Implementing reverse proxy mechanism (load balancer - HTTP) alongside data encryption (TLS/SSL)

Team-2 Members:
Srinivas Baskar
Nithin Veer Reddy
Sitesh Ranjan
Chanheum Park

Project Motivation

- Load Balancer is a vital part of a client server communication.
- It helps regulate requests by distributing them efficiently between multiple servers based on certain conditions.
- In this project we implement a load balancer that can easily be deployed on any platform with just a config file update.
- Besides load balancing, our system also encrypts data for security.
- Encryption is an important aspect of data sharing.
- There are many ways of encrypting the data - key sharing (Asymmetric, symmetric encryption)

Related Work

- Peter Sommerad, Reverse Proxy Patterns: This paper talks about the 3 patterns (Protection Reverse Proxy Pattern, Integration Reverse Proxy pattern, Front Door pattern) to understand reverse proxy solutions and applied to design reverse proxy architectures.
- Jiang Du and GuoXin Nie: Design and implementation of Security Reverse Data Proxy server based on SSL.
- Zahra Mohammed Elngomi, Khalid Khanfar, A Comparative Study of Load Balancing Algorithms: This paper describes the different types of load balancing algorithms explaining their categories and classification. It takes into account two load balancing approaches static and dynamic.

Our techniques

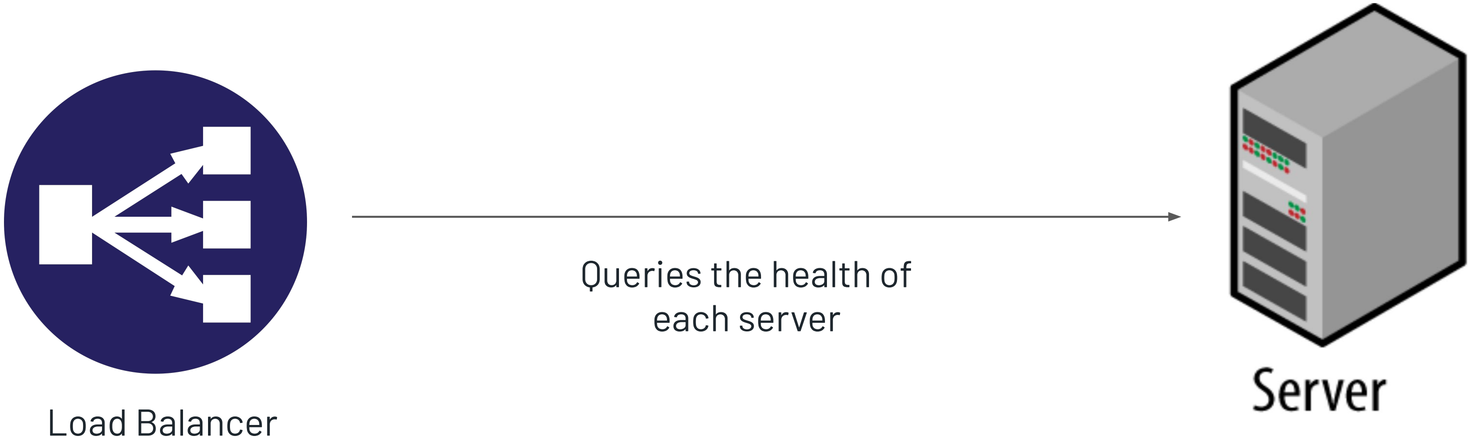
- We deploy a lightweight load balancer that can be used cross platform and is very easy to deploy.
- The load balancer manages the servers by querying them to check their health and status.
- Besides this, the load balancer also compresses requests.
- We used Asymmetric and symmetric key encryption technique to solve the problem.
- Keys sharing is always a chicken & egg problem.
- We used asymmetric keys to share the symmetric keys.

Reverse Proxy
(Load Balancer)

Load Balancer

- Load balancing is the methodical and efficient distribution of network or application traffic across multiple servers in a server farm.
- We designed our load balancer to be lightweight and easy to deploy.
- To achieve this, we only need to input a config file that contains the IP and port of all the servers in the server farm.

Load Balancer



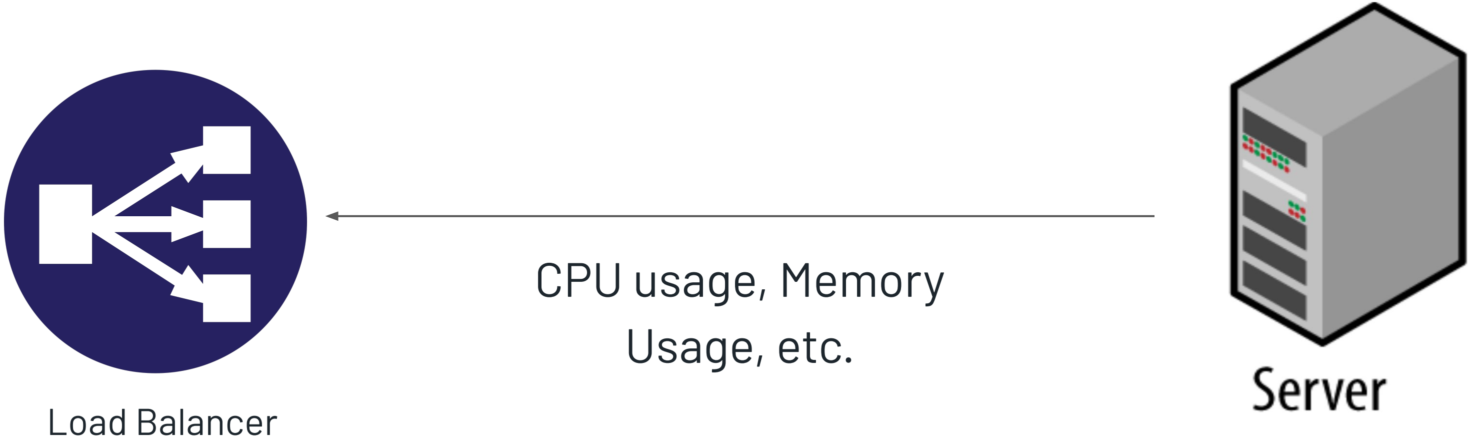
Load Balancer

Queries the health of each server

Server

Load Balancer

- To distribute the incoming requests, it uses a combination of the server's status, including available memory, cpu, etc. to efficiently find a server to process this request.



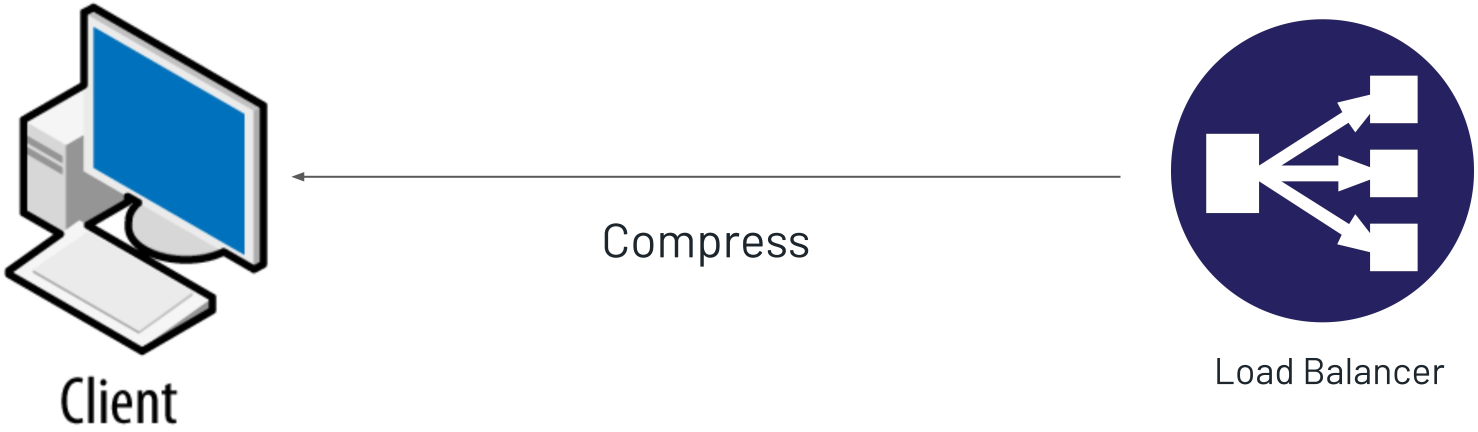
Load Balancer

CPU usage, Memory Usage, etc.

Server

Load Balancer

- It also compresses the outgoing data so that less data is transmitted, thereby reducing the load on the network.



Client

Compress

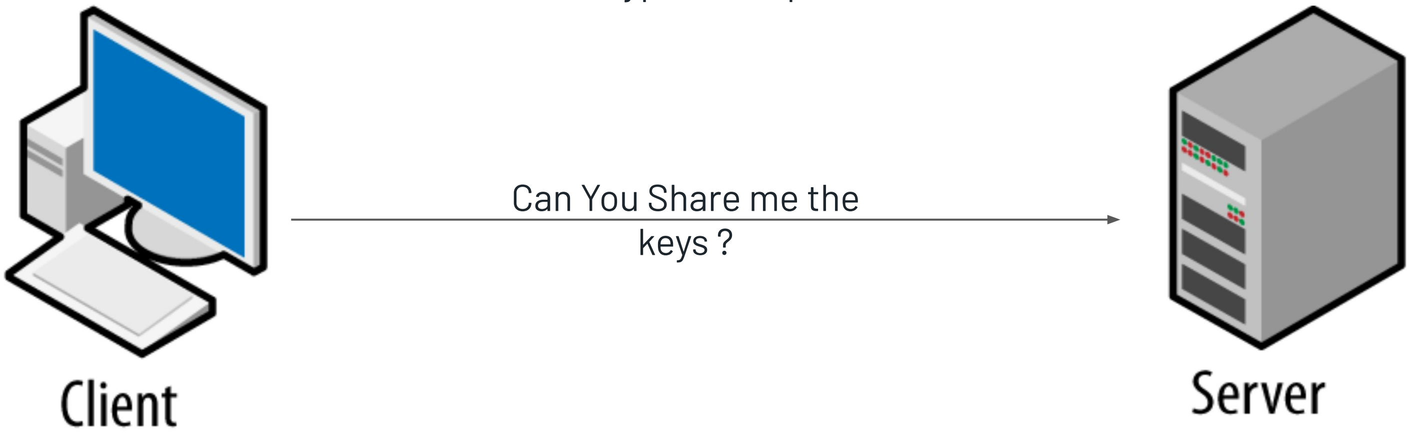
Load Balancer

Encryption

Encryption

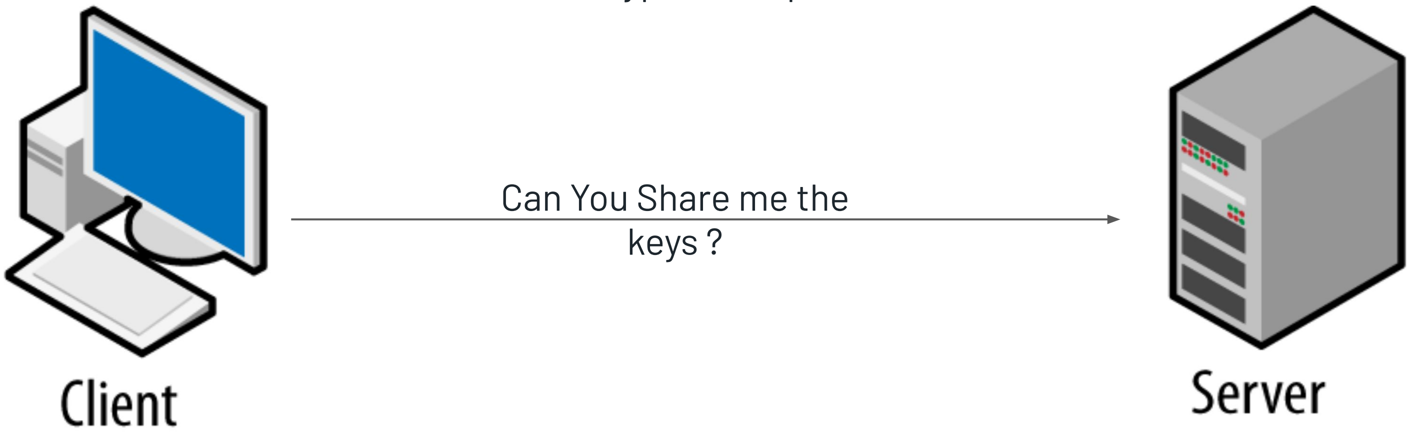
- Encryption is an efficient way to convert an understandable message into a form of code to prevent an unauthorized access.
- All the data exchanged between the server and the client are encrypted.
- To achieve this, we used both the techniques of asymmetric and symmetric key mechanisms.

Encryption Steps



Can You Share me the keys ?

Encryption Steps

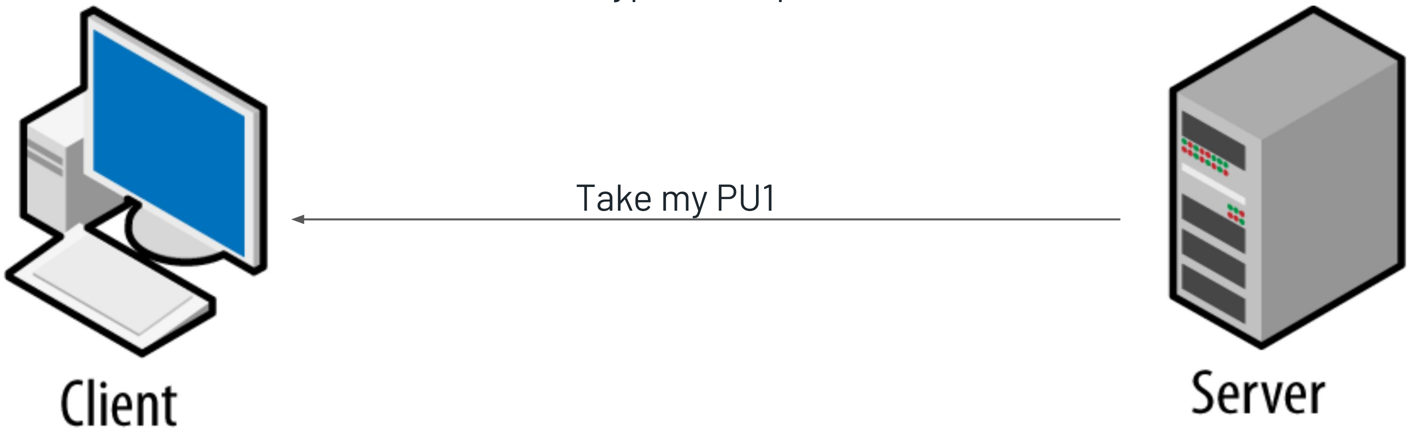


Can You Share me the keys ?

I'll Create asymmetric key combinations

- PU1
- PR1

Encryption Steps

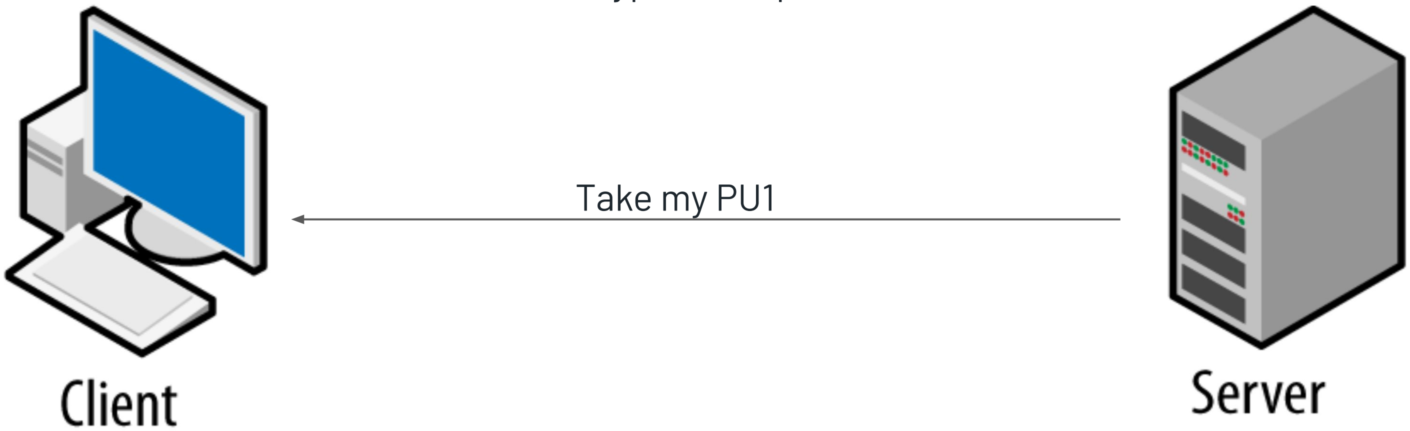


Take my PU1

I'll Create asymmetric key combinations

- PU1
- PR1

Encryption Steps



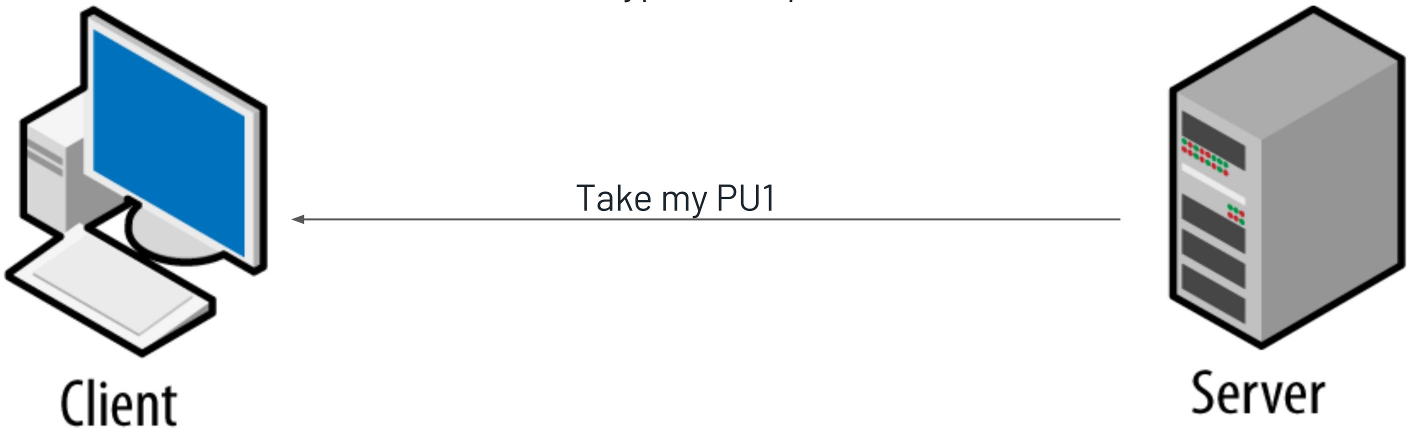
Take my PU1

I’ll store server public
key PU1

I’ll Create asymmetric
key combinations

- PU1
- PR1

Encryption Steps



Take my PU1

Client

Server

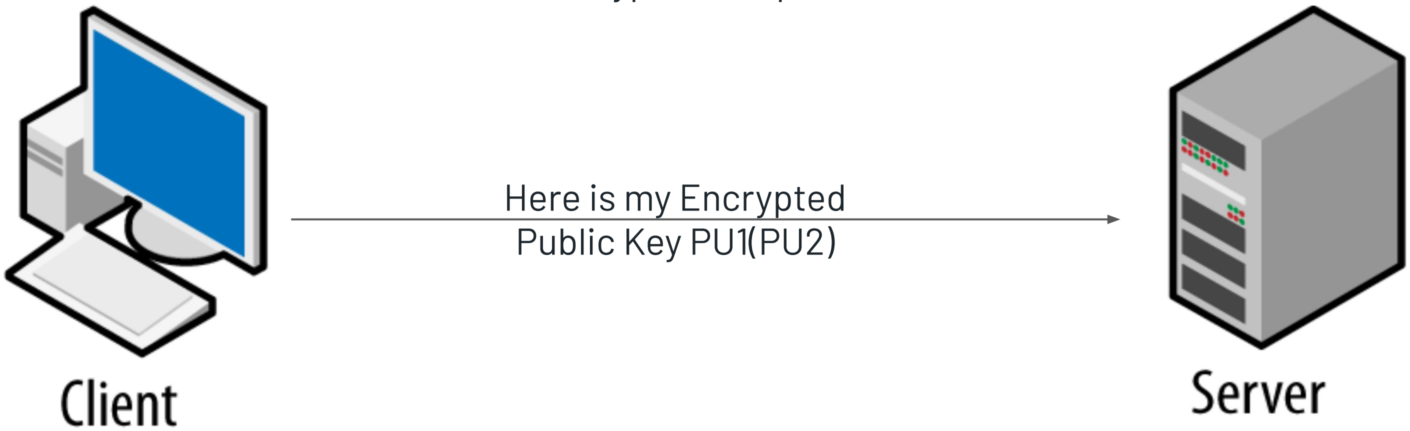
I’ll store server public key PU1.
I’ll also create asymmetric key combinations

- PU2
- PR2

I’ll Create asymmetric key combinations

- PU1
- PR1

Encryption Steps



Here is my Encrypted Public Key PU1(PU2)

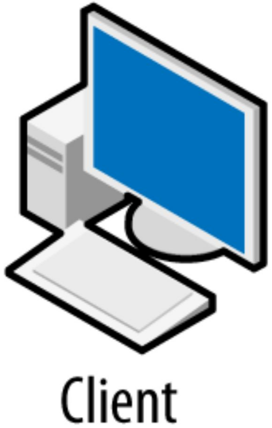
I’ll store server public key PU1.
I’ll also create asymmetric key combinations

- PU2
- PR2

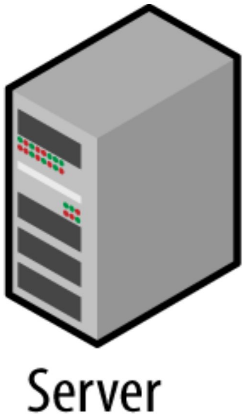
I’ll Create asymmetric key combinations

- PU1
- PR1

Encryption Steps



Here is my Encrypted Public Key PU1(PU2)



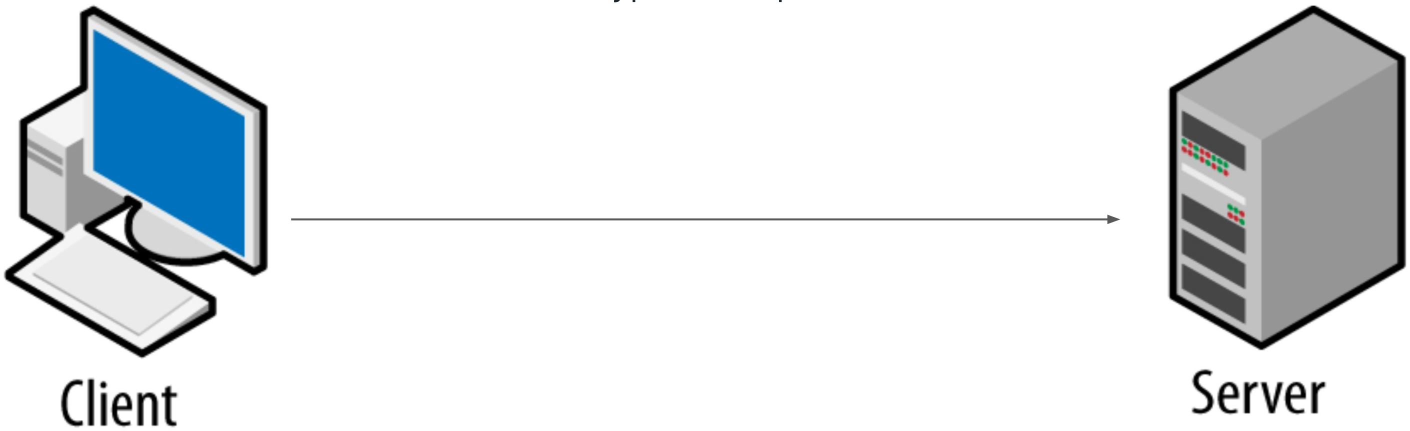
I’ll store server public key PU1.
I’ll also create asymmetric key combinations

- PU2
- PR2

I’ll decrypt using my PR1
I Now have

- PU1
- PR1
- PU2

Encryption Steps



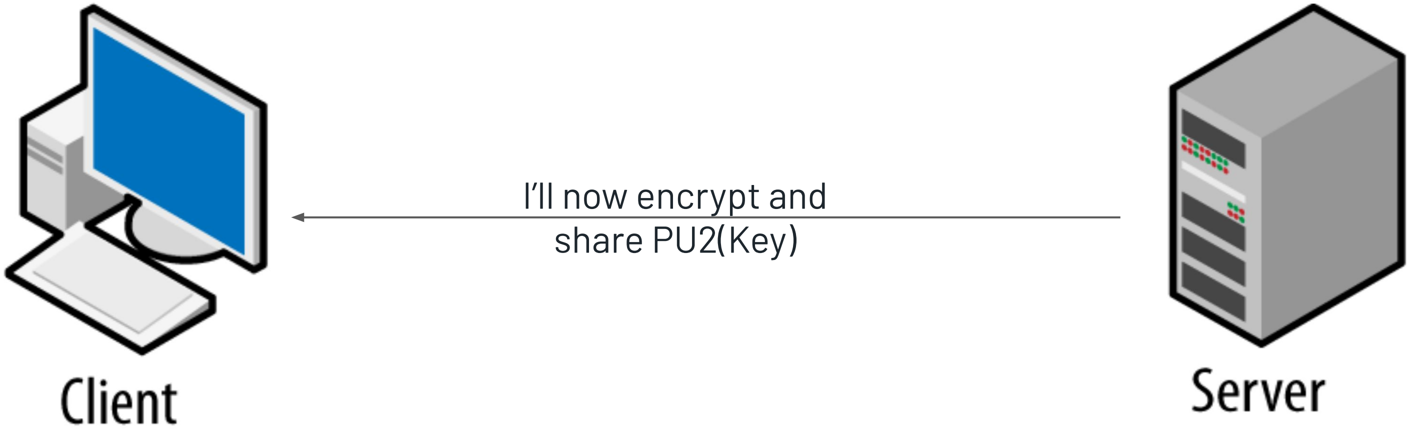
I’ll have

- PU2
- PR2
- PU1

I’ll create symmetric key
I Now have

- PU1
- PR1
- PU2
- Key

Encryption Steps



I'll now encrypt and share PU2(Key)

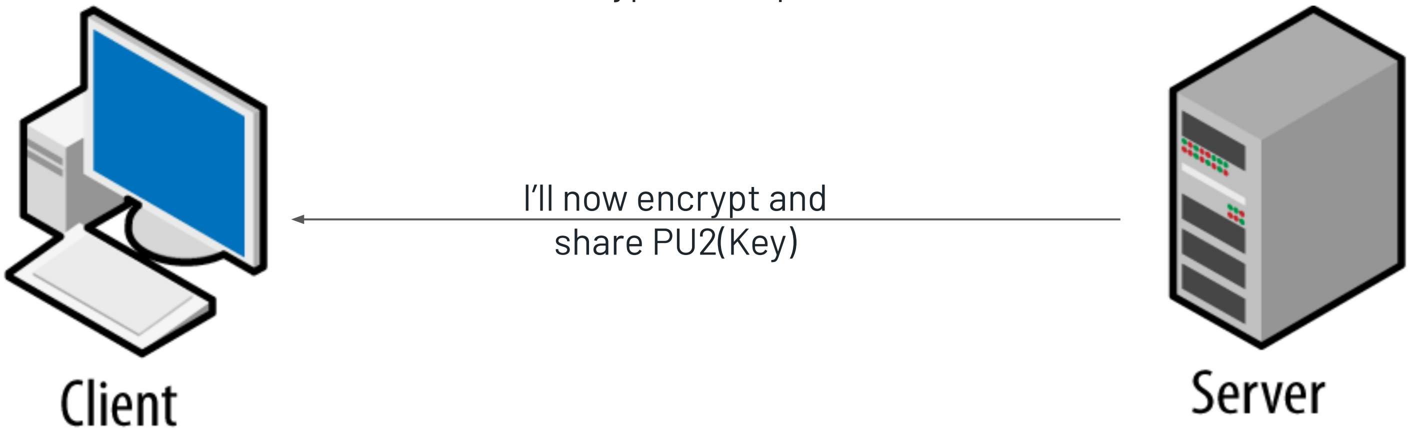
I'll have

- PU2
- PR2
- PU1

I'll create symmetric key
I Now have

- PU1
- PR1
- PU2
- Key

Encryption Steps



I’ll now encrypt and share PU2(Key)

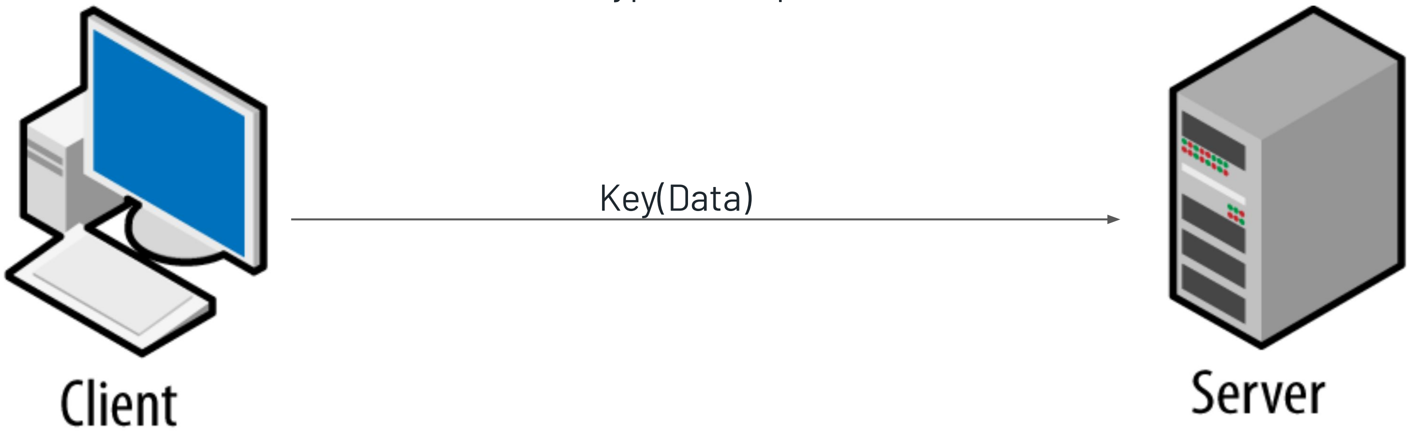
I’ll decrypt using PR2
I’ll now have

- PU2
- PR2
- PU1
- Key

I’ll create symmetric key
I Now have

- PU1
- PR1
- PU2
- Key

Encryption Steps



Key(Data)

I have

- PU2
- PR2
- PU1
- Key

I have

- PU1
- PR1
- PU2
- Key

Encryption Key Storage

- The keys on the server side is stored in a common location so that any server that receives the request can access these keys and process the request.
- To implement this, we use a redis store running on another server solely for this purpose.
- Each server can request the keys from this store based on the client id and decrypt the message and process the request.

Demonstration

Evaluation

Load Balancer Evaluation

- To test the efficiency of the load balancer, we ran 4 servers on different GCP instances.
- Then we simulate high load using stress tool on ubuntu.
  - “stress —cpu n” : runs n threads on the cpu to occupy it.
- We then look at how the requests are being routed.
- In each of the situations, the requests were routed to the servers with least load, thereby receiving a response in a short time.

Load Balancer Evaluation

- When a server is under heavy load, if a request is directed to this server using a naive load balancing algorithm, the request takes 10.73s to finish (each request here includes generating the keys for encryption from scratch).
- Our load balancer takes only 6.86s to finish (given there is at least one server with low load).
- Using the compression algorithm, a dummy response that takes up 1275 bytes is transmitted in only 229 bytes without any loss.
- Besides this, we ran tests to ensure requests are being directed to a server that is down.

Encryption Evaluation

- Client - Server key exchange which includes both symmetric and asymmetric methods using single client request.
- All type of data - files, plain text, pictures etc can be shared by this approach.
- We ran tests to check the correctness and data integrity of the pipeline.
- We used Flask framework to build above application.

Conclusion

- Successfully implemented and deployed the load-balancer that can be used on cross-platform easily.
- Provided end-to-end encryption of data using symmetric and asymmetric encryption.
- Data compression functionality is provided for incoming data and outgoing data to enhance network performance.
- Various functionality and performance tests are performed using GCP to demonstrate the working of load-balancer & data encryption.

Future work

- Currently our load balancer does not support complete caching due to the limited amount of time we had to implement this.
- Implementing a firewall on the load balancer.
- Using multiple load balancers to avoid any bottlenecks - resilience.
- Auto Scaling Mechanism based on CPU, Memory, Bandwidth availability etc.

Retrospective

what went well?

- Setting up of complete pipeline which includes Load-balancer and Encryption was successful.

what was confusing?

- Deploying the load-balancer on the GCP was little confusing as it took some time for the getting the GCP account to set up.

what didn’t go well?

- End to end Complete caching mechanism isn’t completed due to limited time frame.
- Missed in-person team work.

References

- https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/
- https://testdriven.io/courses/http-load-balancer/concepts/
- Peter Sommerad, Reverse Proxy Patterns
- Zeng Zeng and Bharadwaj Veeravalli. Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems, Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04) 1521-9097/04
- Zahra Mohammed Elngomi, Khalid Khanfar. A Comparative Study of Load Balancing Algorithms: A Review Paper, IJCSMC, Vol. 5, Issue. 6, June 2016, pg.448–458
- Jiang Du, GuoXin Nie. Design and Implementation of Security Reverse Data Proxy Server Based on SSL, ICCIC 2011: Information and Management Engineering pp 523-528
- Source Code - https://github.com/nithinveer/NetworkSystemsProject

Questions ?