# Homework 01

*Due: 11:59 pm, Monday February 5*

Please handin your homework assignment on gradescope as a PDF file with each problem on a separate page.

When asked to provide examples, do not repeat the examples given in the lecture or the homework itself, and instead come up with something original.

## Introductory Question

### Problem 1: Trade-offs

At the heart of security is the idea of trade-offs: the balance between the benefits of security on the one hand, and the costs of it on the other hand. These costs can be anything—money, time, convenience, etc. Getting this trade-off right can be tricky.

**Question a)** Describe a security system you've encountered or heard about which successfully mitigates certain threats, but does so at a cost which you deem to be too high given the risks. Why do you think the cost is too high? (Write a paragraph or two.)

**Question b)** Describe a security system you've encountered or heard about which might fail in the face of certain threats, but where you think that the cost of mitigating these threats would not be worth the increased security. Why do you think the extra security wouldn't be worth it? (Write a paragraph or two.)

## Cryptography

### Problem 2: Entropy

The strength of a cryptographic key is often measured in the key's length in bits. However, this can be somewhat misleading. What we really want to know is: what is the entropy of the key, or, more precisely, what is the entropy of the process that generates keys? Recall from lecture that, for the uniform distribution, the entropy is simply the base-2 logarithm of the number of possible outcomes. For example, if we have uniformly-generated keys which are 512 bits long, but only one out of every four possible 512-bit sequences is a valid key, then there are actually only $2^{510}$ possible key values. Thus, a 512-bit key generated with the above process has entropy of 510 bits. More generally, if only $m$ possible bit sequences are valid keys, then the entropy is $\log_2 m$ bits.

What is the entropy of a 512-bit AES key under each of the following scenarios? Please justify your answers and show your work.

**Question a)** The key is generated by choosing a 512-bit string uniformly at random from the set of all possible 512-bit sequences.

**Question b)** The key is produced by a pseudorandom number generator (PRNG) whose seed is a 256-bit string. You may assume that the seed of the PRNG is generated uniformly at random and comes from a source of true randomness.

**Question c)** The key is generated as in (b), except that this time the PRNG is seeded using the cryptographic hash of a user-supplied password. That is, the user-supplied password is hashed using a cryptographic hash function whose output is 256 bits long. Then the hash is used as the seed to the PRNG. The key is then generated as above. Assume that the entropy of user-supplied passwords is 40 bits.[1]

**Question d)** The key is generated as in (b), except that this time the PRNG is seeded using the current time in microseconds since the unix epoch (January 1, 1970, 00:00:00 UTC). You may assume that the time at which the key is generated is publicly broadcasted to within 10-second accuracy (i.e., +/- 10 seconds). What is the entropy of such a key given the broadcasted information?

**Question e)** The key is generated as in (b), except that this time the seed of the PRNG is biased. Specifically, the probability that a given bit of the seed is a 1 is 51%. For this subproblem only, don't actually compute the entropy, but answer the following question: is the entropy of the generated key in this case higher than, lower than, or equal to the entropy in case (b). Why?

## Problem 3: Cryptography Performance

Different cryptographic primitives have different strengths and weaknesses. Symmetric key cryptography is really fast but it assumes that the party that encrypts (Alice) and the party that decrypts (Bob) share the same secret key. On the other hand, public key cryptography does not require the two parties to share a secret key but is much slower. Here are performance numbers from the Crypto++ 5.6.0 Benchmark,[2]which was executed on an Intel Core 2 1.83 GHz machine with 32-bit Windows Vista:

| Operation | MB/Second |
|---|---|
| AES/ECB Encryption (128-bit key) | 114 |
| AES/ECB Decryption (128-bit key) | 114 |

AES Symmetric Key Cryptosystem

| Operation | Millseconds/Operation |
|---|---|
| RSA 2048 Encryption | 0.16 |
| RSA 2048 Decryption | 6.08 |

RSA Public Key Cryptosystem

The table on the left lists the number of MB (1MB = 1,000,000 bytes) that can be encrypted or decrypted per second using the AES symmetric key cryptosystem (ECB, or "electronic codebook", just means that these measurements were taken by splitting the data up into 128-bit chunks and encrypting each chunk individually[3]). The table on the right lists the number of milliseconds required to perform a single encryption or decryption operation using the RSA public key cryptosystem. In this case, RSA is configured to operate on 2048-bit (256-byte) messages.

**Question a)** Using the performance numbers from the Crypto++ 5.6.0 Benchmark, how many seconds would it take to encrypt and then decrypt 1GB (1B bytes) of data using:

i) AES/ECB

ii) RSA 2048[4]

**Question b)** What is the ratio of the larger to the smaller of the two running times above?

[1] A 2007 study (`http://research.microsoft.com/pubs/74164/www2007.pdf`) found that the entropy of passwords used on the Web is 40.54 bits.

[2] `https://www.cryptopp.com/benchmarks.html`

[3] For the curious, see Section 8.1.7 or the textbook for more details on the different ways that block ciphers (symmetric encryption/decryption functions that work on fixed-size blocks of data at a time) can be combined to encrypt plaintexts which are larger than the block size.

[4] For the sake of this question you may assume that the data is processed in batches of 256 bytes. Note, however, that this is just a simplifying assumption for the sake of the homework — this is not actually done in practice.

## Problem 4: Hybrid Cryptosystems

Public key cryptography allows parties who have not pre-shared a (symmetric) secret key to communicate. However, as we saw in the previous problem, encryption and decryption in public key cryptographic schemes is often very slow. Consider the following hybrid scheme which combines both public key and symmetric key cryptography.

As usual, we have two parties, Alice and Bob, who wish to exchange confidential messages. They use an insecure communication channel where an attacker, Eve, is capable of eavesdropping on messages but cannot modify them.

Alice and Bob have agreed on using a public-key cryptosystem and a symmetric cryptosystem with specific parameters (e.g., RSA 2048 and AES256). Alice has a public key/private key pair, $(PK_A, SK_A)$. Bob has a public key/private key pair, $(PK_B, SK_B)$. Alice and Bob each know one another's public keys. To protect the confidentiality of their communication, Alice and Bob use a protocol that consists of a setup phase and a chat phase.

The setup phase is as follows:

1. Alice randomly generates a symmetric key, $k_A$, encrypts it with Bob's public key, and sends the resulting ciphertext as a payload to Bob ($k_A$ is used here as a protocol message).
2. Bob decrypts the ciphertext using his private key, and learns $k_A$.
3. Bob randomly generates a symmetric key, $k_B$, encrypts it with Alice's public key, and sends the resulting ciphertext as a payload to Alice ($k_B$ is used here as a protocol message).
4. Alice decrypts the ciphertext using her private key, and learns $k_B$.

At the end of the setup phase, Alice and Bob have exchanged symmetric keys, $k_A$ and $k_B$.

In the chat phase, Alice and Bob send to each oteher symmetrically encrypted messages. Whenever Alice wants to send a chat message to Bob, she encrypts it with $k_B$ using the symmetric cipher and sends the resulting ciphertext. When Bob wants to send a message to Alice, he encrypts it with $k_A$ using the symmetric cipher and sends the resulting ciphertext to Alice.

**Question a)** Explain why the above protocol ensures confidentiality, that is, Eve cannot learn the plaintexts of the chat messages exchanged between Alice and Bob.

**Question b)** While Eve cannot learn plaintex chat messages, she can match them, that is, she can determine whether two ciphertext chat mesages sent by Alice, or by Bob, correspond to the same plaintext. Show how the chat protocol can be modified in a simple and efficient manner to prevent Eve from matching chat messages.