CS 1699: Privacy in the Electronic Society

Project 1

Released: Tuesday, February 13

Due: Thursday, March 1, 11:59 PM

Motivation

In this course, we've discussed timing attacks in depth. In this project, you will implement and test an algorithm that attempts to obscure the timing information needed to carry out such an attack, by being (nearly) constant in runtime among inputs of the same size.

Your submission should consist of the following components.

- A writeup that completes the W tasks below
- Code that satisfies the C tasks below, in the language of your choice (check with Mingda if you're not sure it will work for him)

All features, bugs, and other details with your code should be made clear in your writeup (i.e., do not submit a separate README or expect your TA to read every comment in your code). Each writing task should be clearly titled, and each code task should be clearly discussed in the writeup. In short, do not make your TA search for the components of your submission. Show off the hard work you did!

Tasks

Task W0: Choose a problem for which the naive algorithm has a runtime that is sensitive to its input, even among those of a given size. Call this *Problem P*. For instance, we discussed modular exponentiation in lecture. Using naive square-and-multiply, the runtime to solve this problem varies with the value of the exponent, even among, say, 512-bit exponents. To start your writeup, formally state Problem P, including the expected inputs and outputs.

Task W1: Explain how Problem P is related to privacy. What role does it (or might it) play in systems trusted with securing user privacy? You may speak in the abstract, e.g., "Matrix multiplication is used in the Hill cipher, a classical substitution cipher considered weak today. Although this cipher isn't in use today, matrix multiplication can contribute to diffusion, so it could be used to encrypt information if used in combination with non-linear operations."

Task C2: Implement the naive algorithm to solve Problem P. Call this *Algorithm A*.

Task C3: Implement a test of Algorithm A for different inputs of the same size. Your goal in this task is to show the timing differences for various inputs. You may need to carefully choose your inputs to highlight the desired effect. You may also need to take multiple samples for each input in order to demonstrate your points.

Task W4: Interpret the results of Task C3. Explain why Algorithm A varies in runtime, even for inputs of the same size. You may refer to line numbers in your code and/or provide pseudocode to make this easier. (If using pseudocode, it must accurately model your submitted code.) Provide figures (graphs, charts, etc.) to demonstrate the results of your testing, and explain how these figures show the effect you described. Next, explain how this might reveal private information. For instance, consider questions like the following: In what environment would the algorithm need to be used in order to mount an attack? What type of attack (e.g., chosen plaintext) needs to be mounted? What measurements would the attacker need to collect? What statistics then need to be computed over these measurements?

Task C5: Implement an algorithm for solving Problem P that is more constant-time than Algorithm A. It does not need to be perfectly constant, but it should seek to mitigate the attacks described in Task W4. Call this *Algorithm B*.

Task C6: Implement a test of Algorithm B for different inputs of the same size, similar to what you did for Algorithm A in Task C3. Your goal in this task is to show that Algorithm B mitigates (but may not completely eliminate) the issues described in Task W4. As before, careful choice of inputs and/or multiple trials may be needed.

Task W7: Interpret the results of Task C6. Overview the techniques used by Algorithm B to hide or disguise the information leaked by Algorithm A. Refer to specific lines of your code and/or use pseudocode that matches your code, as appropriate to aid your discussion. Evaluate the tradeoffs Algorithm B makes relative to Algorithm A to accomplish the constant time property: Does it have longer runtime on average, or use more memory, or require assumptions (e.g., secure random values) that aren't present in Algorithm A?

Task W8: Discuss any remaining features of Algorithm B that may still leak information. How could an attacker mount a timing attack against your improved algorithm? If you believe a timing attack to be impossible, discuss other side-channel attacks that an attacker could carry out.

Problem / Algorithm Choice

Up to 10 bonus points will be awarded for choosing an appropriate Problem P and Algorithm B. More points will be awarded for selections that more thoughtful, interesting, creative, clever, etc. Note that you may consider a side-channel other than timing, but you must be able to quantitatively measure the differences in leakage between Algorithms A and B (which may be difficult without special equipment, e.g., for power analysis attacks).

If you are not interested in bonus points and instead would like to pursue a more guided version of this project, you may use the following default options.

- Let Problem P be modular exponentiation on integers represented as large byte arrays (**not** primitive `int` s, which are too small to be effective for cryptography).
- Let Algorithm A be naive square-and-multiply with grade-school multiplication.
- For Algorithm B, consider Montgomery multiplication, blinding as described by Kocher, Montgomery powering ladder, and/or the techniques described in this report to devise your strategy. A thoughtful Algorithm B may be awarded partial bonus points, even if you choose the default Problem P and Algorithm A.

Note that, even if you use the default suggestion, you must complete each task listed above to demonstrate that you understand the problem's relationship to privacy, cryptography, information leakage, etc.

Grading

Task	Points
Task W0	10 (bonus)
Task W1	10
Task C2	20
Task C3	5
Task W4	15
Task C5	25
Task C6	5
Task W7	15
Task W8	5
Total	**Up to 110**

Note

As this course is an upper-level elective, you are being given a lot of freedom in terms of how you tackle this project. In exchange, you also have a lot of responsibility to demonstrate your hard work adequately to the TA. As such, there are tasks in this assignment that require you to discuss your code in detail. Your discussion

should closely align to, and refer to, your specific implementation. You may use any programming language and library functions that you prefer, but you must ensure that your choice of language/library does not “do the work for you.” This is a 2-week project in a senior-level course; if you satisfy the requirements in only 4 lines of code, you’re probably using a library function that allows you to skip the hardest parts of the assignment. If you need help deciding whether something should be permitted, contact your instructor and/or TA.

Submission

Upload your code and writeup to the Box folder created for you with the name `cs1699-p1-abc123`, where `abc123` is your Pitt username.