University of Michigan, Computer Science and Engineering | Handout 5
EECS 475: Introduction to Cryptography | February 26, 2014
Instructor: Prof. Kevin Fu

Homework 4

This homework is due on *Monday, March 17, 2014* by 10:39:00AM. There are **25 possible points**. N.B.: This is a team homework.

Members of a team should work together to submit a single submission both electronically and on paper. The electronic method is CTools, not email. We recommend performing test submissions early on CTools to avoid losing points at the last minute if you should find your CTools account is not setup properly. You may submit updated code as many times as you wish up until the deadline. We do not accept emailed submissions. Late submissions or submissions that do not follow the strict policies from the syllabus will receive a zero. Please contact the TA well ahead of the deadline if you have a question about these procedures.

Submit each homework[1] problem stapled separately. We will have separate boxes at the front of the lecture hall for submitting each problem. Points will be deducted for solutions that do not follow this process. On the front page of **each** stapled problem, the top right corner must provide the following items:

- Print the discussion section time **to return this graded homework**
- Print **your team number and team name**
- Print the name of **each team member**
- Print the uniqname of **each team member**
- Print the homework-problem number.

Example:

Discussion: Fri 10:30AM
Team: 43 [The Cryptospores]
Name: John P. Doe, Alyssa P. Hacker, Ben Bitdiddle
Uniqname: johnpdoe, aphacker, bbitdiddle
Problem: 4-1

¡If your solution spans multiple pages for a problem, mark the top of each page with your initials. Some problems contain multiple components. For each component of a problem, write a descriptor of the component. Points may be deducted if your TA has problems understanding or reading your solution. In mathematical problems, **show all your work**. If you receive any key insights from someone else or some other resource, you must cite that person or resource.

[1] We will distribute our favorite solution for each problem to the class as the "official" solution—this is your chance to become famous!

Problem 4-1. Team Time (2 pts)

Note: This is due before the final deadline.

Pick a team name and a meme or logo. Set a schedule for regular team meetings. We will give you 1 bonus point if you find a clever a Latin security motto. Post a single entry as a team (anonymously is OK) on Piazza under the "Project" thread by **Wednesday, March 12**. In the posting, include your team name, graphic, and your team number. Do not submit this problem on paper or CTools. Keep it clean.

Problem 4-2. Not so fast, my modular friend (22 pts)

Montgomery reductions allow for fast modular multiplication, especially when the modulus remains static for many multiplications and when the Montgomery residue of a multiplicand can be precomputed and reused. Modular exponentiation produces exactly such an environment. In this problem, you will learn exactly how much faster Montgomery reductions can help the performance of modular multiplications, and when Montgomery multiplication does not help.

Make sure to use avoid all trial division (only bit shifts and the % operator) in Montgomery. Also make sure to use the following version of arithmetic with Montgomery multiplication.

Let $M =$ any odd modulus and $R = 2^n$ where $0 < M < 2^n$ for some positive integer n. It must be the case that $\gcd(M, R) = 1$. For odd moduli, this will remain true. Note that this requirement is relaxed slightly from lecture where we said M must be prime. This new version allows for the use of Montgomery multiplication in RSA where the modulus is not prime.

For readability, we are switching to a tilde rather than an overline to denote M-residues.

There are three major functions, described in "logical value" form as:

- $\tilde{x} = xR \bmod M$ [Montgomery residue]
- $\tilde{z} = \text{MM}(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y}R^{-1} \bmod M$ [Montgomery multiplication]
- $\text{RED}(T) = TR^{-1} \bmod M$ [Montgomery reduction]

Use the corresponding fast algorithm to avoid trial division during the modular reductions in $\text{RED}(T)$ where we require on the input that $0 < T < MR$.

Let $M' = -M^{-1} \bmod R$ [i.e., $RR^{-1} - M'M = 1$]
Let $m = TM' \bmod R$ [Make this fast with bit shifts]
Let $t = (T + mM)/R$ [N.B.: integer arithmetic without modular reduction!]
If $t \geq M$, return $t - M$. Else return t

There are multiple ways to correctly implement the MM function, and the better ones will minimize modular reductions in favor of addition and subtraction when possible.

You'll want to design stateful algorithms that cache some of these precomputed values. For instance, you'll likely reuse M', R, R^{-1} across multiple calls to $\text{RED}(T)$.

(a) Implementation

Implement four versions of the **iterative** version (not recursive version) of modular exponentiation (see Stinson page 177). You may reuse code written by your teammates for previous EECS475 homework. The four versions are: the original `modexp` algorithm, `modexp` using the Chinese Remainder Theorem, original `modexp` algorithm using Montgomery residues and Montgomery reductions, `modexp` using Montgomery residues and Montgomery reductions and the Chinese Remainder Theorem. Submit your code and prove that the four algorithms produce consistent results on the following test cases where $n = 881 * 883$ and $a = 13$.

- $a^{1023} \bmod n$
- $a^{1024} \bmod n$
- $a^{1025} \bmod n$

Note that 881 and 883 are both primes[2].

(b) Performance

Run performance tests of your four algorithms on various modulus lengths (picking appropriately sized primes). For each of the modulus sizes you pick (we suggest a binary search to get started), run at least 3 trials for each algorithm where a trial means randomly selecting a base and exponent and performing `modexp`. Graph the performance with a line chart. The x-axis should be the bit length of the modulus, and the y-axis should be the running times of the four `modexp` algorithms running on the same input values. Pick a range of bit lengths such that you see "interesting" results that show crossover points in performance.

Write down the processor, operating system, and other relevant information about the computer that may impact reproducibility. The TAs will release a template Google spreadsheet for you to modify and resubmit.

Submit this information, your Google spreadsheet, and a graphic of performance.

(c) Good better best

Under which conditions does each algorithm perform the best? Why? How might your choice of processor or programming language affect the results?

Problem 4-3. Team management (1 pt)

Describe the roles and contributions from each team member for this homework assignment.

Problem 4-4. Goto FAIL (1 pt extra credit)

Read up on the man-in-the-middle attack on the Apple usage of OpenSSL. For example, read `http://www.wired.com/threatlevel/2014/02/gotofail/` or

`http://blog.secure-medicine.org/2014/02/an-apple-security-flaw-day-keeps-doctor.html`

Explain in your own words how and why the `SSLVerifySignedServerKeyExchange` function fails to do what a cryptographer would expect. What processes and practices might catch these types of flaws? One extra credit point for an insightful and concise description.

[2] AKA twin primes