好的，下面是如何从环境变量和配置文件加载密钥的示例。 同样需要强调，安全存储和加载密钥至关重要！

文件结构：

```
/
├── server.py
├── client.py
└── config.ini
```

1. `config.ini` (配置文件示例)

```
[security]
secret_key = your_super_secret_key  #不要存储敏感信息在这里，用于测试，实际可以使用更安全的做法比如服务器令牌等
```

2. `server.py` (服务端 - 从环境变量/配置文件加载密钥)

```
import socket
import threading
import sys
import os
import nacl.secret
import nacl.utils
from nacl.exceptions import CryptoError
import configparser  # 导入 configparser

SERVER_HOST = '0.0.0.0'
SERVER_PORT = 8080
TARGET_HOST = '0.0.0.0'
TARGET_PORT = 80
MAX_CONNECTIONS = 10
BUFFER_SIZE = 4096

def load_secret_key():
    """尝试从环境变量或配置文件加载密钥"""

    # 1. 尝试从环境变量读取密钥
    secret_key = os.environ.get("PROXY_SECRET_KEY")  # 环境变量名可以自定义
    if secret_key:
        print("从环境变量加载密钥")
        return secret_key.encode('utf-8')  # 转换为字节

    # 2. 如果环境变量不存在，则从配置文件读取
    config = configparser.ConfigParser()
    config.read("config.ini") #确保文件存在
    try:
        secret_key = config["security"]["secret_key"]
        print("从config.ini加载密钥")
```

```
            return secret_key.encode('utf-8') # 转换为字节
        except KeyError:
            print("无法从config.ini加载密钥")
            pass
        # 3. 如果两者都不存在，抛出异常或使用默认密钥(非常不建议!!!)
        raise ValueError("未找到密钥：请设置环境变量 'PROXY_SECRET_KEY' 或在 config.ini
中配置")

    # 加载密钥
    try:
        SECRET_KEY = load_secret_key()
        # 保证秘钥长度
        while len(SECRET_KEY) < 32:
            SECRET_KEY += b'\0'
        SECRET_KEY = SECRET_KEY[:32]  # 截断到 32 字节
    except ValueError as e:
        print(f"密钥加载错误：{e}")
        sys.exit(1)  # 退出程序
    def handle_client(client_socket, client_address):
        """处理单个客户端连接，进行 ChaCha20 加密通信"""
        print(f"接受来自 {client_address} 的连接")
        try:
            # 连接到目标服务器
            target_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            target_socket.connect((TARGET_HOST, TARGET_PORT))
            print(f"连接到目标服务器 {TARGET_HOST}:{TARGET_PORT}")
            box = nacl.secret.SecretBox(SECRET_KEY)
            # 创建线程函数，用于从客户端转发到目标服务器
            def forward_client_to_target(client_sock, target_sock,secret_box):
                try:
                    while True:
                        encrypted_data = client_sock.recv(BUFFER_SIZE)
                        if not encrypted_data:
                            break
                        nonce = encrypted_data[:nacl.secret.SecretBox.NONCE_SIZE]  #前
24位是nonce
                        encrypted = encrypted_data[nacl.secret.SecretBox.NONCE_SIZE:]
#后面的内容才是加密的
                        try:
                            data = secret_box.decrypt(encrypted,nonce)
                            target_sock.sendall(data)

                        except CryptoError as e:
                             print(f"解密错误:{e}")
                             break


                except Exception as e:
                    print(f"客户端到目标转发错误：{e}")
                finally:
                    client_sock.close()
                    target_sock.close()
                    print(f"客户端 {client_address} 到目标服务器的连接已关闭")
```

```
        #创建线程函数，用于从目标服务器转发到客户端
        def forward_target_to_client(client_sock, target_sock,secret_box):
             try:
                while True:
                    data = target_sock.recv(BUFFER_SIZE)
                    if not data:
                        break;

                    #使用随机数加密数据
                    nonce = nacl.utils.random(nacl.secret.SecretBox.NONCE_SIZE)
                    encrypted = box.encrypt(data, nonce)

                    # 发送随机数+加密数据给客户端
                    client_sock.sendall(encrypted)
             except Exception as e:
                print(f"目标服务器到客户端转发错误:{e}")
             finally:
                 client_sock.close()
                 target_sock.close()
                 print(f"目标服务器到客户端连接已关闭")
        # 分别启动从客户端到目标，以及从目标到客户端的转发线程
        client_to_target_thread =
threading.Thread(target=forward_client_to_target, args=(client_socket,
target_socket,box))
        target_to_client_thread =
threading.Thread(target=forward_target_to_client,args=(client_socket,
target_socket,box))
        client_to_target_thread.start()
        target_to_client_thread.start()

        # 等待两个线程结束
        client_to_target_thread.join()
        target_to_client_thread.join()

    except Exception as e:
        print(f"处理客户端连接时出错: {e}")
    finally:
        client_socket.close()

def main():
    """服务端主函数"""
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  # 允许地
址重用
    try:
        server_socket.bind((SERVER_HOST, SERVER_PORT))
        server_socket.listen(MAX_CONNECTIONS)
        print(f"服务端监听在 {SERVER_HOST}:{SERVER_PORT}")

        while True:
            client_socket, client_address = server_socket.accept()
            client_thread = threading.Thread(target=handle_client, args=
(client_socket, client_address))
            client_thread.start()
```

```
    except Exception as e:
        print(f"服务端出错: {e}")
    finally:
        server_socket.close()
        print("服务端关闭")

if __name__ == "__main__":
    main()
```

3. `client.py` (客户端 - 从环境变量/配置文件加载密钥)

```
import socket
import threading
import sys
import os
import nacl.secret
import nacl.utils
from nacl.exceptions import CryptoError
import configparser

SERVER_HOST = 'your_server_ip'
SERVER_PORT = 8080
LOCAL_HOST = '127.0.0.1'
LOCAL_PORT = 9090
BUFFER_SIZE = 4096

def load_secret_key():
    """尝试从环境变量或配置文件加载密钥"""

    # 1. 尝试从环境变量读取密钥
    secret_key = os.environ.get("PROXY_SECRET_KEY")  # 环境变量名可以自定义
    if secret_key:
        print("从环境变量加载密钥")
        return secret_key.encode('utf-8')  # 转换为字节

    # 2. 如果环境变量不存在，则从配置文件读取
    config = configparser.ConfigParser()
    config.read("config.ini")
    try:
        secret_key = config["security"]["secret_key"]
        print("从config.ini加载密钥")
        return secret_key.encode('utf-8')  # 转换为字节
    except KeyError:
        print("无法从config.ini加载密钥")
        pass

    # 3. 如果两者都不存在，抛出异常或使用默认密钥(非常不建议!!!)
    raise ValueError("未找到密钥：请设置环境变量 'PROXY_SECRET_KEY' 或在 config.ini
中配置")
```

```
# 加载密钥
try:
    SECRET_KEY = load_secret_key()
    # 保证秘钥长度
    while len(SECRET_KEY) < 32:
        SECRET_KEY += b'\0'
    SECRET_KEY = SECRET_KEY[:32]  # 截断到 32 字节
except ValueError as e:
    print(f"密钥加载错误: {e}")
    sys.exit(1)  # 退出程序

def handle_client(client_socket, client_address):
    """处理来自本地浏览器的连接，转发到服务端，使用 ChaCha20 加密"""
    try:
        # 连接到服务端
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.connect((SERVER_HOST, SERVER_PORT))

        box = nacl.secret.SecretBox(SECRET_KEY)
        # 创建线程，用于从客户端转发到服务端
        def forward_client_to_server(client_sock, server_sock,secret_box):
            try:
                while True:
                    data = client_sock.recv(BUFFER_SIZE)
                    if not data:
                        break

                    # 使用随机数加密数据
                    nonce = nacl.utils.random(nacl.secret.SecretBox.NONCE_SIZE)
                    encrypted = secret_box.encrypt(data, nonce)

                    #发送随机数+加密数据给服务器
                    server_sock.sendall(encrypted)
            except Exception as e:
                print(f"客户端到服务端转发错误: {e}")
            finally:
                client_sock.close()
                server_sock.close()

        # 创建线程，用于从服务端转发到客户端
        def forward_server_to_client(client_sock, server_sock, secret_box):
            try:
                while True:
                     encrypted_data = server_sock.recv(BUFFER_SIZE)
                     if not encrypted_data:
                        break

                     nonce = encrypted_data[:nacl.secret.SecretBox.NONCE_SIZE]
#NonceSize = 24字节

                     encrypted =
encrypted_data[nacl.secret.SecretBox.NONCE_SIZE:]#剩下的为加密数据
                     try:
```

```
                        data = secret_box.decrypt(encrypted,nonce)
                        client_sock.sendall(data)

                    except CryptoError as e:
                        print(f"解密错误:{e}")
                        break

            except Exception as e:
                print(f"服务端到客户端转发错误: {e}")
            finally:
                client_sock.close()
                server_sock.close()
        # 分别启动从客户端到目标，以及从目标到客户端的转发线程
        client_to_server_thread =
threading.Thread(target=forward_client_to_server, args=(client_socket,
server_socket,box))
        server_to_client_thread =
threading.Thread(target=forward_server_to_client,args=(client_socket,
server_socket,box))
        client_to_server_thread.start()
        server_to_client_thread.start()

        # 等待线程结束
        client_to_server_thread.join()
        server_to_client_thread.join()

    except Exception as e:
        print(f"处理本地客户端连接时出错: {e}")
    finally:
        client_socket.close()


def main():
    """客户端主函数"""
    local_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    local_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    try:
        local_socket.bind((LOCAL_HOST, LOCAL_PORT))
        local_socket.listen(10)
        print(f"客户端监听在 {LOCAL_HOST}:{LOCAL_PORT}")

        while True:
            client_socket, client_address = local_socket.accept()
            client_thread = threading.Thread(target=handle_client, args=
(client_socket, client_address))
            client_thread.start()

    except Exception as e:
        print(f"客户端出错: {e}")
    finally:
        local_socket.close()
```

```
if __name__ == "__main__":
    main()
```

修改说明:

1. **导入 `configparser`:** 在 `server.py` 和 `client.py` 中导入 `configparser` 模块。
2. **`load_secret_key()` 函数:**
    - **环境变量优先:** 首先尝试从环境变量 `PROXY_SECRET_KEY` 中读取密钥。 如果没有设置该环境变量，则继续尝试从配置文件读取。环境变量命名可根据实际需要修改
    - **配置文件:** 如果环境变量不存在，则使用 `configparser` 读取 `config.ini` 文件中的 `security` 节的 `secret_key` 值。
    - **错误处理:** 如果都找不到密钥，则抛出一个 `ValueError` 异常。 **永远不要使用默认密钥**，因为它非常不安全。 可以在加载失败时选择退出程序或使用更安全的替代方案。 加载方法为优先从环境变量中加载，如果没有环境变量则会从config.ini中加载，如果都加载不到会报错
3. **加载密钥:** 在代码的开头调用 `load_secret_key()` 函数，并将返回的密钥存储在 `SECRET_KEY` 变量中。 务必处理 `ValueError` 异常，以防止程序在没有密钥的情况下继续运行。

使用方法:

1. 创建 `config.ini` 文件，并添加 `security` 节和 `secret_key` 选项。 **注意不要将敏感信息存储在配置文件中，尤其是不应该将配置文件提交到版本控制系统!**
2. 或者，设置环境变量 `PROXY_SECRET_KEY`。例如，在 Linux/macOS 上：`export PROXY_SECRET_KEY="your_super_secret_key"`，在 Windows 上：`set PROXY_SECRET_KEY=your_super_secret_key`
3. 确保服务端和客户端使用相同的密钥。
4. 运行服务端和客户端脚本。

更安全的替代方案:

- **使用专门的密钥管理服务:** 例如 HashiCorp Vault、AWS KMS、Azure Key Vault 等。
- **使用硬件安全模块 (HSM):** HSM 提供最高级别的密钥保护。
- **加密配置文件:** 使用 GPG 或其他加密工具对配置文件进行加密。
- **从安全存储加载密钥:** 例如，可以将密钥存储在 Linux 的 `keyring` 中。

选择最适合你的安全需求和部署环境的密钥管理方法。请务必阅读和理解相关文档，并遵循最佳安全实践。