protocol versions and these are the available values that you can use when setting the `/config/securityDetails/network/common/protocolVersion` security setting.

Protocol Version ID	Notes
`3_0_With_2_0_Hello`	
`3_0_Only`	
`2_0`	Considered deprecated.
`3_0`	
`1_0`	
`1_0_Or_3_0`	

Protocol Version ID	Notes
1_0_Or_3_0_Or_2_0	
3_0_Or_2_0	
1_1	
1_2	
1_1_Or_3_0	
1_2_Or_3_0	
1_1_Or_1_0	
1_2_Or_1_0	
1_2_Or_1_1	
1_1_Or_1_0_Or_3_0	
1_2_Or_1_0_Or_3_0	
1_2_Or_1_1_Or_1_0	Oracle Recommends
1_2_Or_1_1_Or_3_0	
1_2_Or_1_1_Or_1_0_Or_3_0	

Your testing must ensure that all clients used for a particular TLS protocol version support the TLS version being tested because verification of client support for TLS version support is required. Diagnostically, the server log should be reviewed for the handshake protocol processing. The log should contain the protocol version being negotiated. If the client does not support the protocol version that the server is configured for, the server terminates the connection. You may not see an error message or indication overtly sent to the client that a protocol version failed. The failure may appear to the client as a network connection rejection or a certificate failure depending on how the client is set to handle the exception.

> **Note:**
>
> TLS protocols below the 1.0 version should not be used because of documented security defects.

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.

Cipher Suite ID	Notes
TLS_NO_SUCH_CIPHERSUITE	
TLS_RSA_WITH_RC4_128_MD5	
TLS_RSA_WITH_RC4_128_SHA	

Cipher Suite ID	Notes
TLS_RSA_WITH_3DES_EDE_CBC_SHA	Federal Information Processing Standards (FIPS) Compliant
TLS_RSA_WITH_DES_CBC_SHA	
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	
TLS_DH_anon_WITH_RC4_128_MD5	
TLS_DH_anon_WITH_DES_CBC_SHA	
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	FIPS Compliant
TLS_RSA_WITH_AES_128_CBC_SHA	FIPS Compliant
TLS_RSA_WITH_AES_256_CBC_SHA	FIPS Compliant
TLS_RSA_WITH_AES_128_CBC_SHA256	FIPS Compliant
TLS_RSA_WITH_AES_128_GCM_SHA256	FIPS Compliant
TLS_RSA_WITH_AES_256_GCM_SHA384	FIPS Compliant
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	FIPS Compliant Elliptic Curve Cryptography (ECC) ciphers
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	FIPS Compliant ECC ciphers
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	FIPS Compliant ECC ciphers
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	FIPS Compliant ECC ciphers
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	FIPS Compliant ECC ciphers
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	FIPS Compliant(?) ECC ciphers

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.

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.

For example:

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=US/ST=CA/L=Redwood Shores/O=Oracle Corp/OU=Corporate Security/
OU=Deployment Security/CN=Deploy1
        Last Update: Feb 22 19:20:34 2017 GMT
        Next Update: Mar 24 19:20:34 2017 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:7C:A0:BB:FB:6F:75:70:4B:B4:95:18:54:9C:1F:88:2E:A1:1B:EF:E4

            X509v3 CRL Number:
                4097
Revoked Certificates:
    Serial Number: 1000
        Revocation Date: Feb 22 19:20:34 2017 GMT
    Signature Algorithm: sha256WithRSAEncryption
         a6:e5:75:62:93:49:26:6e:79:f1:dd:90:94:bb:99:1c:3a:24:
         99:63:82:d6:f1:56:72:98:cc:8f:6f:61:b8:a4:dd:21:0f:ae:
         fa:38:78:c0:c9:bc:bc:87:61:15:35:e7:20:b8:5e:8f:6a:0a:
         e1:58:e0:30:6d:df:03:8f:6f:de:0a:54:1c:f0:44:e5:28:48:
         56:23:00:60:19:dd:e2:68:2d:35:2b:cc:62:85:b6:34:32:ce:
         c3:f6:8a:b0:bb:b4:66:0e:85:8c:79:b2:32:5c:65:ac:47:99:
         69:c5:bf:bb:ec:1e:7f:40:e2:1f:11:fa:2a:7c:d3:94:de:62:
         e2:8b:de:15:04:2c:67:14:2e:b7:71:29:d5:e2:e1:ee:ac:c3:
         a3:d0:20:41:a9:e0:6a:5b:90:28:35:5a:90:86:51:69:df:27:
         af:3e:0f:c0:d2:32:ab:d2:7a:c5:16:29:f6:ec:04:dd:e7:6d:
         8b:10:06:40:c0:08:32:39:50:33:c0:b9:86:b9:77:19:6f:a6:
         49:65:54:f5:35:c8:27:08:f6:fa:91:3c:ae:2c:b5:c1:52:de:
         42:2c:65:6c:ce:97:52:50:00:53:df:6d:1d:e6:38:9f:61:97:
         d9:aa:60:1c:06:24:aa:f3:ac:8c:d6:85:ed:83:20:2f:50:5c:
         f6:af:78:91:49:a5:b7:cb:96:6c:03:3a:e3:3d:dd:a9:d5:0f:
         5f:3c:47:8c:78:33:65:09:65:8a:08:92:19:58:a1:93:7f:99:
         ee:9d:f1:4a:30:21:63:24:5a:d4:6b:bd:e0:ec:0c:79:09:1f:
         48:a6:39:87:92:0b:f7:25:8e:31:65:ee:10:28:45:bb:55:9c:
         c8:64:49:fe:1d:78:6d:9a:09:67:6b:76:f4:3f:6a:b8:eb:c0:
         0b:0c:ab:92:6d:f5:60:06:34:0f:ef:65:be:c8:af:1d:67:bc:
         36:b7:d1:c0:ea:30:71:3b:2b:ba:16:dc:72:86:90:32:e3:59:
         99:2c:33:7a:2f:63:77:ec:0d:70:89:52:0f:8f:29:13:fd:17:
         18:49:56:65:8d:23:64:ba:e9:b6:74:56:40:9b:1c:65:17:ef:
         bd:2c:77:d4:69:f6:f4:eb:df:a9:31:14:89:fc:1d:24:81:7d:
         85:ba:1d:8f:8b:1b:0d:c2:a3:c2:ea:a5:6e:a2:a7:be:34:16:
         a1:b8:16:a4:f2:32:5a:65:2d:85:14:be:73:6b:de:40:13:bd:
```

```
        f1:3d:7e:65:14:3c:a8:ad:b7:4e:cb:41:53:f4:24:5a:4f:a1:
        56:b6:33:65:f9:ef:b9:40:2d:26:ee:ba:57:d5:f5:75:1b:60:
        8d:f2:24:36:e5:2a:c8:b3
-----BEGIN X509 CRL-----
MIIDKjCCARICAQEwDQYJKoZIhvcNAQELBQAwgZYxCzAJBgNVBAYTAlVTMQswCQYD
VQQIDAJDQTEXMBUGA1UEBwwOUmVkd29vZCBTaG9yZXMxFDASBgNVBAoMC09yYWNs
ZSBDb3JwMRswGQYDVQQLDBJDb3Jwb3JhdGUgU2VjdXJpdHkxHDAaBgNVBAsME0Rl
cGxveW1lbnQgU2VjdXJpdHkxEDAOBgNVBAMMB0RlcGxveTEXDTE3MDIyMjE5MjAz
NFoXDTE3MDMyNDE5MjAzNFowFTATAgIQABcNMTcwMjIyMTkyMDM0WqAwMC4wHwYD
VR0jBBgwFoAUfKC7+291cEu0lRhUnB+ILqEb7+QwCwYDVR0UBAQCAhABMA0GCSqG
SIb3DQEBCwUAA4ICAQCm5XVik0kmbnnx3ZCUu5kcOiSZY4LW8VZymMyPb2G4pN0h
D676OHjAyby8h2EVNecguF6PagrhWOAwbd8Dj2/eClQc8ETlKEhWIwBgGd3iaC01
K8xihbY0Ms7D9oqwu7RmDoWMebIyXGWsR5lpxb+77B5/QOIfEfoqfNOU3mLii94V
BCxnFC63cSnV4uHurMOj0CBBqeBqW5AoNVqQhlFp3yevPg/A0jKr0nrFFin27ATd
522LEAZAwAgyOVAzwLmGuXcZb6ZJZVT1NcgnCPb6kTyuLLXBUt5CLGVszpdSUABT
320d5jifYZfZqmAcBiSq86yM1oXtgyAvUFz2r3iRSaW3y5ZsAzrjPd2p1Q9fPEeM
eDNlCWWKCJIZWKGTf5nunfFKMCFjJFrUa73g7Ax5CR9IpjmHkgv3JY4xZe4QKEW7
VZzIZEn+HXhtmglna3b0P2q468ALDKuSbfVgBjQP72W+yK8dZ7w2t9HA6jBxOyu6
FtxyhpAy41mZLDN6L2N37A1wiVIPjykT/RcYSVZljSNkuum2dFZAmxxlF++9LHfU
afb069+pMRSJ/B0kgX2Fuh2PixsNwqPC6qVuoqe+NBahuBak8jJaZS2FFL5za95A
E73xPX5lFDyorbdOy0FT9CRaT6FWtjNl+e+5QC0m7rpX1fV1G2CN8iQ25SrIsw==
-----END X509 CRL-----
```

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.SCA/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.

HTTP Security and Cache Headers

Review the supported security and cache headers.

The MA server accepts and returns HTTP envelopes that contain a set of headers that govern how the server, the client, and proxies handle the HTTP contents. For HTTP 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 issue 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, exposes the embedded site to Clickjack 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
```

5

Server and Deployment Identities

You must uniquely identify MA servers and deployments using schemes.

In a Common-Named Multiple Server and Deployment configuration that has more than one MA deployment within an environment access by a common name, each server and deployment must be uniquely identifiable. This identity allows coordination services, peers servers, and orchestration ecosystems to differentiate one deployment and server from another when necessary.

Topics:

Using a Universally Unique IDs Scheme

Universally Unique IDs are synonymous with Globally Unique IDs (UUID/GUID).

These IDs provide a standardized format for creating and interpreting identifiers that is a 128–bits long, RFC4122. It can guarantee uniqueness across space and time. Several operating systems provide mechanisms to generate UUIDs including:

```
$ cat /proc/sys/kernel/random/uuid
$ uuidgen
```

UUID can be used to identify distinct deployments and even specific servers within a deployment. The primary issue with UUIDs is that once generated, they can not be regenerated. If the UUID value is lost, there is no way of deterministically recreating it. This is an issue if the UUID is used in a distributed fashion and it is held as reference to a specific deployment. If that deployment loses the value of it originally generated UUID, there is no way of regenerating the UUID. You must take care when safeguarding the UUID.

Using a Deterministically Calculated Unique ID Scheme

A deployment's identity can be deterministically calculated and be unique within a local scope.

This would create a unique ID based on a combination of hardware and file system signatures. For example, the calculated ID could be generated based on the MAC Address of the network interface and the real absolute file system paths that make-up the deployment. Any relocation of the deployment within the file system invalidates the deterministic regeneration of the ID, as would any change in the network interface.

Using an Explicit Naming Scheme

You can use explicit naming to avoid the problem of guaranteed uniqueness to administrators.

While this addresses any potential shortcoming of other schemes with recreation of an ID, this scheme is not recommended in large organizations with a large deployment count.

Creating Server and Deployment IDs

A `serverID` and `deploymentID` is required for each of your servers and deployments. Deployment and server UUIDs are generated by default if you don't define them.

In general, these values are not changed once assigned. These values are also of limited use directly because their application is dependent on the context and requirements of the request or operation.

`serverID`

Each servers generates a unique ID during first start if it finds an absent or null server ID. The server ID is then used to generate a short unique label that can be used as name or tag in cases where the 3- character UUID is to long. Both the `serverID` and short name are expected to be globally unique. It can be used to identify a server without prefixing it with a deployment. The `serverID` is held in each server's local configuration context and only accessible by the owning server.

You can use the `serverID` to limit certain request or action targets to only the server. For example, by including the `serverID` in server generated payloads, the server can validated that it was the originator of the payload by comparing the presented `serverID` with the held `serverID`.

```
"config" : {
     "serverID": "96bc6cab-abb8-4a05-aeff-6d0d385262af"
     "serverIDShortLabel": "lrxsq6u4SgWu/20NOFJirw"
}
```

`deploymentID`

The first server starting within a deployment generates a unique ID if it finds an absent or null deployment ID. The `deploymentID` is a a containment ID and serves to identify a group of related servers. The `deploymentID` is held in the deployment global configuration context and is accessible by all servers within that deployment.

The `deploymentID` can be used to limit server requests or actions to only the servers within the deployment. For example, by including the short label version of the `serverID` in UDP/UDT data, a server can filter and qualify only the information that originated from a server within its own deployment.

```
"global": {
     "deploymentID": "f1df4a18-d0a8-4ba1-9ad0-18da9458baef"
}
```

affiliateDeploymentIDs

Affiliated deployments are deployments that coordinate or cooperate share specific information or contexts. The `affiliateDeploymentIDs` value is a JSON Array type that is initially null (empty). As deployments define actions or operations that should apply to or be valid in other deployments, the `deploymentIDs` of the affiliated deployments are added to the JSON Array of the `affiliateDeploymentIDs`. Specific behaviors or actions can qualify a presented `deploymentIDs` against the list of `affiliateDeploymentIDs` and grant access or operation to the foreign deployment.

For example, an authorization cookie includes a `deploymentIDs` as part of its specification. If the authorization cookie is presented to an foreign deployment that has the originating deployment's `deploymentIDs` listed in its `affiliateDeploymentIDs` list, then the authorization cookie is qualified rather then being filtered out as not originating from the receiving servers.

```
"global": {
     "affiliateDeploymentIDs": [ "deafa2f6-6ee7-48b1-862a-97a9b6d5b9df" ]
}
```

You can update the global configuration using either a bootstrap configuration file or the command-line overrides, for example:

```
$ bin/adminsrvr '{"global": { "affiliateDeploymentIDs":
["deafa2f6-6ee7-48b1-862a-97a9b6d5b9df"]}}'
```

6
Securing Deployments

You can choose to set up a secure or non-secure deployment. A secure deployment involves making RESTful API calls and conveying trail data between the Distribution Server and Receiver Server, over SSL/TLS. You can use your existing wallets and certificates, or you can create new ones.

The instructions for securing deployments is in Setting Up Secure and Non-Secure Deployments in *Using the Oracle GoldenGate Microservices Architecture*.

Part II
Securing Oracle GoldenGate

Use this part to secure your CA and MA environments.

Topics:

- Overview of Security Options
  You can use these security features to protect your Oracle GoldenGate environment and the data that is being processed.
- 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:
- 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.
- 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.
- 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.
- Populating an ENCKEYS File with Encryption Keys
  Learn how to use an `ENCKEYS` file.
- 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 initiate passive and alias connections between your source and target systems.
- 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.

7

Overview of Security Options

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

Security Feature	What it Secures	Supported Databases	Description
Data Encryption Two methods are available: • Encrypting Data with the Master Key and Wallet Method • Encrypting Data with the ENCKEYS Method	• Data in the trails or an extract file • Data sent across TCP/IP networks	Master key and wallet method is the preferred method on platforms that support it. Not valid for the DB2 for i, DB2 z/OS, and NonStop platforms. `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)[1] cipher: AES-128 AES-192 AES-256 • Blowfish[2]
Credential Store Identity Management Managing Identities in a Credential Store	User IDs and passwords (credentials) assigned to Oracle GoldenGate processes to log into a database.	Credential store is the preferred password management method on platforms that support it. Not valid on the DB2 for i, DB2 z/OS, , and NonStop platforms.	User credentials are maintained in secure wallet storage. Aliases for the credentials are specified in commands and parameters.
Password Encryption See Encrypting a Password in a Command or Parameter File.	Passwords specified in commands and parameter files that are used by Oracle GoldenGate processes to log into a database.	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.	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
Command Authentication See Configuring GGSCI Command Security.	Oracle GoldenGate commands issued through GGSCI.	Valid for all Oracle GoldenGate-supported databases and platforms.	Stores authentication permissions in an operating-system-secured file. Configure a `CMDSEC` (Command Security) file.
Trusted Connection See Using Target System Connection Initiation.	TCP/IP connection to untrusted Oracle GoldenGate host machines that are outside a firewall.	Valid for all Oracle GoldenGate-supported databases and platforms.	Use any of the following: • AES-128 • AES-192 • AES-256 • Blowfish

Security Feature	What it Secures	Supported Databases	Description
Manager Security Securing Manager	Access rules for Manager.	Valid for all Oracle GoldenGate-supported databases and platforms.	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.
CryptoEngine	Allows you to select the cryptographic library that better suits your needs: Portability (Classic), Portability and compliance with FIPS-140 standard (FIPS140), or enhanced throughput (Native).	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).	Selects which cryptographic library the Oracle GoldenGate processes will use.

1 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. 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.

2 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.

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.
- To encrypt data across the network, Oracle GoldenGate generates a session key using a cryptographic function based on the master key.

Oracle GoldenGate uses an auto-login wallet (file extension `.sso`), meaning that it 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 DB2 for i, DB2 z/OS, or NonStop platforms.

Topics:

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 i.
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` and compare the AES hash value with the one that you originally recorded. All wallets must show identical key versions and hash values.

   ```
   INFO MASTERKEY VERSION version
   ```

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: 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.)

   ```
   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.

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 must be stopped. This procedure includes steps for performing those tasks in the correct order.

1. Stop Extract.

   ```
   STOP EXTRACT group
   ```

2. On the target systems, issue the following command for each Replicat until it returns `At EOF`.

   ```
   SEND REPLICAT group STATUS
   ```

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

   ```
   STOP EXTRACT group
   ```

4. On the target systems, stop the Replicat groups.

```
STOP REPLICAT group
```

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

   ```
   OPEN WALLET
   ```

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

   ```
   INFO MASTERKEY
   ```

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

   ```
   RENEW MASTERKEY
   ```

8. 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.

9. 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
   ```

10. Copy the updated wallet from the source system to the same location as the old wallet on all of the target systems.

11. 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
    ```

12. Restart Extract.

    ```
    START EXTRACT group
    ```

13. Restart the data pumps.

    ```
    START EXTRACT group
    ```

14. Restart Replicat.

    ```
    START REPLICAT group
    ```

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 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.

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.

Topics:

Encrypting the Data with the ENCKEYS Method

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 AES128 as the default for all database types except the iSeries, z/OS, and NonStop platforms, where BLOWFISH is the default.
- AES128 encrypts with the AES-128 encryption algorithm. Not supported for iSeries, z/OS, and NonStop platforms.
- AES192 encrypts with AES-192 encryption algorithm. Not supported for iSeries, 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 z/OS and DB2 for i. 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

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 Encrypting the Data with the ENCKEYS Method.

```
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.

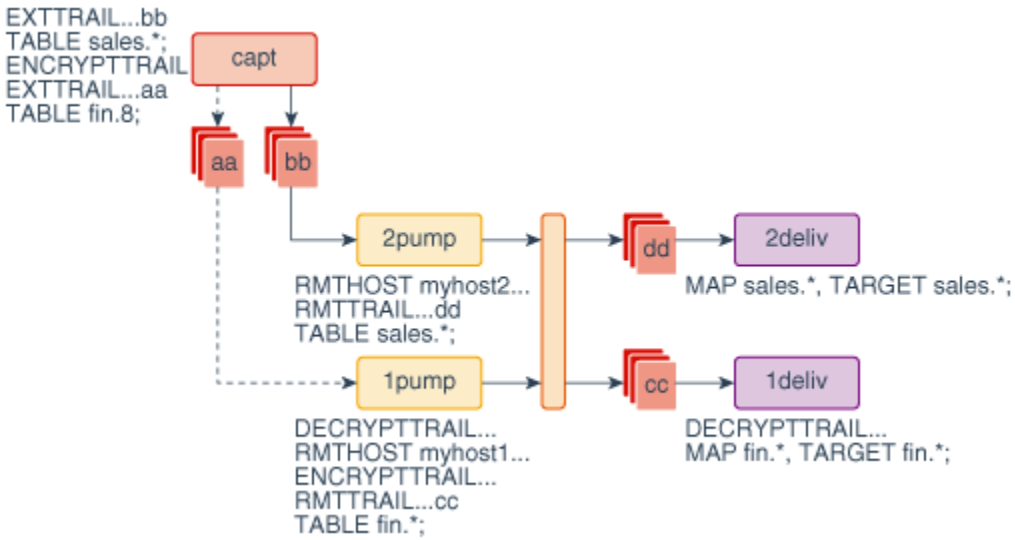
To Decrypt Data for Processing by Replicat

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.

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 specification(s). 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 9-1 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 9-2 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 9-3 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 9-4 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 9-5 Replicat 2 (on myhost2) parameter file

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

10
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:

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. (See *Administering Oracle GoldenGate for Windows and UNIX* for more information about 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*.

For more information about the commands used in this procedure and additional credential store commands, see *Reference for Oracle GoldenGate for Windows and UNIX*.

Specifying the Alias in a Parameter File or Command

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

Table 10-1 Specifying Credential Aliases in Parameters and Commands

Purpose of the Credential	Parameter or Command to Use
Oracle GoldenGate database login[1]	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*

[1] Syntax elements required for USERIDALIAS vary by database type. See *Reference for Oracle GoldenGate for Windows and UNIX* for more information.

11

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 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.

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:

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, AES128 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.

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 shown in Table 11-1. Option descriptions follow the table.

Table 11-1 Specifying Encrypted Passwords in Parameters and Commands

Purpose of the Password	Parameter or Command to Use
Oracle GoldenGate database login[1]	`USERID` *`user`*`, PASSWORD` *`password`*`, &` *`algorithm`* `ENCRYPTKEY {`*`keyname`* `| DEFAULT}`

Oracle GoldenGate database login for Oracle ASM instance	`TRANLOGOPTIONS ASMUSER SYS@`*`ASM_instance_name`*`, &`
	`ASMPASSWORD` *`password`*`, &`
	`algorithm` `ENCRYPTKEY {`*`keyname`* `| DEFAULT}`

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

Password substitution for `{CREATE | ALTER} USER` *`name`* `IDENTIFIED BY` *`password`*	`DDLOPTIONS DEFAULTUSERPASSWORD` *`password`* `&` *`algorithm`* `ENCRYPTKEY {`*`keyname`* `| DEFAULT}`
Oracle TDE shared-secret password	`DBOPTIONS DECRYPTPASSWORD` *`password`*[2] *`algorithm`* `&` `ENCRYPTKEY {`*`keyname`* `| DEFAULT}`
Oracle GoldenGate database login from GGSCI	`DBLOGIN USERID` *`user`*`, PASSWORD` *`password`*`, &` *`algorithm`* `ENCRYPTKEY {`*`keyname`* `| DEFAULT}`

Table 11-1 (Cont.) Specifying Encrypted Passwords in Parameters and Commands

Purpose of the Password	Parameter or Command to Use
Oracle GoldenGate database login to a downstream Oracle mining database from GGSCI	`MININGDBLOGIN USERID user, PASSWORD password,& algorithm ENCRYPTKEY {keyname | DEFAULT}`

[1] Syntax elements required for USERID vary by database type. See *Reference for Oracle GoldenGate for Windows and UNIX* for more information.
[2] 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.
- *password* is the encrypted password that is copied from the ENCRYPT PASSWORD command results.
- *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:

```
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
```

12

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 Encrypting the Data with the ENCKEYS Method)
- ENCRYPT PASSWORD with ENCRYPTKEY *keyname* (see Encrypting a Password in a Command or Parameter File)
- RMTHOST or RMTHOSTOPTIONS with ENCRYPT (see Encrypting the Data with the ENCKEYS Method)

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

Topics:

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)

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
```

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 will be 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.
   ```

13
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:

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 13-1 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 13-2 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 13-2 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.

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 GoldenGate recommends denying write and delete access to everyone but Oracle GoldenGate Administrators.

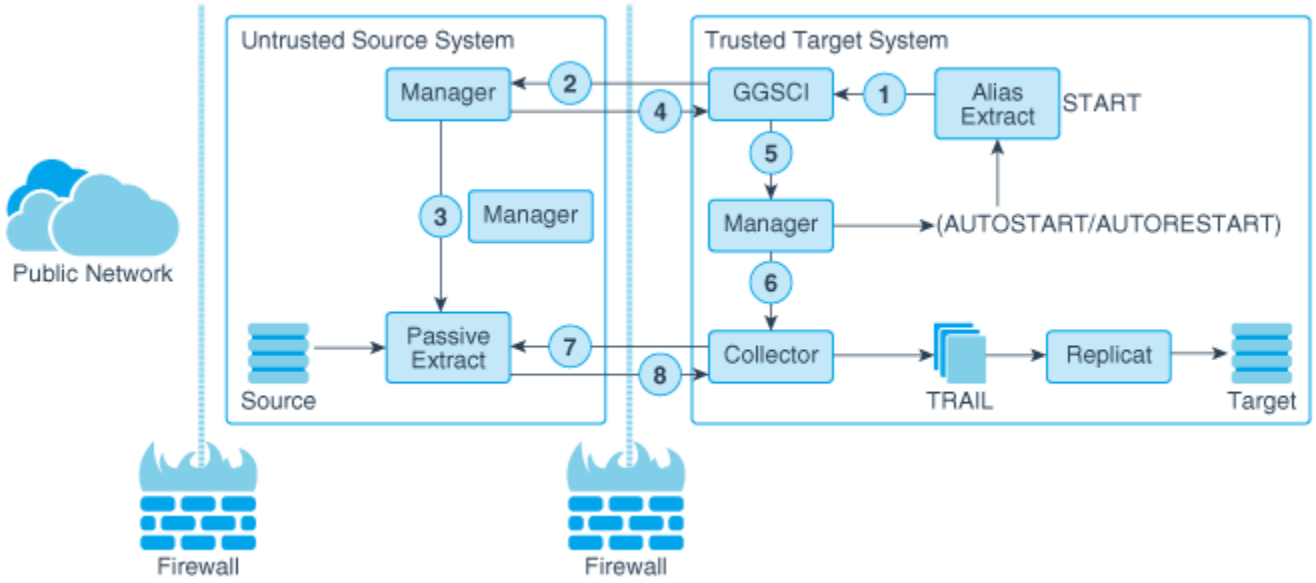
14

Using Target System Connection Initiation

Learn how to initiate passive and alias connections between your source and target systems.

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:

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 for Windows and UNIX*.

For more information about configuring an Extract group, see *Administering Oracle GoldenGate for Windows and UNIX*.

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, see *Administering Oracle GoldenGate for Windows and UNIX*.

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.

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.

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.

15
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
```

For information about the ACCESSRULE options, see *Reference for Oracle GoldenGate for Windows and UNIX*