CSCD27 Computer and Network Security

Assignment 1 : Due: Fri Oct 10, 23:59

The primary goals of this assignment are to improve your understanding of secure email, insider attacks, perfect ciphers, Feistel ciphers, DES, brute-force attacks, block-modes, AES.

As with later assignments, this one includes a mix of written and programming problems. The programming tasks will be covered in some detail in tutorials, and you will have an opportunity to work on your implementation with assistance from the TA. You are strongly encouraged to attend tutorials to take advantage of this help, and to get a timely start on assignment work.

1. [2 marks] **Course Feedback**

   What topic would you most like to see explained in more depth in lecture or discussed in tutorials? You may answer with a single sentence. Send the answer to this question as the content of a signed, encrypted email message as described the PGP/GPG problem below.

2. [5 marks] **PGP/GPG: secure email in practice**

   In this problem, you will gain experience with PGP, a popular format for email encryption. Ordinary email is completely insecure: it is very easy to send forged email with a spoofed `From:` address, and furthermore, in some environments it is easy to eavesdrop or even modify email during transmission.

   Pretty Good Privacy (PGP) was created by Phil Zimmerman as a way for activists to communicate securely over the Internet. Over time, it has evolved into an open standard, called OpenPGP. Several implementations support the PGP message format: PGP is distributed by PGP Corporation; GPG (Gnu Privacy Guard) is an open-source Gnu implementation. Because PGP and GPG are based on common underlying protocols, all the implementations can interoperate.

   Use the instructions provided in this PGP/GPG handout to create a public-private key pair which you will use to send to the TA, your answer to question 1 above.

   Important: Attach your ASCII-armor'd public key to your solution, so that the TA can verify your signature.

3. [2 marks] **GPG: Revoking a Key**

   There are several reasons why you might want or need to revoke an existing GPG key, for example: your private key is stolen or compromised, your key is being replaced by a stronger one, or, because you forget the key's passphrase (it happens, surprisingly often ;-).

Use the instructions provided in this handout to create a revocation certificate for the public-private key pair that you generated for the previous problem.

In general, a revocation certificate should be kept secure, so that someone else can't use it to revoke one of your working keys. However, as evidence of having completed this problem, you should include a copy of your ASCII-armored revocation certificate with your assignment submission. (We promise not to revoke your keys.)

To actually use this certificate to revoke a key, you would import it into your keyring, at which point the associated key would become unusable.

Note, you are not asked to perform the import step as part of this problem, just to generate the revocation certificate for possible future use (e.g. you forget your passphrase).

4. [4 marks] **Insider Threats**

Ken Thompson's Turing-award presentation, "Reflections on Trusting Trust," (posted on Lectures page) describes a technique for creating a difficult-to-detect login "backdoor" to Linux/Unix systems.

You've recently started a new job as the Linux systems administrator for the Physics department. After several complaints from physicists about how slowly their compute-intensive simulation programs are running "ever since you started work in their department (ahem)", you begin to investigate and discover a number of long-running root processes. Initially these appear to be standard system-daemon processes, but you know that those daemons should not be cpu-bound, so something seems a bit fishy.

You learn from one of the physicists that your new job was until recently held by a "rather surly fellow" who was caught using system resources to mine for digital currency, and was subsequently dismissed in a cloud of acrimony.

Armed with this new knowledge, you undertake a more thorough audit of your systems, and discover with some alarm that some key system utilities, including "ls", "ps", "login", and "gcc", have been compromised (tampered).

Since there may be other tampered utilities on your systems, you decide that the only safe way to proceed is to rebuild your Linux systems from source code, which you have verified is untampered. Of course, to do that you need a C compiler, but you can't safely use GCC. Fortunately, you have a copy of the XL C compiler brought from your previous position at IBM, and this resides on your home server, which hasn't been hacked. You know that you can't perform the full Linux recompile using XL, since many Linux utilities were written to take advantage of GCC's C-extensions. However, you can use XL to compile GCC.

Describe how you could determine whether your GCC had been hacked with a backdoor, and how you could safely use it together with XL to rebuild your Linux systems.

5. [4 marks] **Brute-Force Attacks**

Suppose you have access to a modern hexa-core (16-core) computer clocked at 3.4GHz. Assume that each decrypt-checking cycle requires 80 clock cycles on a single core and that each core includes sufficient on-chip cache to be able to decrypt and check candidate-key outputs without having to access the system bus to access shared RAM. You may also ignore the time required to partition the keyspace among the cores.

Estimate the average (not worst case) time required to perform a brute-force key-recovery attack against:

a. DES
b. 3DES
c. AES-128
d. AES-256

If you make any assumptions beyond the ones stated above, be sure to state them with your solution.

6. [8 marks] **Perfect Ciphers**

Suppose Alice and Bob share secret-key K whose value is chosen from the set {0,1,2}, and that Alice wants to send Bob a message M chosen from that same value set {0,1,2}.

(a) If Alice performs encryption as "M XOR K" (using the binary representations of M and K), is this as secure as a perfect substitution cipher (like an OTP)?

(b) If so, explain why it is as secure, and if not, show how you could change the algorithm, but not the message or key configurations, to make it a perfect cipher.

7. [4 marks] **Feistel Ciphers**

Show that Feistel encryption is invertible even when Feistel function `F()` is a one-way (non-invertible) function, by showing algebraically how encryption and decryption are related.

Express your answer in terms of Feistel function `F(), Li, Ri,` and round key `Ki`.

8. [4 marks] **DES for Password Authentication**

The Unix `crypt()` function was the traditional mechanism used to process users' plaintext passwords for authentication. This traditional implementation uses a modified form of the DES algorithm that operates as follows:

- the user's plaintext password is truncated to eight characters and those are coerced down to 7-bits each to form a 56-bit DES key.

- that key is then used to encrypt an all-zero-bit block, with a salt value introduced for randomness, to produce an encrypted block that is then stored in the **`passwd`** file.
- when the user authenticates with their plaintext password, the above process is repeated, and the result compared with the value stored in the **`passwd`** file.

Would this approach be equally secure if the user's password was encrypted, rather than a block containing all 0-bits, using a key-value of all 0-bits? You may assume that users' plaintext passwords are always 8-bytes long, and thus fill a block.

Give a brief justification if you answer "yes", and an example of a serious security vulnerability if you answer "no".

9. [6 marks] **Brute Force Attack on DES**

An adversary, monitoring communicated between Alice and Bob, is able to obtain a single plaintext-ciphertext block-pair (p,c), encrypted under Alice-Bob's secret-key k, in other words, e(k,p) = c.

If the adversary performs a brute-force attack by testing all possible values k' in the DES keyspace to look for k' values such that e(k',p) = c, what is the probability that they will find a unique k' that satisfies this equality, and thus will have broken Alice+Bob's key k?

10. [8 marks] **Block-Cipher Modes of Operation**

ReSaY Secure Payment Systems has asked you to review their new trade-secret block-cipher encryption mode that they plan to sell to credit-card companies to protect Point-of-Sale (PoS) transactions.

PoS terminals need to securely transmit sequences of 128-bit blocks of plaintext: $P_1 \ldots P_n$ using 128-bit secret-key K that each PoS terminal shares with the credit-card company server (each terminal has a different secret-key K).

A PoS terminal initiates an encrypted transmission by choosing a random 128-bit value C_0, which serves as the first block of ciphertext. For i > 0, the i'th ciphertext block is computed as:

$$C_i = C_{i-1} \text{ XOR } \text{AES-128}(K, P_i)$$

The transmitted ciphertext stream C is the catenation (||) of blocks:

$$C = C_0 \,||\, C_1 \,||\, C_2 \,||\, \ldots \,||\, C_n$$

i. [2 marks] What is the role of random-value C_0 in the encryption process? Your answer should express its purpose, not just its name.

ii. [4 marks] What is your assessment of ReSaY's new trade-secret block mode, is it secure enough to be used for credit-card transactions?

If you think it is secure, list some properties that make it secure.

If you think it is not secure, describe at least one serious vulnerability.

iii. [2 marks] If ReSaY tweaked their block-mode by changing the algorithm for encrypting block C_i to:

$$C_i = \text{AES-128}(K, (C_{i-1} \text{ XOR } M_i))$$

would your assessment from part (ii) above change? Briefly explain why/not.

11. [6 marks] **CBC Integrity Vulnerability**

Suppose you determine that the communication between OLG (Ontario Lottery and Gaming) and merchants who sell lottery tickets is encrypted using AES-CBC with a random IV and using an OLG-invented protocol "Plain-Speak". The Plain-Speak message sent to a merchant terminal, when a winning ticket for $100 is submitted by that merchant, looks like this:

```
Give Patron $100
```

or, in Hex:

```
476976652050617472f6e2024313030
```

which is encrypted with AES-CBC as:

```
b1710117cfe1cc5549bbb45f0bad1c8c ae11fb7f135d0ac6591bf66facf651b7
```

Your challenge is to modify the above AES-CBC message so that it will decrypt to string "Give Patron $987" - without the quotes ($999 might arouse suspicions, so let's not be greedy).

Express your solution as a hex-encoded AES-CBC encrypted message like the one shown above.

12. [28 marks] **Advanced Encryption Standard (AES) Implementation**

For the programming portion of this assignment you will develop an implementation of the AES block cipher in Python. Initially, you may find this coding to be a challenge, but as you become more familiar with the AES algorithm components, I hope you come to appreciate the underlying simplicity and structure that makes AES reasonably understandable.

The top-level **`encrypt()`** and **`decrypt()`** functions take parameter key and plaintext/ciphertext values expressed as 32-character hexadecimal strings, and both encrypt() and decrypt() return 128-bit values, also expressed as 32-character hex strings.

Although AES supports key sizes of 128, 192, and 256 bits, you need only implement support for 128-bit keys. Also you may assume that plaintext and ciphertext values are exactly 128-bits in length, in order to avoid block-padding issues, which are not part of the AES algorithm per se.

AES is a reasonably complicated algorithm, and with this in mind, we will be covering AES algorithm-details in both lectures and tutorials. To help you structure your solution, starter code is provided (linked to the Assignments Web page).

Unit testing

Implement a set of nose tests to convince yourself that your functions behave as expected. The marker(s) will supply their own nose tests for your code.

Black box unit testing of a complex encryption algorithm like AES can be very challenging, since the slightest error would likely lead to complete corruption of the output result. To mitigate this, you are strongly encouraged to unit-test your code using the intermediate computation values provided in Appendices A1, B, and C1 of the NIST AES standard (link also provided in Assignment 1 resources list). These intermediate values will allow you to spot an erroneous value as soon as it creeps into your computation.

You may want to try your hand at "test-driven development" for this problem. This development methodology advocates writing tests first and then developing code to pass the tests. If you start by writing unit tests based on the NIST AES standard's test vectors, you will have a solid method for assessing whether your code is working. Consider writing tests that examine intermediate values in addition to final output values (e.g. test whether a single step of **`shift_rows()`** is working).

Work Packages

To reduce the individual workload for this problem, and to make the exercise more realistic, the implementation has been divided up into the roughly equal "work packages" listed below. Pick any two of these packages to implement, or, alternatively, you may elect to work with a partner, sharing the work to complete all four work packages.

1. Key-handling functions: sbox_lookup, sub_key_bytes, init_key_schedule, add_round_key, unit testing (14 marks)
2. S-box byte substitution and row shifting including: sbox_lookup, inv_sbox_lookup, sub_bytes, inv_sub_bytes, shift_bytes_left, shift_bytes_right, shift_rows, inv_shift_rows, unit testing (14 marks)
3. Column mixing, including: mix_columns, inv_mix_columns, gf_mult, unit testing (14 marks)
4. Encryption and decryption: the top-level functions that transform plaintext to ciphertext and back, comprehensive unit testing (14 marks)

Assessment

Be sure to **clearly indicate** in your submission which work packages you selected, and if working with partner(s), make sure each partner(s) name(s) appears in code-header comments.

The marker will supply correct implementations of work packages that you do not implement, and will then run tests against this complete implementation.

Your implementation will be marked based on correctness, adequacy of nose tests, and adherence to good coding practices such as modularity, use of comments, descriptive variable names, etc.

Submission Instructions

Submit on mathlab.utsc your answers to written problems in a1.txt (or a1.pdf), Python file aes.py containing your implementations of your chosen AES functions for the programming problem, together with file aes_test.py containing your AES nose tests.

```
submit -c cscd27f14 -a a1 a1.txt aes.py aes_test.py
```

No hardcopy submission is required, except for the assignment cover sheet.

References

1. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf