CS 4351 Computer Security, Fall 2026
University of Texas at El Paso
Homework 2: Cryptographic Algorithms and Implementation
Due on 09/24/2026, at 11:59pm
Total: 60 points
You need to submit a pdf generated from LaTeXfor your analysis report. If you do not have a local LaTeXsetup, I suggest
https://www.overleaf.com/. You need to write the math formulas correctly using LaTeX.
This homework should be finished independently. AI violations are taken seriously.

Overview

This homework will allow you to gain hands-on experience with practical cryptographic algorithms and libraries. You will complete two SEED Security Labs focused on symmetric-key encryption and public-key cryptography, then analyze your results and performance measurements.

Lab Environment Setup

Single VM for All Tasks: Both Lab 1 (Secret-Key Encryption) and Lab 2 (RSA Encryption) can be completed on the same Ubuntu 20.04 VM provided by SEED Labs. All necessary cryptographic tools and libraries (OpenSSL) are pre-installed.

Setup Instructions:

1. Visit the SEED Labs website: `https://seedsecuritylabs.org/`
2. Download and set up the Ubuntu 20.04 VM following the detailed instructions at: `https://seedsecuritylabs.org/labsetup.html`
3. No additional software installation is required for either lab
4. For detailed task descriptions and context, refer to the official lab PDFs available on each lab webpage

Note for Apple Silicon (M1/M2/M3/M4) Users: The standard SEED Labs x86-64 VMs may not run directly on Apple Silicon Macs via VirtualBox. Alternative options include: (1) **VMware Fusion** - we have tested this appraoch. (2) Docker containers (if available for the labs), (3) Parallels Desktop with ARM64 Ubuntu, or (4) Cloud-based VMs (AWS, Linode, DigitalOcean). Consult with your TA if you experience compatibility issues. The TA has also prepared a separte labsetup document for Apple Silicon users: link.

Required Labs and Tasks:

Lab 1: Secret-Key Encryption Lab [30 points]

Lab Link: `https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Encryption/`

Complete the Secret-Key Encryption Lab using OpenSSL. This lab explores symmetric encryption algorithms (AES) and the importance of different encryption modes. You will encrypt and decrypt files using various cipher modes to understand their security properties and performance characteristics.

Detailed Task Breakdown:

1. **[5 points] Task 1: Basic AES Encryption in Different Modes**

   Learn how to use OpenSSL command-line tools to encrypt files. Specifically:

   - **Test File:** Create a simple plaintext file to encrypt. You can use any of these approaches:
     - Create a text file: `echo "Test message for encryption" > plaintext.txt`

  - Or use an existing file: `cp /etc/hosts plaintext.txt`
  - Or create a binary file: `dd if=/dev/urandom of=plaintext.bin bs=1K count=1`
- **Generate your own key and IV:** Do not use hard-coded values. Generate random cryptographic keys:
  - Generate a 128-bit AES key: `openssl rand -hex 16`
  - Generate an IV: `openssl rand -hex 16`
  - Record these for use in all encryption operations
- Encrypt the file using AES-128 in ECB mode: `openssl enc -aes-128-ecb -in plaintext -out ciphert`
- Encrypt the same file using AES-128 in CBC mode: `openssl enc -aes-128-cbc -in plaintext -out cip`
- Repeat with AES-256 keys for both modes (generate a 256-bit key using `openssl rand -hex 32`)
- Decrypt each file to verify you recover the original plaintext
- Document the commands used and verify successful decryption

For detailed context and additional cipher modes, see the official SEED Labs PDF handout at the link above.

2. **[5 points] Task 2: ECB Mode Vulnerability Analysis (Image-Based)**

   Understand why ECB (Electronic Codebook) mode is cryptographically insecure by visually comparing encrypted images:

   - **Image File Setup (Required):**
     - **Option A (Easiest):** Download a small image file (PNG, BMP, JPG) from the internet (size: 1-5 MB)
     - **Option B:** Convert an image to raw binary format: `convert image.png raw:image.bin` (requires ImageMagick)
     - **Option C:** Use a simple PPM image (SEED Labs documentation may provide example images)
   - **Encryption Steps (Using OpenSSL):**
     - **Generate your own key and IV:**
       * Generate a random 128-bit key (32 hex characters):
         ```
         openssl rand -hex 16 > key.txt
         KEY=$(cat key.txt)
         ```
       * Generate a random 128-bit IV (32 hex characters):
         ```
         openssl rand -hex 16 > iv.txt
         IV=$(cat iv.txt)
         ```
       * Or use the example key and IV below for testing (but use your own for submission):
         · Example KEY: `00112233445566778899aabbccddeeff` (DO NOT use this in actual submission)
         · Example IV: `00000000000000000000000000000000` (DO NOT use this in actual submission)
     - Encrypt with ECB mode (using your key):
       `openssl enc -aes-128-ecb -in image.bin -out image_ecb.bin -K $KEY -noiv`
     - Encrypt with CBC mode (using your key and IV):
       `openssl enc -aes-128-cbc -in image.bin -out image_cbc.bin -K $KEY -iv $IV`
   - **Visualization and Comparison:**
     - **Method 1 (Binary Comparison):** View hex dumps to compare patterns:
       * `hexdump -C image_ecb.bin` head -30—
       * `hexdump -C image_cbc.bin` head -30—
       * Note: ECB will show repeating byte patterns; CBC will appear random
     - **Method 2 (Visual - if using image files):** Convert encrypted binaries back to viewable images:

        * If original was PNG/BMP: Try viewing encrypted versions in an image viewer (they will appear corrupted)
        * Observe: ECB-encrypted images may still show recognizable patterns/shapes; CBC-encrypted images will appear as noise
    - **Analysis and Report:**
        - Include hex dumps or screenshots showing side-by-side comparison
        - Explain why ECB reveals patterns even in ciphertext
        - Write 2-3 sentences explaining: (i) What patterns appear in ECB mode, (ii) Why CBC does not show these patterns, (iii) Why this makes ECB insecure

    This task visually demonstrates why ECB mode is dangerous in cryptographic systems.

3. **[5 points] Task 3: Multiple Cipher Modes Comparison**

    Explore and implement encryption using various modes of operation:

    - Implement encryption and decryption using at least three different modes: CBC, CTR, and one additional mode (OFB, CFB, GCM, etc.)
    - Test each mode with different key sizes (128-bit and 256-bit minimum)
    - Create a test file (at least 1KB) and verify that decryption produces the original plaintext for each mode
    - Document the OpenSSL commands used for each mode

4. **[5 points] Task 4: Performance Measurement and Analysis**

    Measure and document encryption/decryption performance:

    - **Encryption Steps (Using OpenSSL):**
        - **Generate your own key and IV for all tests:**
            * `KEY=$(openssl rand -hex 16)` (for AES-128)
            * `IV=$(openssl rand -hex 16)`
            * For AES-256: `KEY=$(openssl rand -hex 32)`
        - Measure encryption time for AES-128 in CBC mode:
          `time openssl enc -aes-128-cbc -in small_file.bin -out encrypted.bin -K $KEY -i`
        - Measure decryption time:
          `time openssl enc -d -aes-128-cbc -in encrypted.bin -out decrypted.bin -K $KEY`
        - Repeat measurements multiple times and average the results (account for system variation)
        - Repeat for all key sizes and modes (128-bit, 256-bit; CBC, CTR, etc.)

5. **[5 points] Task 5: Additional Lab Exploration**

    Complete additional tasks from the official SEED Labs PDF handout, which may include:

    - Exploring IV (initialization vector) usage and importance
    - Understanding padding in block ciphers
    - Analyzing the relationship between key size and security

    Refer to the full lab handout at `https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Encryption/` for complete task descriptions.

Deliverables for Lab 1:

- **Code submission:** Create a file `lab1_commands.txt` containing all OpenSSL commands used for each task

- **Verification output:** Include output logs showing successful encryption and decryption
- **Performance data:** Submit timing results in a CSV or table format
- **Analysis document:** Write your analysis comparing ECB vs CBC security properties
- **All files in ZIP:** Package all code, scripts, logs, and analysis in a file named `lab1.zip`

Lab 2: RSA Encryption and Signature Lab [30 points]

Lab Link: `https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_RSA/`

Complete the RSA Public-Key Encryption and Signature Lab. This lab teaches how to use public-key cryptography for encryption and digital signatures using OpenSSL. You will work with RSA keys, encrypt/decrypt data, and create and verify digital signatures—all essential components of modern cryptographic systems.

Detailed Task Breakdown:

1. **[5 points] Task 1: RSA Key Generation and Timing**

   Learn how to generate RSA key pairs and measure the computational cost:

   - Generate RSA key pairs with multiple key sizes using OpenSSL:
     - 1024-bit key: `openssl genrsa -out private_1024.pem 1024`
     - 2048-bit key: `openssl genrsa -out private_2048.pem 2048`
     - 3072-bit key: `openssl genrsa -out private_3072.pem 3072`
   - Extract public keys from private keys: `openssl rsa -in private_key.pem -pubout -out public_key.`
   - Measure the time it takes to generate each key size using the `time` command
   - Document your measurements in a table

2. **[5 points] Task 2: RSA Encryption and Decryption**

   Implement RSA encryption and decryption with proper padding:

   - **Create test files:**
     - Small file (1KB): `dd if=/dev/urandom of=small_plaintext.bin bs=1K count=1`
     - Larger file (1MB): `dd if=/dev/urandom of=large_plaintext.bin bs=1M count=1`
   - Encrypt files using RSA with PKCS#1 v1.5 padding: `openssl rsautl -encrypt -inkey public_key.pem`
   - Decrypt the ciphertext using the private key: `openssl rsautl -decrypt -inkey private_key.pem -in c`
   - Test with multiple key sizes (2048-bit and 3072-bit minimum)
   - Verify that decrypted content matches the original plaintext (use `diff` or `cmp`)
   - Document all commands and verification results

   Note: RSA can only encrypt data smaller than the key size minus padding overhead (e.g., 2048-bit RSA can encrypt approximately 245 bytes with PKCS#1 v1.5 padding). For larger files, students should handle them block-by-block or use hybrid encryption.

3. **[5 points] Task 3: RSA Encryption Performance Analysis**

   Measure and compare encryption/decryption performance across key sizes:

   - Measure encryption time for 2048-bit and 3072-bit RSA keys using the `time` command
   - Measure decryption time for each key size
   - Test with the small file (1KB) and larger file (1MB)

- Create a performance table including:
  - Key size (2048-bit, 3072-bit)
  - File size (1KB, 1MB)
  - Encryption time
  - Decryption time
  - Throughput (bytes per second) for smaller operations
- Analyze why encryption and decryption times differ

4. **[5 points] Task 4: Digital Signatures**

   Create and verify RSA digital signatures:

   - Sign a file using your private key with SHA-256 hash: `openssl dgst -sha256 -sign private_key.pem -ou`
   - Verify the signature using the public key: `openssl dgst -sha256 -verify public_key.pem -signature`
   - Test signing and verification with 2048-bit and 3072-bit keys
   - Test with both small and large files
   - Measure the time for signing and verification operations
   - Intentionally modify the plaintext file and demonstrate that signature verification fails
   - Document all commands and verification results

5. **[5 points] Task 5: Advanced Topics and Additional Lab Tasks**

   Complete additional exploration from the official SEED Labs PDF, which may include:

   - Understanding OAEP padding vs PKCS#1 v1.5 padding
   - Exploring different hash functions (SHA-1, SHA-256, SHA-512) for signatures
   - Certificate usage in public-key infrastructure (if covered in the lab)
   - Comparing RSA performance with elliptic curve cryptography (if available)

   Refer to the full lab handout at `https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_RSA/` for complete descriptions.

Deliverables for Lab 2:

- **Code/Scripts submission:** All shell scripts, Python scripts, or command logs used for key generation, encryption, decryption, signing, and verification
- **Key files:** Your generated RSA private and public keys (only use for this lab, do not reuse in production)
- **Performance measurements:** Timing results and analysis in table format
- **Test logs:** Output showing successful encryption/decryption and signature verification
- **Analysis document:** Explain key generation time complexity, encryption vs. decryption speed differences, and signature verification results
- **All files in ZIP:** Package all code, keys, logs, and analysis in a file named `lab2.zip`

Performance Analysis Report

In addition to completing the lab tasks, you must create a comprehensive performance analysis report that integrates findings from both labs. This report should be submitted as a single PDF document.

I. Lab Completion Summary

- Clearly list all tasks completed from Lab 1 (Secret-Key Encryption) and Lab 2 (RSA)
- Specify OpenSSL version used and confirm same Ubuntu 20.04 VM was used for both labs
- List any additional optional tasks completed

II. Performance Measurements and Results
Compile all timing results from both labs into well-organized tables:
Lab 1 - Symmetric Encryption Results:

- Table 1: Key generation times (if applicable)
- Table 2: Encryption times for AES-128 and AES-256 in multiple modes (CBC, CTR, etc.)
  - Small file (1KB) and large file (10MB)
  - Include throughput calculations in MB/s
- Table 3: Decryption times for same configurations
- Table 4: Mode comparison and security analysis (ECB vs CBC vs others)

Lab 2 - Public-Key Cryptography Results:

- Table 5: RSA key generation times for 1024-bit, 2048-bit, 3072-bit keys
- Table 6: Encryption times for 2048-bit and 3072-bit RSA keys
  - Small file (1KB) and larger file (1MB)
- Table 7: Decryption times for same key sizes and file sizes
- Table 8: Digital signature generation and verification times
  - Include per-byte timings if applicable

III. Performance Analysis and Interpretation
For each of the following aspects, provide detailed analysis (2-3 sentences each) with (i) expected performance, (ii) observed results, and (iii) justification of any differences:

1. **File Size Impact on Symmetric Encryption:** How does per-byte encryption/decryption speed change between 1KB and 10MB files? Which mode shows the most consistent performance? Why might throughput be affected by file size?
2. **Encryption Mode Efficiency:** Compare the performance of CBC vs. CTR vs. other modes tested. Are there significant differences? Relate performance differences to the operational characteristics of each mode.
3. **Key Size Impact on Encryption:** Analyze how AES key size (128-bit vs. 256-bit) affects encryption speed for symmetric encryption. Is the impact significant?
4. **RSA Key Size Scaling:** Compare RSA performance when key size increases from 2048-bit to 3072-bit. How does key size affect key generation, encryption, decryption, and signing time? Is the relationship linear, exponential, or polynomial?
5. **Symmetric vs. Public-Key Performance:** Compare AES encryption speed with RSA encryption speed. Which is orders of magnitude faster and why? In what practical scenarios would you use each?
6. **Signature Operations:** Is RSA signature generation significantly different from decryption? Is verification significantly faster than signing? Explain why based on RSA mathematics.

7. **Security vs. Performance Trade-off:** Discuss how ECB's insecurity relates to its performance advantage over CBC. Is the performance difference worth the security risk?

IV. Security Observations
Include written analysis of:

- Why ECB mode reveals patterns in plaintext through ciphertext patterns
- The role of padding in RSA encryption and how it affects maximum plaintext size
- Why digital signature verification with RSA is computationally less expensive than generating signatures (hint: public exponent)
- The importance of key size in public-key cryptography for long-term security

V. Submission Requirements

Submit THREE files via your course management system:

1. **`lab1.zip`** - Contains all Lab 1 deliverables:
    - OpenSSL commands used (in a file named `lab1_commands.txt` or `commands.sh`)
    - Output logs showing encryption/decryption success
    - Performance timing data (CSV or text format)
    - Analysis of ECB vs. CBC security properties
    - Any test files or scripts created
2. **`lab2.zip`** - Contains all Lab 2 deliverables:
    - All shell scripts or commands used for each task (named `lab2_commands.txt` or `commands.sh`)
    - Generated RSA key files (both private and public keys)
    - Test output logs showing encryption/decryption and signature verification success
    - Performance timing data and analysis tables
    - Demonstration of signature verification failure on tampered data
    - Any Python or shell scripts used for automation
3. **`hw2_report.pdf`** - Performance Analysis Report (single PDF document) containing:
    - All analysis sections listed above (Sections I-V)
    - Performance tables and figures
    - Detailed written analysis and interpretation
    - References to specific lab tasks and findings
    - Must be generated from LaTeX or a professional document tool

Important: Verify that your ZIP files can be extracted correctly before submission. Test by extracting them on your own system. If files cannot be extracted, you risk losing points. Do *not* include executable binaries; focus on source code, scripts, configuration files, and text output logs.