Midterm Exam Solutions

CS161 Computer Security, Spring 2008

1.

To encrypt a series of plaintext blocks $p_1, p_2, \ldots p_n$ using a block cipher E operating in electronic code book (ECB) mode, each ciphertext block $c_1, c_2, \ldots c_n$ is computed as $c_i = \mathrm{E}_k(p_i)$.

Which of the following **is not** a property of this block cipher mode?

(a) Any repeated plaintext blocks will result in identical corresponding ciphertext blocks.
(b) Decryption can be fully parallelized.
(c) If a ciphertext block is modified or corrupted, then after decryption the corresponding plaintext block and all the following plaintext blocks will be affected.
(d) None of the above; that is, (a), (b), and (c) are all properties of the ECB block cipher mode.

Answer: The correct answer is (c). In ECB, altering a ciphertext block only affects a single plaintext block.

2.

To encrypt a series of plaintext blocks $p_1, p_2, \ldots p_n$ using a block cipher E operating in cipher block chaining (CBC) mode, each ciphertext block $c_1, c_2, \ldots c_n$ is computed as $c_i = \mathrm{E}_k(p_i \oplus c_{i-1})$, where c_0 is a public initialization vector (IV) which should be different for each encryption session.

Which of the following **is** a property of this block cipher mode?

(a) Any repeated plaintext blocks will result in identical corresponding ciphertext blocks.
(b) Decryption can be fully parallelized.
(c) If a ciphertext block is modified or corrupted, then after decryption the corresponding plaintext block and all the following plaintext blocks will be affected.
(d) None of the above; that is, neither (a), (b), nor (c) are properties of the CBC block cipher mode.

Answer: The correct answer is (b). Each plaintext block can be computed using only two ciphertext blocks, independent of the other plaintext blocks: $p_i = \mathrm{D}_k(c_i) \oplus c_{i-1}$.

Note that (c) is not a property of CBC. A modification to a ciphertext block will affect that plaintext block and the one immediately following it, but none after that.

3.

Consider a k-bit hash function $h : \{0,1\}^* \to \{0,1\}^k$. Assume h operates ideally in the sense that each distinct input to h is mapped to a random member of $\{0,1\}^k$. Assume an attacker is trying to finding a collision of h, that is, any two $x_1, x_2 \in \{0,1\}^*$ such that $h(x_1) = h(x_2)$. How does the expected number of tries (evaluations of h) before the attacker succeeds grow with respect to k?

(a) $\Omega(2^k)$
(b) $\Omega(2^{\sqrt{k}})$
(c) $\Omega(2^{k/2})$
(d) $\Omega(2^{\log k})$

Answer: The correct answer is (c), $\Omega(2^{k/2})$, i.e., $\Omega(\sqrt{2^k})$.

4.

The Diffie-Hellman protocol is used to generate a shared secret key between two parties using a public channel. It proceeds as follows.

> Let p be a large prime and g be a generator of $\mathbb{Z}_p^*$; both are publicly known parameters. Alice selects a random $a \in \mathbb{Z}_p$ and sends $x = g^a \mod p$ to Bob. Bob selects a random $b \in \mathbb{Z}_p$ and sends $y = g^b \mod p$ to Alice. The shared key is $g^{ab} \mod p$, which Alice may compute as $y^a \mod p$ and Bob may compute as $x^b \mod p$. A message $m \in \mathbb{Z}_p^*$ may be encrypted using this key as $c = m \cdot g^{ab} \mod p$.

Which of the following public key encryption and digital signature schemes is most similar to the Diffie-Hellman protocol?

(a) RSA encryption.
(b) RSA signatures.
(c) ElGamal encryption.
(d) ElGamal signatures.

Answer: The correct answer is (c). Diffie-Hellman and ElGamal encryption are exactly the same operations used in a somewhat different context.

More precisely, suppose Alice generates an ElGamal public key and sends it to Bob, then Bob encrypts a message under that key and sends the ciphertext to Alice. Then Alice and Bob have computed and communicated exactly the same values they would have if they performed a Diffie-Hellman key exchange then sent the message using the shared key, as described above.

5.

Alice knows that she will want to send a single 128-bit message to Bob at some point in the future. To prepare, Alice and Bob first select a 128-bit key $k \in \{0,1\}^{128}$ uniformly at random.

When the time comes to send a message $x \in \{0,1\}^{128}$ to Bob, Alice considers two ways of doing so. She can use the key as a one time pad, sending Bob $k \oplus x$. Alternatively, she can use AES to encrypt x. Recall that AES is a 128-bit block cipher which can use a 128-bit key, so in this case she would encrypt x as a single block and send Bob $\text{AES}_k(x)$.

Assume Eve will see either $k \oplus x$ or $\text{AES}_k(x)$, that Eve knows an initial portion of x (a standard header), and that she wishes to recover the remaining portion of x.

If Eve is an all powerful adversary and has time to try out **every possible key** $k \in \{0,1\}^{128}$, which scheme would be more secure?

(a) The one time pad would be more secure. Even if Eve tried all possible keys, she would not be able to recover the unknown portion of x. If AES was used, Eve could eventually learn the unknown portion of x.
(b) AES would be more secure. Even if Eve tried all possible keys, she would not be able to recover the unknown portion of x. If the one time pad was used, Eve could eventually learn the unknown portion of x.
(c) They would be equally secure. Either way, Eve could eventually learn the unknown portion of x.
(d) They would be equally secure. Either way, Eve would not be able to learn the unknown portion of x.

Answer: The correct answer is (d). Even after trying every possible key (including the actual one), Eve will have no way of recognizing the correct plaintext or even narrowing down the possibilities in any way.

Why is this? Well, since AES is a distinct permutation on $\{0,1\}^{128}$ under each possible key, and the key was selected uniformly at random, given any plaintext, each possible ciphertext is equally likely. So when AES is used for a single block with a random key of the same length, the effect is exactly the same as using a one time pad: the ciphertext reveals no information about the plaintext.

6.

Message authentication codes (MAC) and digital signatures both serve to authenticate the content of a message. Which of the following best describes how they differ?

(a) A MAC can be verified based only on the message, but a digital signature can only be verified with the secret key used to sign the message.
(b) A MAC can be verified based only on the message, but a digital signature can only be verified with the public key of the party that signed the message.
(c) A MAC can only be verified with the secret key used to generate it, but a digital signature can be verified based only on the message.
(d) A MAC can only be verified with the secret key used to generate it, but a digital signature can be verified with the public key of the party that signed the message.

Answer: The correct answer is (d).

7.

Let p be a large prime and g be a generator of $\mathbb{Z}_p^*$.

The *discrete logarithm problem* is the task of computing a given $g^a \mod p$, where a is an exponent randomly selected from $\mathbb{Z}_p$.

The *computational Diffie-Hellman problem* is the task of computing $g^{ab} \mod p$ given $g^a \mod p$ and $g^b \mod p$, where a and b are exponents randomly selected from $\mathbb{Z}_p$.

Which of the following best describes the relationship between these problems?

(a) They are equivalent; if either can be efficiently solved, the other can.
(b) If the computational Diffie-Hellman problem can be efficiently solved then the discrete logarithm problem can be efficiently solved, but the converse is not known to be true.
(c) If the discrete logarithm problem can be efficiently solved then the computational Diffie-Hellman problem can be efficiently solved, but the converse is not known to be true.
(d) None of the above.

Answer: The correct answer is (c). To show that the computational Diffie-Hellman problem reduces to the discrete logarithm problem, imagine you have an algorithm to efficiently compute discrete logs and you are given the task of solving the Diffie-Hellman problem. Then you could easily compute a from $g^a \mod p$ and then compute $(g^b)^a \mod p = g^{ab} \mod p$. No reduction in the other direction is known.

8.

Let p be a large prime and g be a generator of $\mathbb{Z}_p^*$.

Suppose we are considering the function $h : \mathbb{Z} \to \mathbb{Z}_p^*$ for use as a hash function, where $h(m) = g^m \mod p$.

Four basic properties are typically desired of cryptographic hash functions. The *compression* property requires that messages of any length be hashed to a finite domain. The *preimage resistance* (a.k.a. one-way) property requires that it be hard to find a message that hashes to a particular value. The *second preimage resistance* (a.k.a. weak collision resistance) property requires that, given one message, it is hard to find a second message with the same hash as the first message. The *collision resistance* (a.k.a. strong collision resistance) property requires that it be hard to find any two messages with the same hash.

Since we treat messages as arbitrary integers (not just members of $\mathbb{Z}_p^*$), h satisfies the compression property.

If we assume the difficulty of the discrete logarithm problem in $\mathbb{Z}_p^*$, which of the other three properties does h satisfy?

(a) All of them: preimage resistance, second preimage resistance, and collision resistance.
(b) Only preimage resistance and second preimage resistance.
(c) Only preimage resistance.
(d) None of them.

Answer: The correct answer is (c).

It can be shown that h satisfies preimage resistance with a reduction from the discrete logarithm problem. The reduction is trivial in that they are almost exactly the same problem. If you have an algorithm which can produce preimages, you need only reduce them modulo p to produce the correct answer for the discrete logarithm problem.

To see that it is not second preimage resistant, note that for any message m, the message $m + p - 1$ will hash to the same value (and $m + 2(p-1), m + 3(p-1), \ldots$). And if it is not second preimage resistant, there is no way it can be collision resistant, because that is a strictly stronger condition.

9.

The following protocol is used to establish a shared key K_{ab} between two parties A and B, assuming A and B each share a key with a mutually trusted server S.

1. $A \rightarrow S : n_a, A, B$
2. $S \rightarrow B : E_{K_{as}}(n_a, A, B, K_{ab}), E_{K_{bs}}(n_a, A, B, K_{ab})$
3. $B \rightarrow A : E_{K_{as}}(n_a, A, B, K_{ab}), E_{K_{ab}}(n_a), n_b$
4. $A \rightarrow B : E_{K_{ab}}(n_b)$

The values n_a and n_b are nonces selected by A and B. Like the Needham Schroeder protocol, this protocol is vulnerable to a key freshness attack.

More specifically, assume an eavesdropper records the messages above in one execution of the protocol, then at some later point manages to compromise the session key K_{ab}. With this information, an active adversary can then trick B into reusing K_{ab} in a session with the attacker. B will mistakenly believe it is communicating with A and that K_{ab} is a fresh key generated for the two of them by S.

Show how this may be done.

Assume the adversary can arbitrarily intercept messages and drop or modify them before the intended recipient sees them. The adversary can also send new or replayed messages to any party, making them appear to come from any other party. However, the adversary has not compromised either of the long-lived keys (K_{as} and K_{bs}).

Answer: The following is the most straightforward way of accomplishing this attack. Assume the adversary has already observed one run of the protocol and subsequently compromised K_{ab} somehow.

The adversary replays message 2 to B, making it appear to come from S. B (who maintains no state between executions of the protocol, as was clarified during the exam) thinks that A is once again trying to initiate a session with it and that S has generated K_{ab} for them.

B then sends a new message 3 to A; it differs from the previous message 3 only in B's choice of nonce: $E_{K_{as}}(n_a, A, B, K_{ab}), E_{K_{ab}}(n_a), n'_b$.

The adversary intercepts this message before it reaches A and replies to B with a message $E_{K_{ab}}(n'_b)$, making it appear to come from A. Note that the adversary can compute $E_{K_{ab}}(n'_b)$ because it has K_{ab} and has just observed n'_b. Now the adversary and B can continue communicating, and B will mistakenly believe it has a secure session with A.

10.

An n out of n secret sharing scheme is a randomized algorithm which takes a secret string x and produces n shares $s_1, s_2, \ldots s_n$. The shares must have the property that any set of $n-1$ or fewer shares reveals no information about x, but all n shares completely determine x.

Show how to construct an n out of n secret sharing scheme for an ℓ-bit secret $x \in \{0,1\}^\ell$ using the exclusive or ($\oplus$) function. Your answer should specify any random values selected and show how to compute each of the shares $s_1, s_2, \ldots s_n$ based on those values and x.

Answer: The most straightforward solution is the following. Pick $n-1$ random, ℓ-bit values $r_1, r_2, \ldots r_{n-1} \in \{0,1\}^\ell$. Then set

$$\begin{aligned} s_1 &= r_1 \\ s_2 &= r_2 \\ &\vdots \\ s_{n-1} &= r_{n-1} \\ s_n &= x \oplus r_1 \oplus r_2 \oplus \cdots r_{n-1} \end{aligned}$$

Note that the secret can be reconstructed as $s_1 \oplus s_2 \oplus \cdots s_n = x$.

11.

A program running on Alice's system occasionally needs to pick a random AES key. After reading through the source code for the program, Alice discovered that it does so by calling `srand` to seed the pseudorandom number generator with a combination of the current time and process ID, then repeatedly calling `rand` to generate the bytes of the key.

Alice has heard that this is a very insecure method of selecting a random symmetric key due to the predictability of process ID's and the current time and the flaws of most `rand` implementations. To remedy this situation, Alice sets her computer's clock to a random time and configures her kernel so that process ID's are selected randomly rather than sequentially. Furthermore, she replaces the calls to `srand` / `rand` with a SHA-1 hash, truncating the output down to a 128-bit AES key.

The relevant portion of the new source code (which is in C) is given below:

```
int x;
char buf[20];

/* set x to current time (in seconds since the epoch) */
x = time(0);

/* xor in the process ID for more randomness */
x = x ^ getpid();

/* hash x with SHA-1 and put the result in buf */
sha1(buf, &x, 4);

/* now we will use the first 16 bytes of buf as a 128-bit AES key */
```

In this code, `sha1(char* out, char* in, int k)` is a function that computes the 160-bit SHA-1 hash of a `k`-byte message starting at address `in` and places the result at address `out`.

Are Alice's changes sufficient? If you think the new system is reasonably secure, explain why. If you think it is insecure, state how you would go about breaking it.

Answer: Alice's changes are in no way sufficient to secure the system; anything encrypted with a key selected in this way can be decrypted in a matter of minutes.

This is because the SHA-1 hash is computed over only four bytes, resulting in only about four billion possible keys. The most straightforward way to attack this system is to try each of the 2^{32} possible values for x in turn, each time using SHA-1 to hash x then attempting decryption with that key.

This is possible regardless of how Alice has set her computer's clock or how it chooses process ID's. If an attacker can somehow derive or narrow down the possible settings for Alice's clock (e.g., using a separate protocol from the one being attacked), they could speed up the attack somewhat, but it may not be worth bothering since the set of possible keys is so small already.

12.

Suppose we have an undirected graph $G = (N, E)$, where N is a set of nodes and we represent the edges as a subset E of $N \times N$. Since G is undirected, E is a symmetric relation on N. A *3-coloring* of G is a mapping

$$f : N \to \{\text{“red”}, \text{“green”}, \text{“blue”}\}$$

such that

$$(n_1, n_2) \in E \Longrightarrow f(n_1) \neq f(n_2) \quad .$$

Merlin claims to know of a 3-coloring f of G and wants to prove this in zero-knowledge to Arthur (who also has G). Like many other zero-knowledge proofs about graph properties, the protocol they use will take the following general form.

Phase 1 Merlin commits to some information about G and / or his coloring f.

Phase 2 Arthur sends him a random challenge.

Phase 3 Merlin responds, and Arthur checks some property of the response and that it matches the previous commitments, then either accepts or rejects.

The protocol must have the following properties:

Completeness If Merlin is being honest (i.e., he does have such a 3-coloring f) and both he and Arthur follow the protocol, Arthur will always accept.

Soundness If the claim Merlin is making is impossible (i.e., there is no 3-coloring of G), then no matter what Merlin does, if Arthur follows the protocol he will reject with some probability.

Zero-knowledge If Merlin follows the protocol, then no matter what Arthur does, Arthur will not learn anything about Merlin's solution (f).

Efficiency All computations required of Arthur are polynomial time.

For this problem, we modify the soundness requirement slightly to allow the probability of Arthur rejecting (in the case that no 3-coloring exists) to depend on the size of the graph G. So, for example, if the protocol ensures Arthur rejects with probability $\frac{1}{|N|}$ or $\frac{1}{|E|}$, then that is acceptable. Arthur can always repeat the protocol more times for larger graphs to get more assurance.

Design a suitable protocol for this problem by filling in the template on the following page.

Try to specify your protocol concretely without becoming bogged down in the notation and details. If it's clear that you have a correct solution in mind, you won't be penalized for minor mistakes or edge cases. Don't worry about how commitments are implemented; you just can denote a commitment to a string or value s by $c(s)$.

Hints: This problem is much simpler than the "edge flagging" (a.k.a. vertex cover) zero-knowledge graph problem on the second homework. In particular, it's not necessary to permute the names of the nodes of G and make a permuted adjacency matrix. The simplest solution will only take a few lines to specify. When considering the challenge that Arthur will pose in Phase 2, keep in mind the property that a valid 3-coloring must satisfy: given any edge, the colors of the nodes at either end must differ.

Answer: In a couple pages we give the simplest, most straightforward solution, but first we go over some general commentary on this solution and other possible attempts.

In short, the given solution has Merlin randomly permute his coloring, then commit to the color of each node. Arthur challenges Merlin with a particular edge, Merlin reveals the colors of those two nodes, and Arthur checks that they differ. By inspection, we can see that this protocol satisfies our completeness and efficiency requirements.

To see that it is sound, suppose the graph cannot be three colored. Then there is at least one edge such that both nodes have the same color, and Merlin has at least a $\frac{1}{|E|}$ chance of getting caught.

To see that it is zero-knowledge, note that the only information Merlin reveals (other than commitments which are not later opened) is the color of each of two nodes connected by an edge. Call these colors x and y. Since Merlin first applied a random permutation to the colors, x is equally likely to be red, green, or blue, and y is equally likely to be either of the two colors x is not. Arthur could have generated random values x and y with the

same distribution on his own, so his interaction with Merlin does not reveal anything new.

A great many students attempted to solve this problem by having Merlin first permute the names of the nodes, but not the colors. Unfortunately, this almost always results in a protocol which is not zero-knowledge.

To understand why, let us consider one such protocol. As the first step, Merlin randomly selects a permutation $\pi : N \to N$. Then for each edge $e_i \in E$, Merlin commits to the names and colors of the nodes: $c_{i1} = c(n, n'), c_{i2} = c(f(n), f(n'))$, where $e_i = (n, n')$. He also commits to the permutation $c_\pi = c(\pi)$. Then Arthur may challenge him to either open c_π and all the name commitments c_{i1} (in which case he checks that they match the graph), or the color commitment for one edge (in which case he checks that they differ).[1] The latter possibility is what gradually leaks information to Arthur.

Suppose Arthur repeats this protocol a large number of times, obtaining t responses to the second type of challenge. If Arthur totals the number of times he sees each color mentioned as r, g, and b, then he can use those numbers to compute something about the overall coloring. Specifically, as t grows larger, $\frac{r \cdot |E|}{t}$ tends toward the sum of the degrees of the red nodes. This sum and the corresponding sums for blue and green may not be useful for some graphs. For other graphs however, Arthur may be able to easily narrow down the possibilities for the color of each of the original nodes and compute the solution. Note that it is not enough to simply hide the coloring of some graphs; in order for the protocol to be considered zero-knowledge, it must not reveal any information at all, no matter which graph it is used on.

This is just one way of showing this type of protocol leaks information; there may be other approaches to breaking it. Now, on the following page, we give a correct protocol (which is also much simpler than the incorrect one above).

[1] As it stands this protocol also has a problem with soundness. We ignore that here because it can be fixed by adding some more commitments and because we are trying to see why it is not zero-knowledge.

- Phase 1. Your answer for this part should first specify any random values Merlin selects and any other computations he performs. Next it should give a list of one or more commitments $c_1 = c(s_1), c_2 = c(s_2), \ldots c_\ell = c(s_\ell)$ he forms and sends to Arthur.

  Answer: Let $N = \{n_1, n_2, \ldots n_{|N|}\}$. First Merlin selects a random permutation of the three colors $\pi : \{\text{“red”}, \text{“green”}, \text{“blue”}\} \rightarrow \{\text{“red”}, \text{“green”}, \text{“blue”}\}$. Then for each $i \in \{1, 2, \ldots |N|\}$, Merlin computes $s_i = \pi(f(n_i))$ and sends the commitment $c_i = c(s_i)$ to Arthur.

- Phase 2. For this part, specify the set from which Arthur selects a random challenge to be sent to Merlin. Example sets from which to draw the challenge include $\{0, 1\}$ (a single coin flip), E (an edge), $N \times N$ (a pair of nodes), etc.

  Answer: Arthur challenges Merlin with an edge $(n_i, n_j) \in E$.

- Phase 3. For this part, first specify one or more of the original commitments $c_1, c_2, \ldots c_\ell$ which Merlin opens in response to the challenge. Assume “opening” a commitment means that both the value committed to and the randomness used in forming the commitment are sent to Arthur, and that Arthur checks that they match the corresponding commitment from Phase 1. Next specify any additional checks Arthur performs on the values revealed.

  Answer: In response to the challenge $(n_i, n_j) \in E$, Merlin opens the two commitments c_i and c_j, revealing $s_i = \pi(f(n_i))$ and $s_j = \pi(f(n_j))$ (and the corresponding randomness). Arthur checks that these values match the commitments c_i and c_j and that $s_i \neq s_j$.