**CS 6262: Network Security** March 2, 2009

# Problem Set 2: Crypto and Sockets

*Instructor: Prof. Nick Feamster* *College of Computing, Georgia Tech*

This problem set has three questions, each with several parts. Answer them as clearly and concisely as possible. You may discuss ideas with others in the class, but your solutions and presentation must be your own. Do not look at anyone else's solutions or copy them from anywhere. (Please refer to the Georgia Tech honor code, posted on the course Web site).

Turn in your solutions in on **March 16, 2009** by 11:59 p.m. -0500, to the grader.

Note: Please use either C or C++ for Q2 and Q3, and make sure that it compiles using a recent version of gcc/g++. Apart from correctness of the result, the running time of your programs in Q2 and Q3 (part 1) also affects your grade. The grade for a program, 'g' will be the sum of three components:

- 'i' - for your implementation
- 'c' - functionality (i.e., correctness)
- 't' - running time, a weighted score, with the submission with the best running time receiving the full 't' points.

Please make sure your programs have a batch mode that lets us measure the runtime. *Your code must support the invocations as specified on the class wiki.*
`http://davis.gtnoise.net/wiki/doku.php?id=cs6262:problem_set_2_updates`

## 1 Warm Up

Suppose that you have a hash function $H$, where the probability of collision is $2^{-20}$. You are designing a network protocol that takes the hash of each packet header as inputs. Your goal is to design a protocol so that each packet produces a unique "signature"; every router along a path that hashes that packet should produce the same signature.

1. Which fields in the packet header would you choose as input? Which ones might pose problems for achieving the above goals (think about packet headers that may be modified in-flight)? Hint: You need to select headers that include both unique headers for each packet, and are invariant across paths.
2. Suppose that you have a flow that is $n$ packets. Assuming a uniform hash function, what is the probability that two packets in that flow will produce the same output (i.e., what is the probability that a collision will occur within the first $n$ packets). Express the collision probability as a function of $n$.
3. To reduce the probability of collision, you might consider producing the packet signature using $k$ independent uniform hash functions (as is done with a Bloom filter). Given $k$ independent hash functions, what is the probability that two packets in an $n$-packet stream will produce the same signature?

## 2 Fast Exponentiation

Your implementation for this problem should be in C (or C++).

Implement a program that computes exponents efficiently, using the mechanism described in the textbook. Your program should be able to deal with exponents of up to 32 bits and modulo value of up to 32 bits. Make use of the long long 64-bit arithmetic found on most modern machines. Your program should take base, exponent and modulus as arguments, such as

```
exp 45678 123 6789
```

Please print out the running time of your operations.

## 3 Ciphers

Your implementation for this problem should be in C (or C++).

1. Implement one round of DES and one round of RC5 (for RC5, use a 64-bit key and 64-bit blocks). Turn in the source code for your implementation of each stream cipher, including compilation and running instructions.
2. Use a profiler to examine the complexity of each of the operations in the stream cipher (e.g., substitution, swapping, etc.). How much time is spent in each of the operations, for each of the stream ciphers? For this part of the problem, please include a table that breaks down the operation of the ciphers, with a percentage of CPU time spent in each part of the process.
3. One of the major design motivations for DES, in particular, was that certain operations could be performed more efficiently in hardware than in software. In the case of DES, briefly explain which operations could be made more efficient in hardware, and why they could be more efficient in hardware than in software (Hint: think about parallelization, for example.)

## 4 OpenSSL Programming

In this problem, you will implement a public-key-based secure channel using OpenSSL. For extra credit, you can perform some extensions to the sockets layer itself.

1. Download and install the openssl library.
2. Write a simple client and server that uses RSA public key encryption between the client and server.
3. Modify your client and server to use an elliptic-curve based public-key algorithm (this should be a small modification). Note that the size of the public keys are much smaller in this case. What advantages can you think of for using smaller keys?

4. *Extra credit:* Recall that, in lecture, we discussed the benefits of self-certifying protocols over standard key distribution mechanisms. Modify your client/server setup so that the client can communicate with the server using self-certifying techniques.

   *Hint:* You might do this in a number of ways. You could modify the lookup service, hack the IP layer and use proxies, etc. Be creative, and state your assumptions.

## Quiz Review Topics

For your benefit, here's a rough list of topics that should help you in studying for the quiz. This is not necessarily a complete list, but it should serve as a fairly good starting point for studying.

1. General Security Concepts
   - definitions: confidentiality, authentication, authorization, etc.
   - threat taxonomies and attack models
2. Cryptography
   - secret key ciphers: substitution, permutation, combinations thereof
   - historical secret key ciphers
   - security arguments for one-time pad
   - block and stream ciphers
   - public key algorithms: Diffie-Hellman, RSA
   - fast exponetiation techniques
   - operation of RSA public key algorithm
   - types of attacks: dictionary, chosen plaintext, known plaintext, etc.
   - attacks on RSA (those covered in lecture)
3. Key Management
   - key distribution centers
   - public key infrastructures: design, drawbacks, etc.
   - Kerberos: roughly how it works
4. Authentication and Access Control
   - Lattice models
   - access control: matrices, capabilities, ACLs
   - taint propagation
   - static vs. dynamic tainting
5. System Security
   - Why crypotsystems fail
   - Secure fault localization
6. Malware and Worms
   - buffer overflow attacks: stack smashing, off-by-one, format string
   - SQL injection attacks
   - work propagation: models, scanning strategies, etc.