Universidade do Minho
Escola de Engenharia

Master's in Telecommunication and Informatics Engineering

Network Management

Project Report

Camila Pinto - PG53712

Barbara Fonseca - PG53677

Eduarda - PG53793

Contents

1 Introduction

The following report is part of the Network Management Course project, covering crucial aspects of the project development. It discusses the adopted strategies, addressing both the overall architecture and specific implemented modules, along with the mechanisms used in each one of them. The section dedicated to the MIB details the system's interaction with this component, exploring potential use cases and their impacts on MIB fields. The critical analysis encompasses functional tests, providing an in-depth view of SNMPkeys' performance, with a critical discussion of results highlighting strengths and challenges faced. To conclude the report, a summary is presented synthesizing key findings, offering a comprehensive view of the project.

The project focuses on creating an SNMPkeys system, a simplified solution for key management in distributed environments. Following a similar structure to SNMP, SNMPkeys utilizes an agent maintaining a MIB. Its functionalities include configuration reading, dynamic key generation, and responding to requests for key generation and verification. It is important to note that the system foregoes security mechanisms, focusing on initial configuration. Communication between the manager and agent occurs through a protocol analogous to SNMP, encapsulated with UDP.

2 Adopted Strategies

The execution of this project involved the usage of various strategies to ensure the proper development of the SNMPkeys system. In Python, a Matrix class was developed for key generation, an agent communicating with managers through the UDP protocol, and an MIB for storing relevant information.

2.1 Architecture

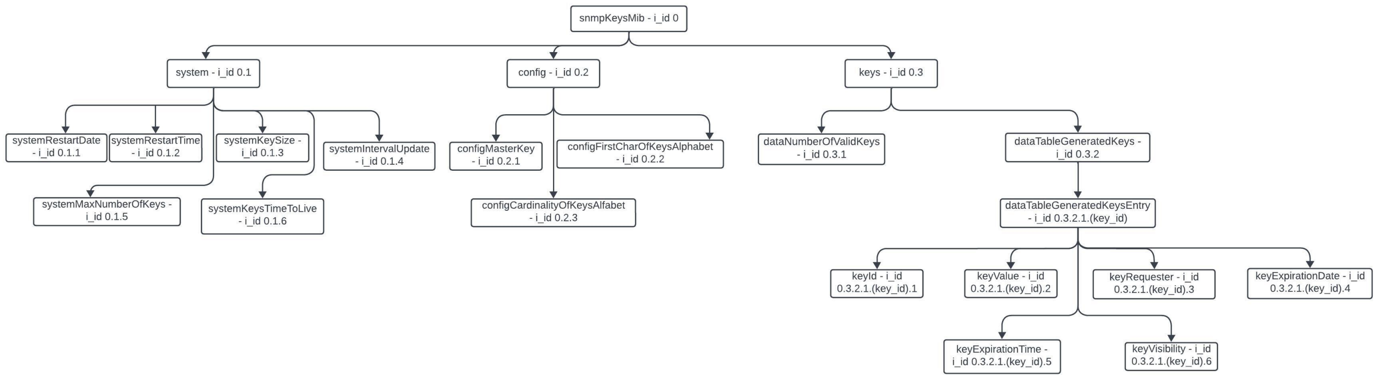
Initially, the focus was on a detailed analysis of requirements and the definition of crucial parameters such as K, M, T, V, and X during the initial agent setup and dynamic creation of the Z key matrix. A MatrixZ class was developed using Python, representing a matrix used for cryptographic key generation. This matrix is initialized based on parameters like m and k, undergoing rotation and transposition processes. Periodic matrix updates occur through the UpdateMatrix method, which performs rotations on rows and columns, and key generation is based on the updated matrix in the GenerateKey method. The use of locks ensures security in multithreaded environments, given the concurrent nature of matrix updates. The class is part of a larger system aimed at secure cryptographic key generation in a distributed context.

Subsequently, development focused on implementing the SNMPkeys agent responsible for generating and providing cryptographic keys to SNMP managers. Key functionalities include dynamic key generation, responding to SNMP manager requests, and managing the MIB table to store relevant information about generated keys and other configuration parameters. The agent operates in a distributed environment and uses an architecture similar to SNMP, including primitives such as GET and SET, enabling communication with managers through the UDP protocol.

For the manager's creation, a Python script was implemented, processing command line arguments. The client constructs a request containing information such as identifiers, the quantity of pairs, and the pairs themselves. Before sending the request, a check is performed to ensure its validity, avoiding duplicate identifiers in short intervals. After sending, the manager awaits the agent's response, which is then presented. Special attention is given to controlling instance identifiers 'P' to ensure the integrity and security of transmissions. The implementation uses locks to protect critical operations involving the reading and writing of shared data, such as obtaining and updating 'last P' values.

Finally, an MIB class was implemented to manage relevant information, such as cryptographic keys and configuration parameters, in the context of the SNMP protocol. The MIB will be detailed in the next section.

3 MIB



snmpKeysMib - i_id 0
system - i_id 0.1
config - i_id 0.2
keys - i_id 0.3
systemRestartDate - i_id 0.1.1
systemRestartTime - i_id 0.1.2
systemKeySize - i_id 0.1.3
systemIntervalUpdate - i_id 0.1.4
systemMaxNumberOfKeys - i_id 0.1.5
systemKeysTimeToLive - i_id 0.1.6
configMasterKey - i_id 0.2.1
configFirstCharOfKeysAlphabet - i_id 0.2.2
configCardinalityOfKeysAlfabet - i_id 0.2.3
dataNumberOfValidKeys - i_id 0.3.1
dataTableGeneratedKeys - i_id 0.3.2
dataTableGeneratedKeysEntry - i_id 0.3.2.1.(key_id)
keyId - i_id 0.3.2.1.(key_id).1
keyValue - i_id 0.3.2.1.(key_id).2
keyRequester - i_id 0.3.2.1.(key_id).3
keyExpirationDate - i_id 0.3.2.1.(key_id).4
keyExpirationTime - i_id 0.3.2.1.(key_id).5
keyVisibility - i_id 0.3.2.1.(key_id).6

Figure 1: MIB Diagram

The Management Information Base (MIB) of the system is organized into three distinct groups: system, config, and keys, each playing a specific role in the operation of the SNMP agent.

The system group focuses on providing information about the system's state, while the config group manages settings related to key creation and manipulation. The keys group handles the generated keys and stores them in the snmpKeysMib.

During the agent's initialization, the system and config groups are added to the snmpKeysMib to establish the initial state and configure essential parameters. In the keys group, the dataNumberOfValidKeys field is initialized with the value 0, indicating the absence of keys in the snmpKeysMib.

When the manager requests the creation of a key, the agent generates the key and stores all associated parameters in the snmpKeysMib. Each key has a limited lifespan, after which it is removed from the snmpKeysMib. During key addition and removal operations, the dataNumberOfValidKeys field is adjusted, incremented or decremented depending on the action.

The manager also has the ability to request the viewing of a specific field, identified by its Instance Identifier (I-ID), and its subsequent instances in lexicographical order. If the number of instances to follow is greater than the number of fields in the snmpKeysMib, an error message is sent to indicate the inability to fulfill the manager's request. This organized structure of the MIB and its operations reflects the efficiency and logic of the implemented management system.

4 Critical Analysis

A critical analysis of the SNMPkeys system is essential to evaluate the proposed solutions concerning the established objectives. This system was designed with the purpose of providing an infrastructure for key creation and sharing in distributed management environments. The following discusses the main results and considerations obtained during the analysis.

4.1 Set Primitive

For the set primitive, an UDP connection is established, and a thread is created for each client. Subsequently, the key is generated based on the Z matrix, where the key identifier is the number of times the Z matrix has been updated up to that point.

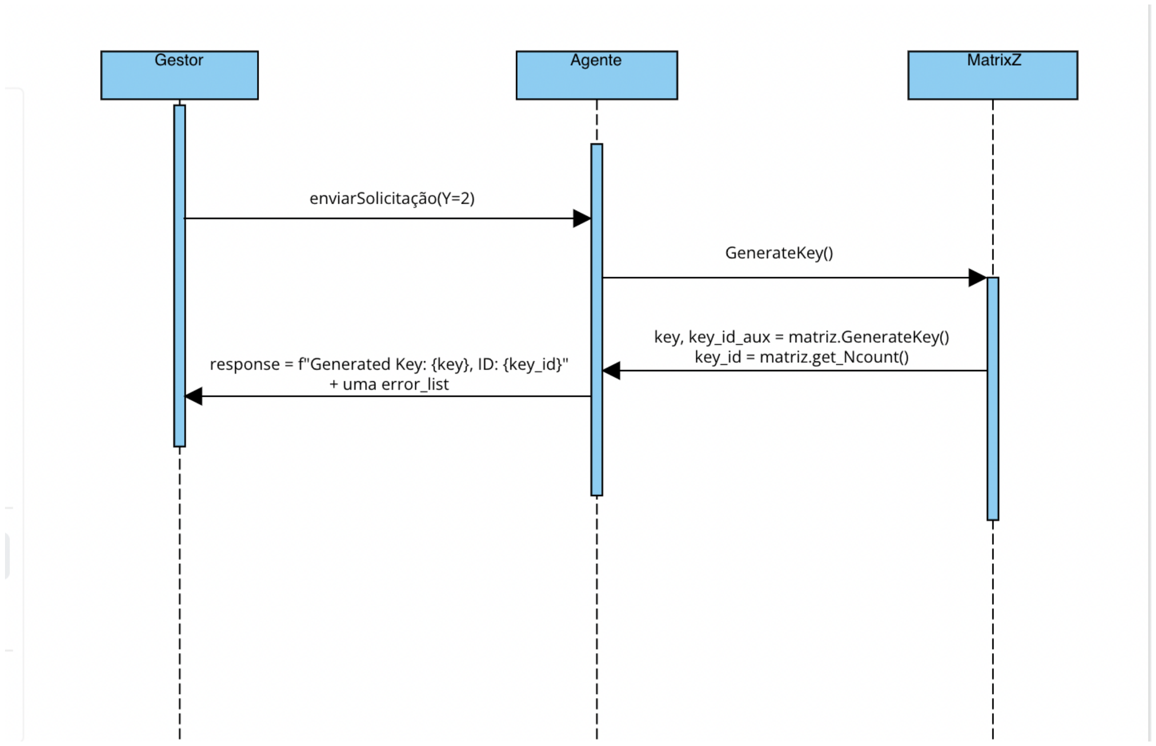
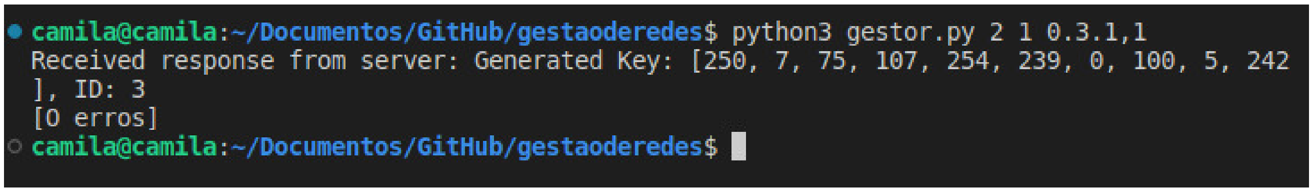

Gestor
Agente
MatrixZ
enviarSolicitação(Y=2)
GenerateKey()
key, key_id_aux = matriz.GenerateKey()
key_id = matriz.get_Ncount()
response = f"Generated Key: {key}, ID: {key_id}"
+ uma error_list

Figure 2: Sequence Diagram Set Primitive

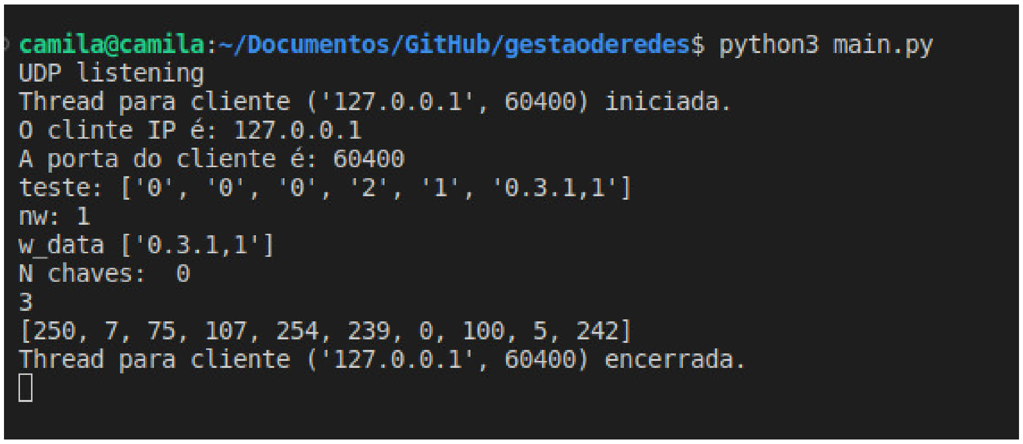
As depicted in Figure 3, the manager initiates the set request using the identifier 2. The subsequent parameter is an integer indicating the quantity of pairs in a list W, followed by the list itself.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 gestor.py 2 1 0.3.1,1
Received response from server: Generated Key: [250, 7, 75, 107, 254, 239, 0, 100, 5, 242
], ID: 3
[0 erros]
camila@camila:~/Documentos/GitHub/gestaoderedes$ 
```

Figure 3: Set Request

In Figure 4, the agent checks the manager's address and port. It prints the w_data, which represents the manager's data, to verify the received information's correctness. Afterward, it verifies in the MIB whether the key limit has been exceeded; if not, it returns 0. Finally, it sends the generated key, stores it in the MIB, and terminates the thread.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 main.py
UDP listening
Thread para cliente ('127.0.0.1', 60400) iniciada.
O clinte IP é: 127.0.0.1
A porta do cliente é: 60400
teste: ['0', '0', '0', '2', '1', '0.3.1,1']
nw: 1
w_data ['0.3.1,1']
N chaves:  0
3
[250, 7, 75, 107, 254, 239, 0, 100, 5, 242]
Thread para cliente ('127.0.0.1', 60400) encerrada.
```

Figure 4: Set Request Execution

4.2 Get Primitive

For the get primitive, an UDP connection is established, and a thread is created for each client. The agent receives the manager's request, looks up the information in the MIB, and sends the response.

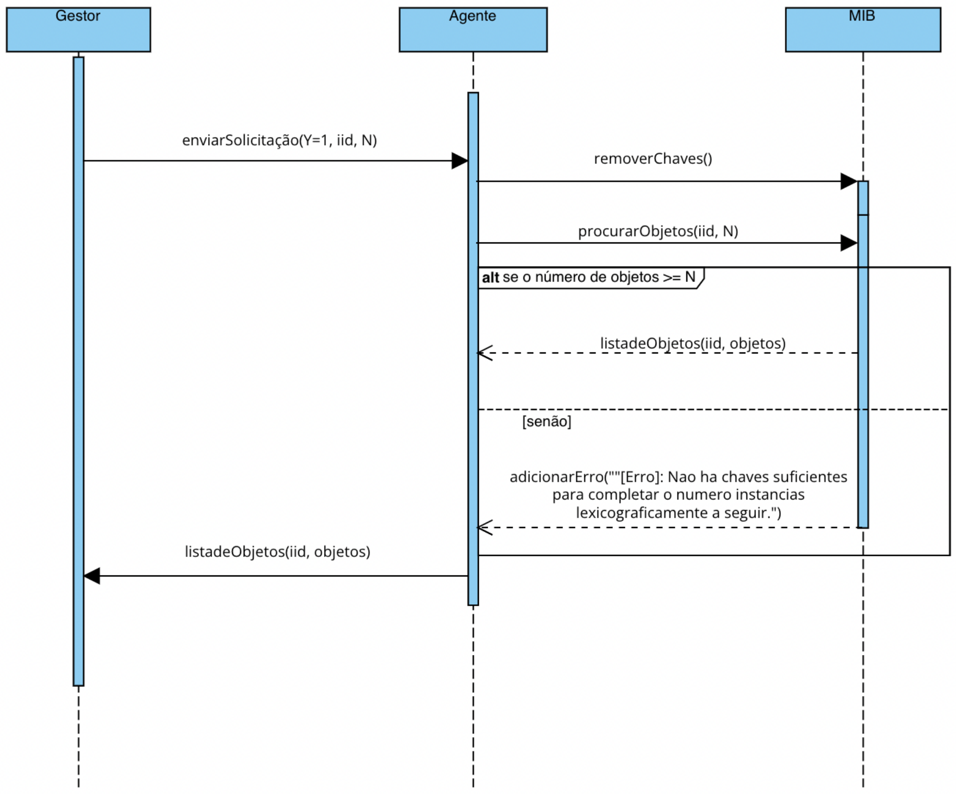
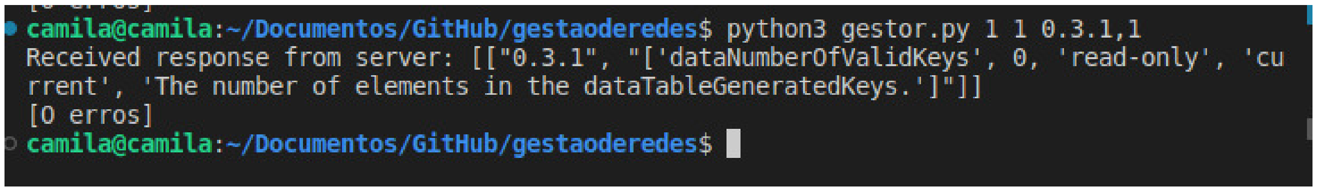

Gestor
Agente
MIB
enviarSolicitação(Y=1, iid, N)
removerChaves()
procurarObjetos(iid, N)
alt se o número de objetos >= N
listadeObjetos(iid, objetos)
[senão]
adicionarErro(""[Erro]: Nao ha chaves suficientes para completar o numero instancias lexicograficamente a seguir.")
listadeObjetos(iid, objetos)

Figure 5: Sequence Diagram Get Primitive

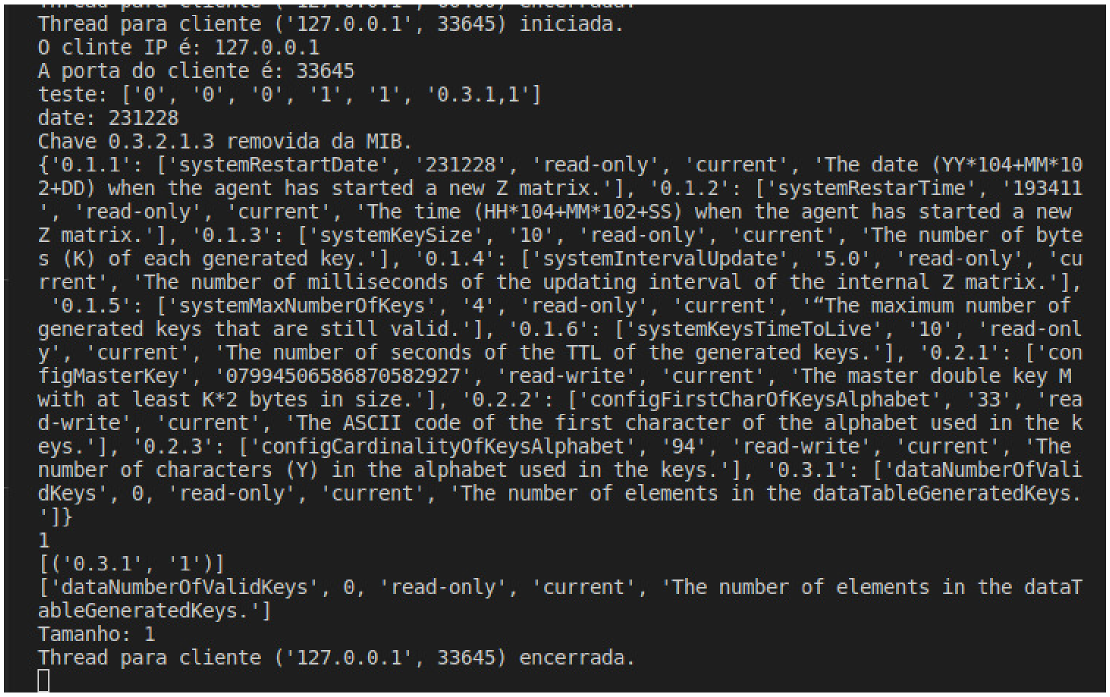
As illustrated in Figure 6, the manager initiates the get request using the identifier 1. The subsequent parameter is an integer indicating the quantity of pairs in a list L, followed by the list itself, which contains the I-ID and instances lexicographically following the instance referenced by I-ID.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 gestor.py 1 1 0.3.1,1
Received response from server: [["0.3.1", "['dataNumberOfValidKeys', 0, 'read-only', 'cu
rrent', 'The number of elements in the dataTableGeneratedKeys.']"]]
[0 erros]
camila@camila:~/Documentos/GitHub/gestaoderedes$ 
```

Figure 6: Get Request

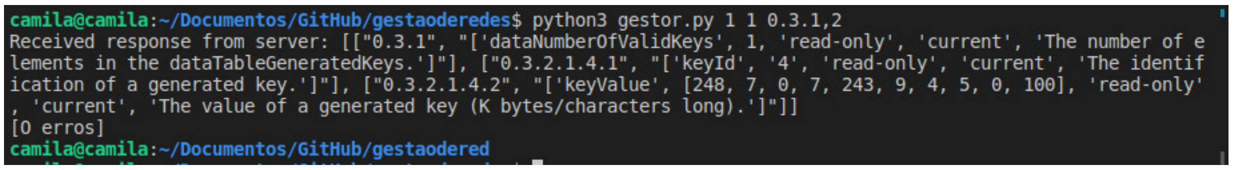
In Figure 7, the agent searches the MIB to determine if the I-ID requested by the manager exists. If it does, the agent will send the requested instance to the manager.



```
Thread para cliente ('127.0.0.1', 33645) iniciada.
O clinte IP é: 127.0.0.1
A porta do cliente é: 33645
teste: ['0', '0', '0', '1', '1', '0.3.1,1']
date: 231228
Chave 0.3.2.1.3 removida da MIB.
{'0.1.1': ['systemRestartDate', '231228', 'read-only', 'current', 'The date (YY*104+MM*10
2+DD) when the agent has started a new Z matrix.'], '0.1.2': ['systemRestarTime', '193411
', 'read-only', 'current', 'The time (HH*104+MM*102+SS) when the agent has started a new
Z matrix.'], '0.1.3': ['systemKeySize', '10', 'read-only', 'current', 'The number of byte
s (K) of each generated key.'], '0.1.4': ['systemIntervalUpdate', '5.0', 'read-only', 'cu
rrent', 'The number of milliseconds of the updating interval of the internal Z matrix.'],
 '0.1.5': ['systemMaxNumberOfKeys', '4', 'read-only', 'current', '“The maximum number of
generated keys that are still valid.'], '0.1.6': ['systemKeysTimeToLive', '10', 'read-onl
y', 'current', 'The number of seconds of the TTL of the generated keys.'], '0.2.1': ['con
figMasterKey', '07994506586870582927', 'read-write', 'current', 'The master double key M
with at least K*2 bytes in size.'], '0.2.2': ['configFirstCharOfKeysAlphabet', '33', 'rea
d-write', 'current', 'The ASCII code of the first character of the alphabet used in the k
eys.'], '0.2.3': ['configCardinalityOfKeysAlphabet', '94', 'read-write', 'current', 'The
number of characters (Y) in the alphabet used in the keys.'], '0.3.1': ['dataNumberOfVali
dKeys', 0, 'read-only', 'current', 'The number of elements in the dataTableGeneratedKeys.
']}
1
[('0.3.1', '1')]
['dataNumberOfValidKeys', 0, 'read-only', 'current', 'The number of elements in the dataT
ableGeneratedKeys.']
Tamanho: 1
Thread para cliente ('127.0.0.1', 33645) encerrada.
```

Figure 7: Get Request Execution

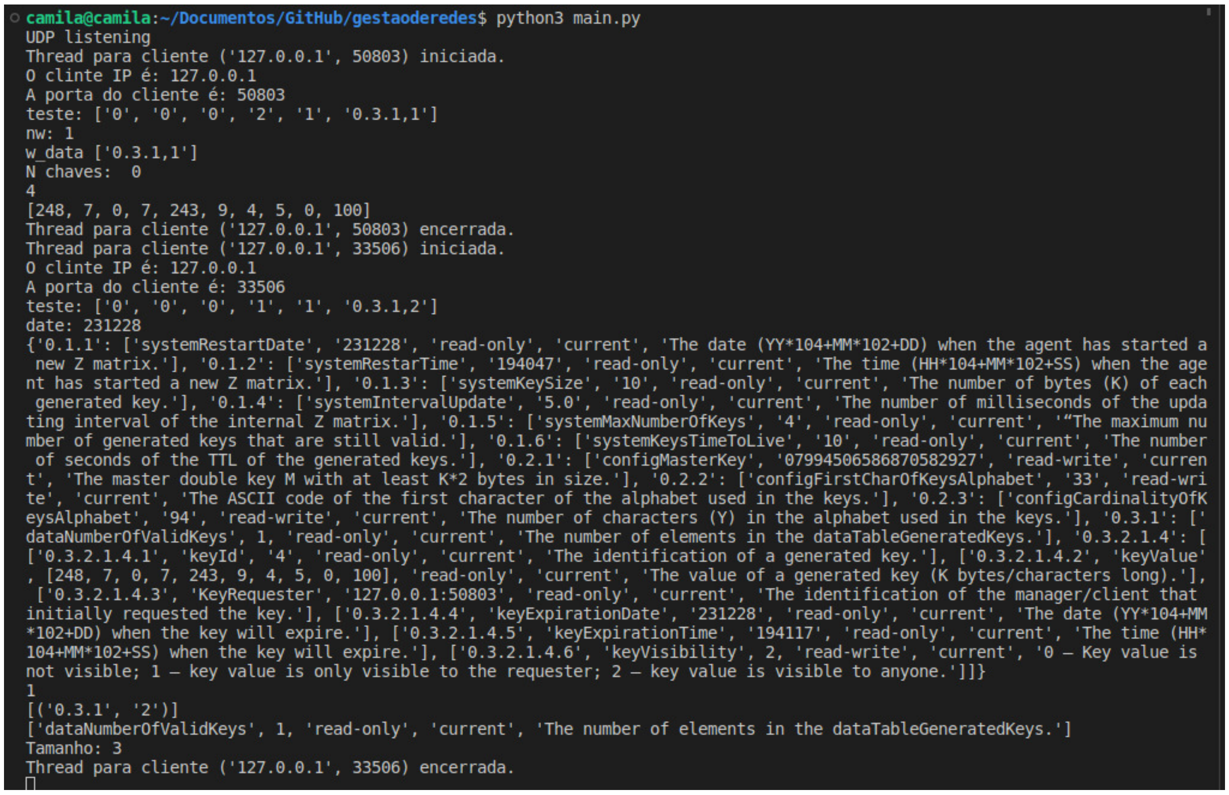
Figure 8 represents a scenario where the manager requests not only the instance of the I-ID but also the instances lexicographically following it.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 gestor.py 1 1 0.3.1,2
Received response from server: [["0.3.1", "['dataNumberOfValidKeys', 1, 'read-only', 'current', 'The number of e
lements in the dataTableGeneratedKeys.']"], ["0.3.2.1.4.1", "['keyId', '4', 'read-only', 'current', 'The identif
ication of a generated key.']"], ["0.3.2.1.4.2", "['keyValue', [248, 7, 0, 7, 243, 9, 4, 5, 0, 100], 'read-only'
, 'current', 'The value of a generated key (K bytes/characters long).']"]]
[O erros]
camila@camila:~/Documentos/GitHub/gestaodered
```

Figure 8: Get Multiple Request

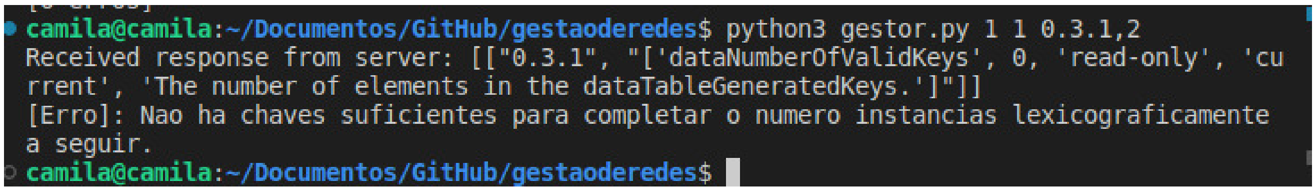
Figure 9 illustrates the execution of the request in the scenario where the manager requests not only the instance of the I-ID but also the instances lexicographically following it. The agent searches in the MIB to check if the I-ID and the instances requested by the manager exist. If they do, the agent sends the required information.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 main.py
UDP listening
Thread para cliente ('127.0.0.1', 50803) iniciada.
O clinte IP é: 127.0.0.1
A porta do cliente é: 50803
teste: ['0', '0', '0', '2', '1', '0.3.1,1']
nw: 1
w_data ['0.3.1,1']
N chaves:  0
4
[248, 7, 0, 7, 243, 9, 4, 5, 0, 100]
Thread para cliente ('127.0.0.1', 50803) encerrada.
Thread para cliente ('127.0.0.1', 33506) iniciada.
O clinte IP é: 127.0.0.1
A porta do cliente é: 33506
teste: ['0', '0', '0', '1', '1', '0.3.1,2']
date: 231228
{'0.1.1': ['systemRestartDate', '231228', 'read-only', 'current', 'The date (YY*104+MM*102+DD) when the agent has started a
 new Z matrix.'], '0.1.2': ['systemRestarTime', '194047', 'read-only', 'current', 'The time (HH*104+MM*102+SS) when the age
nt has started a new Z matrix.'], '0.1.3': ['systemKeySize', '10', 'read-only', 'current', 'The number of bytes (K) of each
 generated key.'], '0.1.4': ['systemIntervalUpdate', '5.0', 'read-only', 'current', 'The number of milliseconds of the upda
ting interval of the internal Z matrix.'], '0.1.5': ['systemMaxNumberOfKeys', '4', 'read-only', 'current', '“The maximum nu
mber of generated keys that are still valid.'], '0.1.6': ['systemKeysTimeToLive', '10', 'read-only', 'current', 'The number
 of seconds of the TTL of the generated keys.'], '0.2.1': ['configMasterKey', '07994506586870582927', 'read-write', 'curren
t', 'The master double key M with at least K*2 bytes in size.'], '0.2.2': ['configFirstCharOfKeysAlphabet', '33', 'read-wri
te', 'current', 'The ASCII code of the first character of the alphabet used in the keys.'], '0.2.3': ['configCardinalityOfK
eysAlphabet', '94', 'read-write', 'current', 'The number of characters (Y) in the alphabet used in the keys.'], '0.3.1': ['
dataNumberOfValidKeys', 1, 'read-only', 'current', 'The number of elements in the dataTableGeneratedKeys.'], '0.3.2.1.4': [
['0.3.2.1.4.1', 'keyId', '4', 'read-only', 'current', 'The identification of a generated key.'], ['0.3.2.1.4.2', 'keyValue'
, [248, 7, 0, 7, 243, 9, 4, 5, 0, 100], 'read-only', 'current', 'The value of a generated key (K bytes/characters long).'],
 ['0.3.2.1.4.3', 'KeyRequester', '127.0.0.1:50803', 'read-only', 'current', 'The identification of the manager/client that
initially requested the key.'], ['0.3.2.1.4.4', 'keyExpirationDate', '231228', 'read-only', 'current', 'The date (YY*104+MM
*102+DD) when the key will expire.'], ['0.3.2.1.4.5', 'keyExpirationTime', '194117', 'read-only', 'current', 'The time (HH*
104+MM*102+SS) when the key will expire.'], ['0.3.2.1.4.6', 'keyVisibility', 2, 'read-write', 'current', '0 – Key value is
not visible; 1 – key value is only visible to the requester; 2 – key value is visible to anyone.']]}
1
[('0.3.1', '2')]
['dataNumberOfValidKeys', 1, 'read-only', 'current', 'The number of elements in the dataTableGeneratedKeys.']
Tamanho: 3
Thread para cliente ('127.0.0.1', 33506) encerrada.
```

Figure 9: Get Multiple Request Execution

In Figure 10, the case represents where there are not enough lexicographically following instances to fulfill the manager's request. In such a situation, the agent sends the existing instances along with an error message.



```
camila@camila:~/Documentos/GitHub/gestaoderedes$ python3 gestor.py 1 1 0.3.1,2
Received response from server: [["0.3.1", "['dataNumberOfValidKeys', 0, 'read-only', 'cu
rrent', 'The number of elements in the dataTableGeneratedKeys.']"]]
[Erro]: Nao ha chaves suficientes para completar o numero instancias lexicograficamente
a seguir.
camila@camila:~/Documentos/GitHub/gestaoderedes$
```

Figure 10: Multiple Request Execution Error

4.3 Revoke Key

Before executing the set and get primitives, the agent thoroughly traverses the MIB, checking the expiration date of each key. If the date has expired, the key is removed; otherwise, it remains unchanged. This verification ensures the integrity and updating of the MIB before any set or get operation.

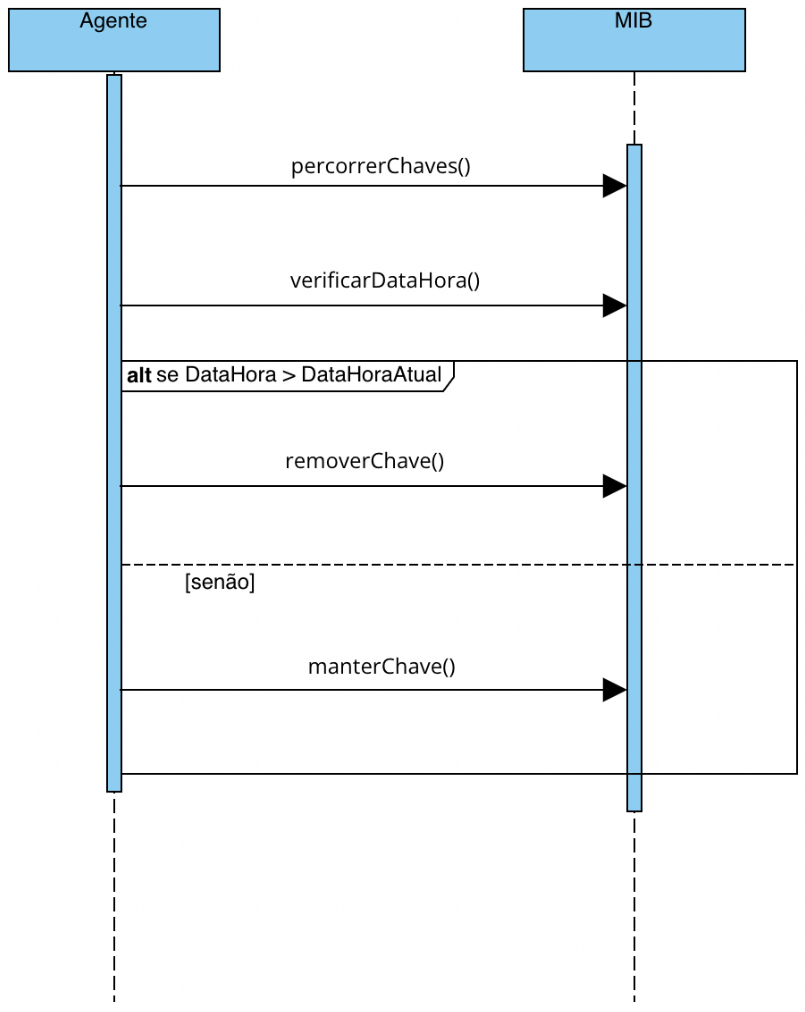


Figure 11: Sequence Diagram Revoke Key

5 User Guide

Server: python3 main.py

Client: python3 gestor.py

- snmpset: python3 gestor.py 2 [quantidadepares] [oid]
- snmpget: python3 gestor.py 1 [quantidadepares] [oid],[nºinstancialexicograficamenteseguintes]

6 Conclusion

The realization of this practical work has allowed us to solidify our knowledge about the architecture of the Internet-standard Network Management Framework (INMF). Although we adopted only an analogous approach for our SNMPkeys model, it was possible to gain hands-on experience in the implementation and understanding the underlying concepts. We deepened our understanding of the protocol layer and the processes involved in key exchange between the agent and managers, as well as the functionalities involved in the process. Additionally, during the completion of this practical work, we comprehended the structure of the Management Information Base (MIB), understood its functioning, and applied it to the functionalities.

Overall, we considered that we implemented each block - key generation and maintenance, communication, protocol, and MIB - correctly, along with the functionalities of responding to a key generation request and key verification.

Given that the current model does not address security methods and lacks authentication, confidentiality, or access control requirements, the implementation of these aspects could be considered a potential future development.