Quintessence Labs

APPLICATION NOTE

qCrypt Key Management Server and qClient KMIP C SDK

Performance Measurement and Optimisation Tips

Overview

This document discusses methods of optimising performance of applications that communicate with the qCrypt Key Manager over KMIP. Performance measurements and qClient C SDK source code examples are presented.

Intended Audience: Application developers, system architects and operations personnel.
Assumptions: Familiarity with networking, access to VM or physical deployment of qCrypt appliances, and an ability to build and run C programs using the qClient C SDK.

Introduction

The qCrypt Key Manager conforms to the OASIS KMIP standard. This requires that KMIP client-server communication is protected over a mutually authenticated TLS channel. During setup of a TLS session, handshake messages are exchanged that require public key cryptographic (PKC) operations. PKC operations are relatively complex due to the nature of the algorithms involved, and as such, can account for a significant portion of the time taken for the exchange of short messages. If the qCrypt appliance uses an HSM, then PKC operations involving the server's private key are performed within the HSM. This adds additional delay to the handshake.

This document presents performance measurements for a number of KMIP operations, and shows how load sharing, and connection pooling can dramatically improve performance.

Connection Establishment Overview

In order for a KMIP client to communicate with a KMIP server, a mutually authenticated TLS session must be established over a TCP connection. During the TLS handshake, both the client and server perform a number of PKC operations to authenticate each other. Additionally, the establishment of the secret key used to encrypt the communications between client and server involves PKC operations.

PKC operations use mathematical operations that are relatively CPU intensive. If a Hardware Security Module (HSM) is used to perform the PKC operations, then there can be additional relatively significant delays in moving data between host memory, and the HSM.

For short-lived sessions, the time to complete the TLS handshake can significantly impact system performance. This type of delay is not unique to KMIP and TLS. This is a well-known issue that has been around since the dawn of electronic communications, and later, computing. Commonly used solutions to this problem include load sharing, and connection pooling. These solutions can also be applied to improve KMIP's TLS session establishment.

In the case of load sharing, more resources are used. This can be thought of as simply providing more aggregate compute power that is shared across the entire load. If one resource is busy, another resource can provide service, concurrently.

In the case of connection pooling, also called connection caching, several connections are established, kept open, and re-used many times so that a new connection establishment handshake is not required for each transaction that takes place between the client and server.

Establishing a connection using the qClient SDK

The SDK function, `qlc_connect_key_manager()`, is used to establish a mutually authenticated TLS session between the client and KMIP server. `qlc_connect_key_manager()` takes two arguments. The first is the address of a pointer to a `*qlc_km_ctx_t` type. This is an opaque structure that holds context information related to the session between the client and server.

The second argument is an unsigned char pointer that contains connection information, including server IP address and port number, client private key, client certificate, and trusted CA certificate. In qClient sample code, this information is often stored in a file which is indicated by pre-pending an "@" symbol to the connection information.

```
Modules={
  Search_Path= 'E:\\usr\\local\\lib':'C:\\Windows\\System32'},
Presentation={
  Protocol=KMIP, Version='1.3':'1.2':'1.1':'1.0', Format=TTLV},
Session = {
  Protocol=SSL,
  Host='kmip.mydomain.com',
  Port='5696',
  Certificate=PEM:'E:/usr/local/etc/certs/my-cert.pem',
  CA_Cert=PEM:'E:/usr/local/etc/certs/ca-cert.pem',
  Public=PEM:'E:/usr/local/etc/certs/my-cert.pem',
  Private=PEM:'E:/usr/local/etc/certs/my-key.pem':'password',
  Authenticate}
```

Example qClient SDK configuration information

The code fragment below shows how `qlc_connect()` is used.

```
qlc_node_t *rsp = NULL;
qlc_km_ctx_t *ctx = NULL;
int ret = QLC_ERR_NONE;

rsp = qlc_connect_key_manager(&ctx, "@kmip.cfg");
if ((ret = qlc_ok(rsp)) != QLC_ERR_NONE)
{
  printf("Connection failed, error: %s\n", qlc_explain(rsp));
  goto end;
}
printf("Connection established with server\n");
qlc_release(rsp);
rsp = NULL;
```

Connection establishment example code fragment

When this code executes, qClient opens and parses the configuration file, and establishes a TCP connection with the server on the specified port – 5696 has been assigned by IANA for KMIP. After establishing the TCP connection, qClient begins the TLS handshake, and uses the credential files identified in the configuration file for mutual authentication. Additionally, qClient sends a KMIP Discover Versions request to the server. The response to this request tells the client the versions of KMIP that are supported by the server.

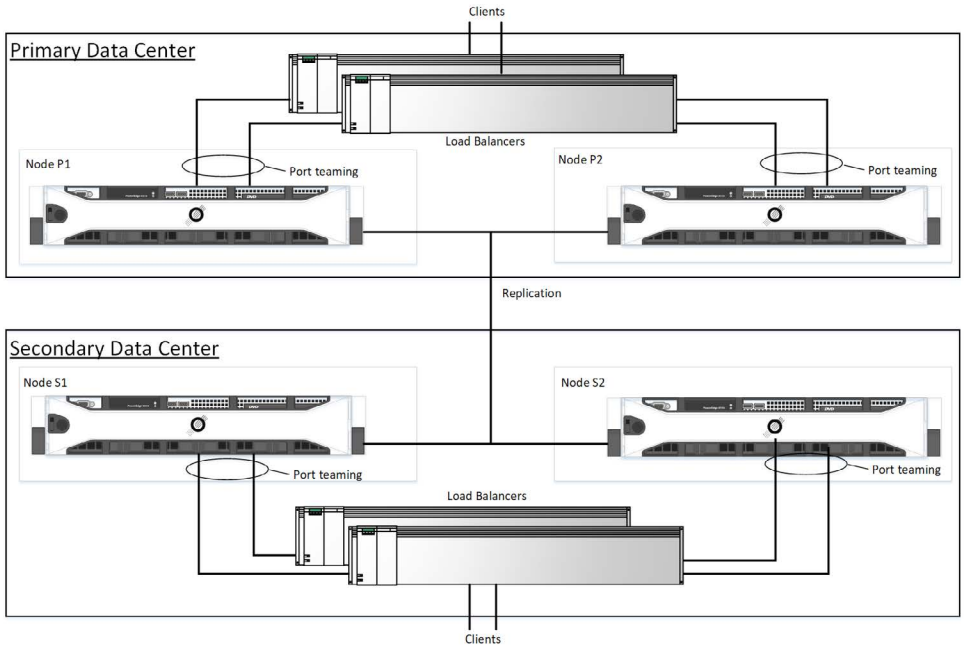
After successful establishment of a TLS session, the ctx structure can be used in `qlc_perform()`, and `qlc_execute()` functions to send KMIP requests to the server using the established TLS session.

When `qlc_disconnect_key_manager()` is called, the TLS session and TCP connection are shut down. In order for the client to communicate again with the server, it must establish a new TLS session by calling `qlc_connect_key_manager()`.

Deployment Architectures

Typically, for operational reasons, compute, storage, and network infrastructure are built to support high availability and resilience through replication of equipment in geographically separated data centers. Key management systems built using qCrypt key management servers can consist of one or more qCrypt appliances. Like other IT resources, replication of key management servers and geographic separation is recommended for availability and service resilience.

An added consideration with key management is the risk of key loss. The risk of loss of keys, as well as loss of operations is greatest when only a single key manager is deployed. Network isolation, power failure, and device malfunction can all lead to loss of service. In the worst case, keys may be lost as well, potentially leading to large losses of encrypted data.



Two-plus-two qCrypt replication deployment

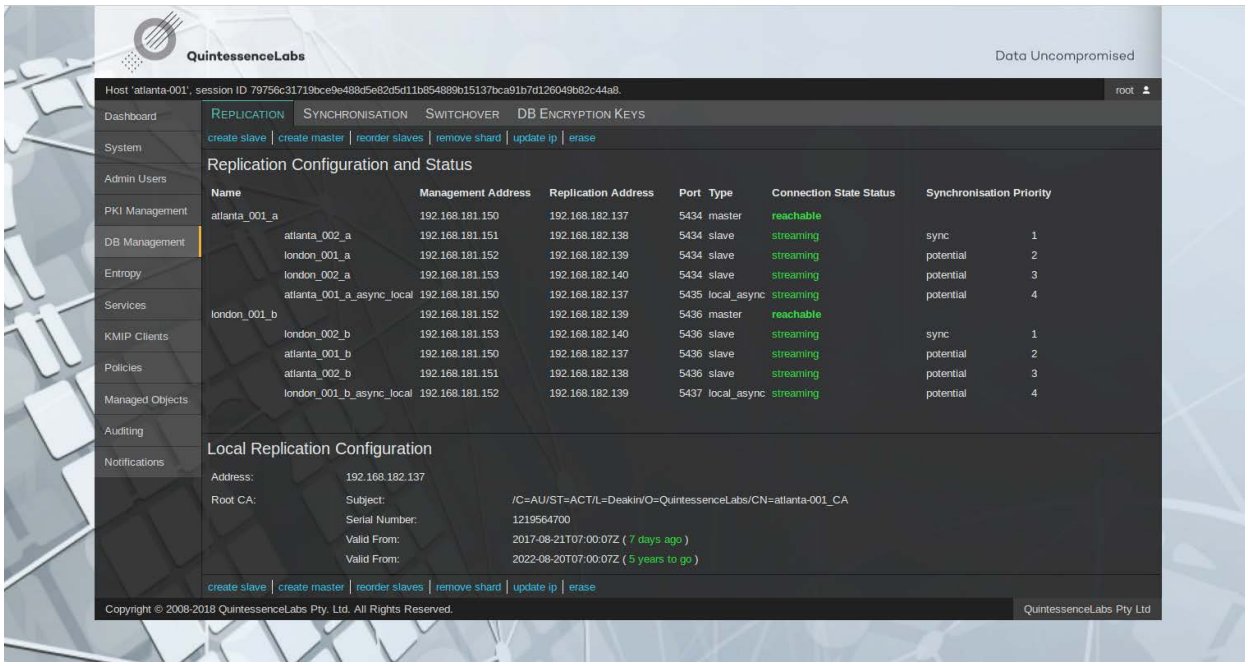
qCrypt key managers can be deployed in replicated pairs to support active-active, and active-standby system architectures. In this scenario, one key manager provides key management service, while a second operates as a hot standby. To mitigate the risk of key loss in failure scenarios, an active qCrypt key manager, when deployed for replication, always has one synchronous replication partner. The impact of network bandwidth and latency on performance needs to be considered carefully.

Additionally, for a two-node replication deployment, if one key management appliance goes offline – whether for maintenance, or due to network or device failure – the remaining key manager will not have a replication partner. The system will become vulnerable to key loss, just as in a single node deployment, in this failure scenario.

Recommended best practice for system architectures is deployment of no less than four qCrypt key managers. This supports service continuation in case of multiple failures, and also permits active-active service, either with two active nodes in one data center, or one active node in each of two data centers.

qCrypt fully supports load balancers, resulting in both automatic switching of client traffic to currently active key management nodes, and sharing of resources to improve overall system service.

The diagram above illustrates a "two-plus-two" deployment of qCrypt key managers, with two qCrypt nodes in each of two data centers. Load balancers and/or DNS can be used to switch traffic to active nodes. Any two nodes (two in primary data center, two in secondary data center, or one in each data center), can be configured to operate as active key managers, with the remaining two operating as hot standby nodes. To maintain best performance, synchronous replication should be configured between an active key manager node and a collocated key manager node.



Example of two-plus-two deployment configuration

The screen shot above shows an example of a two-plus-two configuration. Two nodes are deployed in a data center in Atlanta, and two nodes are deployed in a data center in London. At the time of the screen capture, one node in Atlanta (atlanta-001), and one node in London (london-001), were operating as masters. The atlanta-002 node, collocated with atlanta-001, is operating as a synchronous slave to atlanta-001. The london-002 node, collocated with london-001, is operating as a synchronous slave to london-001.

Performance Measurement and Results

Test environment

A "2+2" load-balanced test environment was deployed in VMware on a laptop. Device and software specifications were as follows:

1. Host computer
    a. Dell XPS 15 laptop
    b. Processor: Intel Core i7-3632QM CPU @ 2.20GHz
    c. RAM: 16.0 GByte
    d. OS: Windows 10, 64-bit
    e. SSD: 256 GByte SSD, plus 2TByte SSD
2. VMware Workstation 12 Pro version 12.5.7 build-5813279
3. Load balancer
    a. F5 BIG-IP version 12.1.2 Build 0.0.249
    b. VM guest
        i. Memory: 4 GByte
        ii. Processors: 2
        iii. HDD 1: 142 GByte
        iv. HDD 2: 20 GByte
        v. Network Adaptor: 4 x NAT
4. qCrypt-VM x 4
    a. Release: 1.6
    b. VM guest
        i. Memory: 1 GByte
        ii. Processors: 1
        iii. HDD: 40 GByte
        iv. Network Adaptor: NAT (Client, Management)
        v. Network Adaptor: LAN Segment (Replication)

Test configurations

The following test configurations were used:

1. All connections to the same KM server, connections not cached
2. All connections to the same KM server, connections cached
3. Connections load-balanced across two KM servers, connections not cached
4. Connections load-balanced across two KM servers, connections cached

In all cases a "2+2" configuration of qCrypt was used; i.e. a total of four qCrypt-VM nodes, two nodes providing KM service (master nodes), two nodes running as slave-only nodes. Replication enabled from each master node to all other nodes.

Performance tests

The qClient sample program, s_speed, was used to test the performance of the following operations:

1. Get AES-256 key
2. Wrap key using NIST AES key wrap with 256-bit key
3. Create AES-256 key
4. Create RSA-2048 key pair
5. Create ECDSA Curve P-256 key pair

s_speed supports a number of command line options:

```
usage: s_speed <option>
where options are:
-help                  - Display program usage.
-?                     - Display program usage.
@<configuration file>  - Connection command configuration file name.
-repeat <count>        - (Optional) Repeat the operation count times. Default is 100
                         times.Count must be an integer greater than 0.
-no_reuse              - (Optional) Do not re-use the connection; i.e. establish a
                         new TLS session prior to each operation. Default behaviour
                         is to re-use an already established connection.
-server                - Query server vendor and information.
-versions              - Discover protocol versions supported by the server.
-get                   - Get a symmetric key.
-create                - Create symmetric keys.
-create_rsa            - Create RSA key pair.
-create_ecc            - Create ECC key pair.
-wrap                  - Perform wrap operation.
```

Help output of s_speed

In all performance tests, the repeat count value was left at the default value of 100.

Tests were run with the no_reuse switch included, and with it omitted. In the results tables and graphs, this is indicated by "(not cached)", and "(cached)" comments respectively.

Tests were run against a single active qCrypt server, as well as against a pair of active qCrypt servers. For the former case, the client connected directly over TLS to the qCrypt server. For the latter case, qClient connected to the VIP address of the F5 BIG-IP load balancer which was set to round-robin mode.

At all times, four qCrypt-VM nodes were connected in a replication group.

Each test was repeated with a single client instance, and in increments of one, to ten concurrent client instances. Concurrent instances were run on the Dell XPS-15 host machine from a Cygwin bash shell. Examples:

```
./s_speed $PERFORMANCE_CLIENT -create_ecc
```

Example of single instantiation of s_speed test for ECC

```
./s_speed $PERFORMANCE_CLIENT -create_ecc & ./s_speed
$PERFORMANCE_CLIENT -create_ecc & ./s_speed
$PERFORMANCE_CLIENT -create_ecc & ./s_speed $PERFORMANCE_CLIENT
-create_ecc & ./s_speed $PERFORMANCE_CLIENT -create_ecc
```

Example of five concurrent instantiations of s_speed test for ECC

Performance test results

Get AES-256 symmetric key

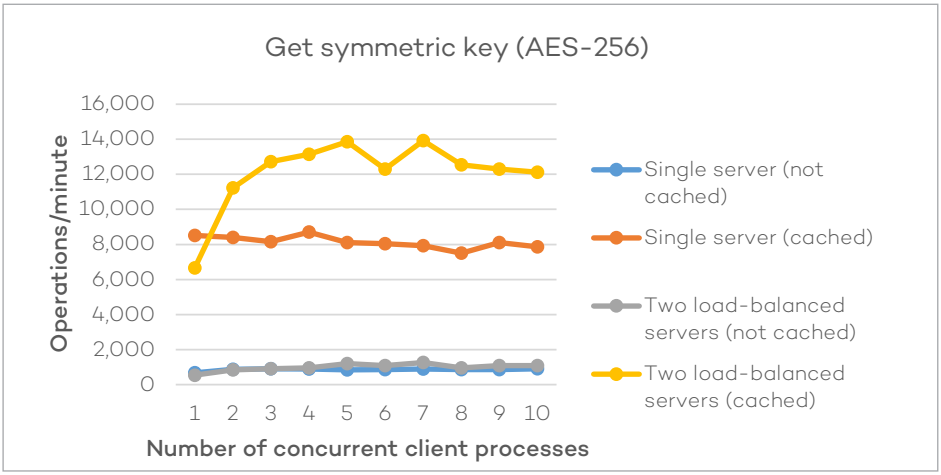
For the get key operation, connection caching improved performance by approximately an order of magnitude; e.g. for five concurrent clients connecting to a single server, performance increased from 840 to 8,100 operations per minute, and for two load-balanced servers performance increased from 1,200 to 13,860 operations per minute.

For two or more concurrent clients using cached connections, performance was approximately 50% higher for two load-balanced servers versus a single server.

For a single client process, performance was higher for the single server case than the two load-balanced server case. This is most likely mostly due to the overheads introduced by the load balancer in the data path.

A final observation is that with two or more concurrent clients, performance remained fairly flat. This indicates that the server(s) are easily able to serve the load presented by the clients. It would be interesting to extend the tests beyond ten concurrent clients to determine when server capacity is reached. For the test configuration used (i.e. a single laptop computer running multiple virtual machines) it is likely that laptop performance limitations would impact results, and therefore the tests were limited to just the ten concurrent clients.

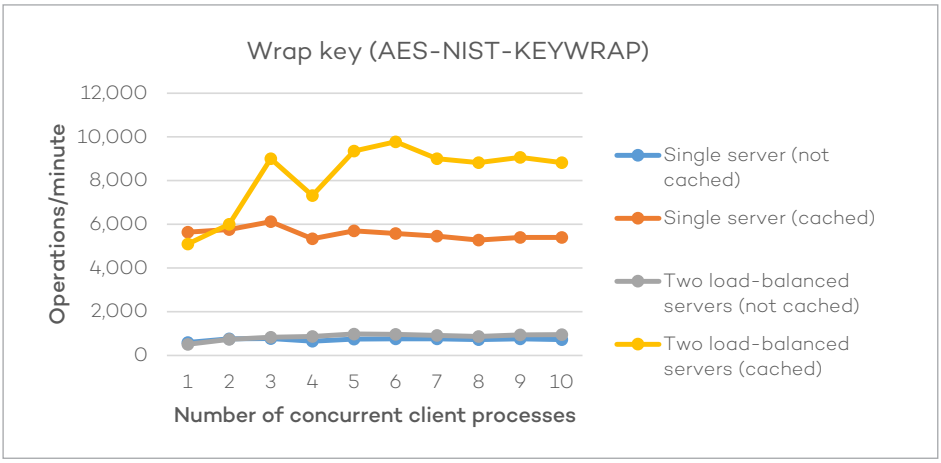
Get symmetric key (AES-256)				
Number of concurrent client processes	Single server		Two loaded-balanced servers	
	(not cached)	(cached)	(not cached)	(cached)
1	672	8,520	540	6,660
2	870	8,400	840	11,220
3	900	8,160	900	12,720
4	888	8,700	960	13,140
5	840	8,100	1,200	13,860
6	864	8,040	1,080	12,300
7	882	7,920	1,260	13,920
8	864	7,500	960	12,540
9	864	8,100	1,080	12,300
10	900	7,860	1,080	12,120



NIST AES key wrap

For the NIST AES key wrap operation, relative performance was very similar to get key; i.e. cached connections improved performance by approximately an order of magnitude, two load-balanced servers provided approximately 50% greater transaction rate than a single server, and beyond two concurrent clients, performance remained flat.

Wrap key (AES-NIST-KEYWRAP)				
Number of concurrent client processes	Single server		Two loaded-balanced servers	
	(not cached)	(cached)	(not cached)	(cached)
1	594	5,640	510	5,100
2	756	5,760	732	6,000
3	774	6,120	828	9,000
4	648	5,340	864	7,320
5	750	5,700	984	9,360
6	756	5,580	966	9,780
7	756	5,460	917	9,000
8	720	5,280	864	8,820
9	756	5,400	942	9,060
10	720	5,400	960	8,820

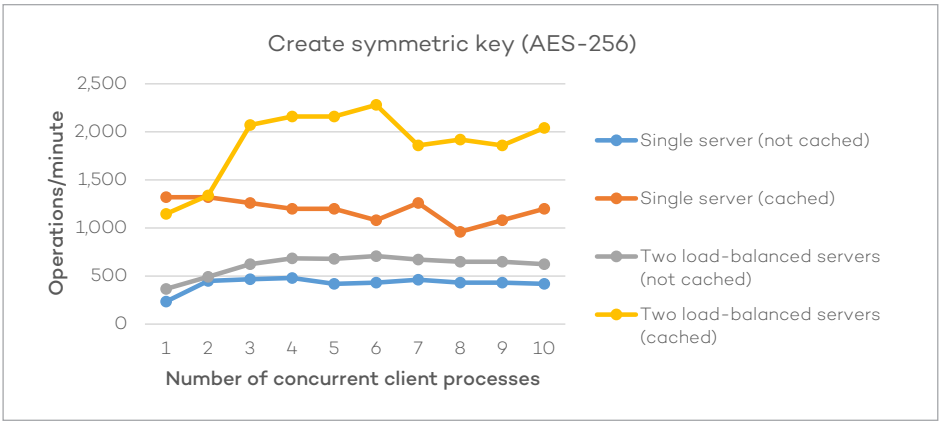


Create AES-256 key

For the create AES-256 key operation, lower relative performance between cached and non-cached cases was observed. This is due to the fact that the time taken for the create key operation is more significant relative to TLS connection setup time. That is, the relative contribution to total transaction time by TLS setup is reduced. Even so, cached configurations provided significantly higher transaction rates.

Using two load-balanced servers combined with caching of connections increased performance over a single connection-cached server by approximately a factor of two.

Create symmetric key (AES-256)				
Number of concurrent client processes	Single server		Two loaded-balanced servers	
	(not cached)	(cached)	(not cached)	(cached)
1	234	1,320	366	1,146
2	450	1,320	492	1,338
3	468	1,260	624	2,070
4	480	1,200	684	2,160
5	420	1,200	678	2,160
6	432	1,080	708	2,280
7	462	1,260	672	1,860
8	432	960	648	1,920
9	432	1,080	648	1,860
10	420	1,200	624	2,040

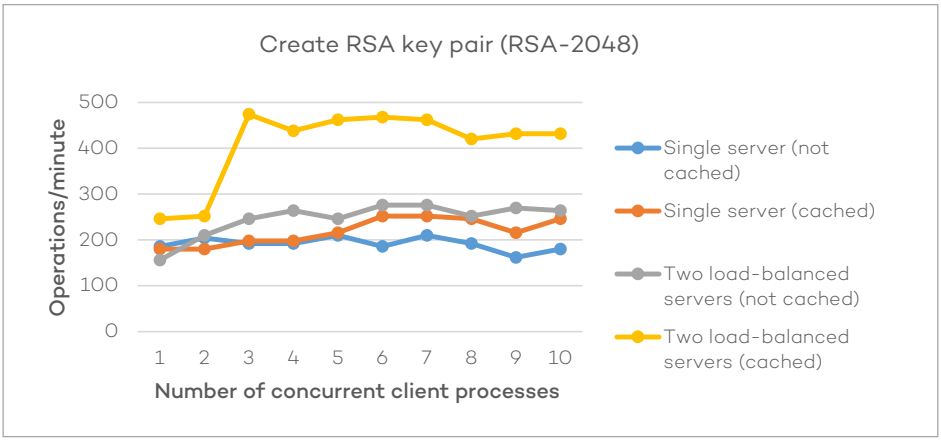


Create RSA-2048 key pair

For the create RSA-2048 key pair operation, connection caching provided only marginal performance improvement. This is because key pair creation time dominates the transaction time.

Significant performance increases were seen when connection caching was combined with two load-balanced servers.

Create RSA key pair (RSA-2048)				
Number of concurrent client processes	Single server		Two loaded-balanced servers	
	(not cached)	(cached)	(not cached)	(cached)
1	186	180	156	246
2	204	180	210	252
3	192	198	246	474
4	192	198	264	438
5	210	216	246	462
6	186	252	276	468
7	210	252	276	462
8	192	246	252	420
9	162	216	270	432
10	180	246	264	432

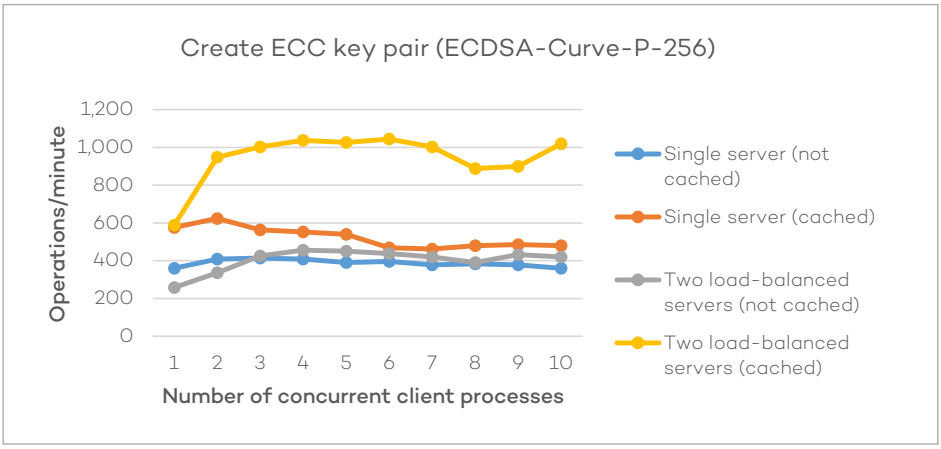


Create ECDSA-Curve-P-256 key pair

For the create ECDSA key pair operation connection caching marginally improves performance. As for the create RSA-2048 key pair operation, this is because ECDSA key pair creation time dominates the transaction time.

Significant performance increases were seen when connection caching was combined with two load-balanced servers.

Create ECC key pair (ECDSA-Curve-P-256)				
Number of concurrent client processes	Single server		Two loaded-balanced servers	
	(not cached)	(cached)	(not cached)	(cached)
1	360	576	258	588
2	408	624	336	948
3	414	564	426	1002
4	408	552	456	1,038
5	390	540	450	1,026
6	396	468	438	1,044
7	378	462	420	1,002
8	384	480	390	888
9	378	486	432	900
10	360	480	420	1,020



Conclusions

The results show that for all operations tested, connection caching, and load-balancing two servers improved performance. Most significant performance improvements, when enabling connection caching, were seen when the operation time was relatively short compared to the TLS session establishment time.

Sharing load between two servers always improved performance by up to a factor of two.

The relative performance numbers published in this application provide useful information for architectural design and deployment; e.g. connection caching in clients reduces total transaction time, and load balancing of multiple servers increases aggregate throughput.

The actual performance numbers published in the application note were generated from a system implemented using virtual machines on a single Windows laptop, and therefore are not representative of a typical production environment. However, the performance numbers do provide indicative relative performance that should be representative of production systems, and therefore, the conclusions drawn in this document should also be applicable to those systems.

Replication was enabled at all times, so the relative impact of replication traffic, and synchronous versus asynchronous replication was not tested. Even so, it is logical to expect that latency on synchronous replication links would impact performance.

To maximise KMIP operation performance, the following recommendations are made:

1. Enable connection caching in the client;
2. Deploy multiple servers behind a load balancer; and
3. Configure replication node priorities so that a slave node collocated with its master node has highest priority; i.e. synchronous replication traffic flows over the lowest latency, highest bandwidth link, and other replication traffic flows asynchronously.

About QuintessenceLabs

QuintessenceLabs' portfolio of modular products addresses the most difficult security challenges, helping implement robust security strategies to protect data today and in the future. For more information on QuintessenceLabs' data protection solutions, please visit **www.quintessencelabs.com**

Quintessence Labs

AUSTRALIA
Unit 1, Lower Ground
15 Denison St
Deakin, ACT 2600
+61 2 6260 4922

UNITED STATES
175 Bernal Road
Suite 220
San Jose CA 95119
+1 650 870 9920

www.quintessencelabs.com

Document ID: 3236
AN-2017239-001