Pseudo code for implementing security for API keys and sensitive data

Instructions:

1. Get the Tools
    - Import stuff like os (for secrets), dotenv (for the .env file), and logging (to write messages).
2. Set Up Logging
    - Turn on logging to track what's happening (like errors or success).
3. Load API Keys
    - Use load_dotenv() to load API keys from a hidden file called .env.
    - Make sure .env is in .gitignore so it doesn't get uploaded online.
    - Check if all keys are there; if one is missing, log an error.
4. Check If Keys Are Okay
    - Make sure the keys are:
        - Strings.
        - Long enough (like 10+ characters).
    - Throw an error if they're missing or wrong.
5. Hide Keys in Logs
    - When showing keys in logs, only show the first and last few characters.
        - Ex. abcd1234 → ab***34.
6. Lock the Keys
    - Use encryption to lock the keys (optional but safer).
    - Save the locked version so no one can steal them.
7. Unlock the Keys
    - When you need to use the keys, unlock them with a secret password.
8. Use the Keys
    - Add the keys securely when making API requests (don't print them out or show them).
9. Catch Problems
    - If anything goes wrong (missing keys, errors), write it in the logs so you can fix it.
10. Run the Program
    - Run everything by calling the main() function. It will load, check, lock, and use the keys safely.

```
# Import necessary libraries
import os
from dotenv import load_dotenv
import logging

# Initialize logging
logging.basicConfig(level=logging.INFO)

# Load environment variables from a .env file
def load_environment_variables():
    """
    Load environment variables from a .env file securely.
    Ensure .env file is included in .gitignore to avoid accidental commits.
    """
    load_dotenv()  # Loads .env file into the environment
    API_KEYS = {
        "PLAID": os.getenv("PLAID_KEY"),
        "OPEN_EXCHANGE": os.getenv("OPEN_EXCHANGE_KEY"),
        "ALPACA": os.getenv("ALPACA_KEY"),
        "FIREFLY": os.getenv("FIREFLY_KEY")
    }

    for key, value in API_KEYS.items():
        if not value:
            logging.error(f"Missing API key for {key}. Check your .env file.")

    return API_KEYS

# Validate sensitive data
def validate_keys(api_keys):
    """
    Validate that all API keys are loaded and follow expected patterns.
    """
    for key, value in api_keys.items():
        if not value:
            raise ValueError(f"API key for {key} is missing.")
        if not isinstance(value, str) or len(value) < 10:  # Adjust length as appropriate
            raise ValueError(f"Invalid API key format for {key}.")

# Mask sensitive data in logs
def mask_sensitive_data(data):
    """
    Mask sensitive data for logging or debugging purposes.
```

```
    """
    if not data:
        return "N/A"
    if len(data) <= 4:
        return "***"
    return data[:2] + "***" + data[-2:]

# Encrypt API keys for runtime use (optional)
def encrypt_key(key, secret):
    """
    Encrypt an API key with a secret key (optional step for enhanced security).
    """
    from cryptography.fernet import Fernet
    cipher_suite = Fernet(secret)
    encrypted_key = cipher_suite.encrypt(key.encode())
    return encrypted_key

def decrypt_key(encrypted_key, secret):
    """
    Decrypt an API key for use at runtime.
    """
    from cryptography.fernet import Fernet
    cipher_suite = Fernet(secret)
    decrypted_key = cipher_suite.decrypt(encrypted_key).decode()
    return decrypted_key

# Store and retrieve keys securely
def secure_storage_demo():
    """
    Demonstrate secure storage and retrieval of API keys.
    """
    # Load environment variables
    api_keys = load_environment_variables()

    # Validate API keys
    validate_keys(api_keys)

    # Log keys with masking
    for key, value in api_keys.items():
        logging.info(f"{key}: {mask_sensitive_data(value)}")

    # Optional: Encrypt keys for runtime use
    secret = os.getenv("ENCRYPTION_SECRET")  # Load a secret key from environment
    encrypted_keys = {k: encrypt_key(v, secret) for k, v in api_keys.items()}
```

```
    # Decrypt when needed
    decrypted_keys = {k: decrypt_key(v, secret) for k, v in encrypted_keys.items()}

    return decrypted_keys

# Secure API key usage in app
def use_api_keys():
    """
    Use API keys securely in the application.
    """
    api_keys = secure_storage_demo()

    # Example API request with secure key usage
    headers = {
        "Authorization": f"Bearer {api_keys['PLAID']}"
    }
    response = requests.get("https://api.plaid.com/some-endpoint", headers=headers)
    return response.json()

# Main function to execute security processes
def main():
    """
    Main function to ensure API keys are securely managed.
    """
    try:
        decrypted_keys = secure_storage_demo()
        logging.info("Secure storage demo completed successfully.")
    except ValueError as e:
        logging.error(f"Error in secure storage: {e}")
    except Exception as e:
        logging.error(f"Unexpected error: {e}")

# Run the main function
if __name__ == "__main__":
    main()
```