

Midterm Quiz Solutions

PLEASE READ ALL THE INSTRUCTIONS

These are the solutions to the midterm quiz.

Problem Q-1. On-Line Gambling

In class, we discussed a fair coin flipping protocol (see lecture 11). In it, Alice and Bob successfully agree on a random bit using a secure bit-commitment scheme.

Secure On-Line Gambling Company (tm) is planning on setting up a web-casino, and would like to have protocols that are secure and trustworthy.

The first game system they wish to implement is Gin. In it, a dealer deals cards to participants (including herself) face down, and game proceeds, with participants drawing more cards as they reveal sets of cards which follow certain patterns (or discard). To ensure that nobody cheats, anyone caught cheating will lose the game by default.

First we design our own bit-agreement protocol. We need a system with the following property:

$$D_{K_1}(E_{K_2}(E_{K_1}(M))) = E_{K_2}(M)$$

That is, encryption/decryption is commutative, so we can encrypt and decrypt out of order, as long as we encrypt and decrypt with the appropriate keys the right number of times.

RSA has this property when the participants use a common modulus. That is:

$$\begin{aligned} n &= pq \\ d_1 &= e_1^{-1} \pmod{\phi(n)} \\ d_2 &= e_2^{-1} \pmod{\phi(n)} \\ D_{K_1}(E_{K_2}(E_{K_1}(M))) &= (((M)^{e_1})^{e_2})^{d_1} \pmod{n} \\ &= M^{e_1 e_2 d_1} \pmod{n} \\ &= M^{(e_1 d_1) e_2} \pmod{n} \\ &= M^{e_2} \pmod{n} \\ &= E_{K_2}(M) \end{aligned}$$

This does, however, mean that n, p, q are known by all participants, and so knowing either of d or e is enough to know the other part of the key.

(a) The protocol for bit agreement is the following:

- Alice and Bob generate an encryption/decryption key pair (with a set pre-agreed modulus) but do NOT send either of the two over (i.e. the public key is still kept private).
- Alice creates two messages, one indicating “heads” and the other indicating “tails”. She sends Bob (in some random order) the encryption of both: $E_A(M_1), E_A(M_0)$
- Bob picks a message at random (he can’t decrypt), encrypts it, and sends it to Alice: $E_B(E_A(M_x))$.

- Alice decrypts it using her private key, to get one of the two messages encrypted under Bob's key (she can't tell which): $D_A(E_B(E_A(M_x))) = E_B(M_x)$. She sends it to Bob.
- Bob decrypts the message with his private key to decrypt the message, and reveal the result of the coin flip. He sends the message to Alice.
- Alice verifies the message is indeed correct (she may have put some unique string in there that Bob couldn't have guessed).
- Both now reveal their keys to prove they did not cheat somehow, and check that all messages were computed correctly.

This protocol is obviously not limited to exchanging a “heads” or “tails”.

Explain how, by modifying this protocol slightly, Bob can choose and look at ten of fifty-two strings that Alice has to offer, without letting Alice know which he has chosen.

- (b) How can Bob ask for more cards (strings) and look at them without Alice finding out what cards he has?
- (c) Alice also needs to deal cards to herself. Obviously we can't have Alice choosing these cards, since she knows what they are and can pick. Explain how we can have Bob deal her cards in a way that she can believe is fair (i.e. Bob can't control which cards she will have), and such that they play with a real deck (i.e. there are 52 cards in the game, and no card appears twice). Work this into the the protocol so that Alice and Bob can have a hand dealt to them.
- (d) Can Alice ask for more cards? How?

- (a) The protocol is similar to the bit-flipping
- Alice and Bob generate an encryption/decryption key pair (with a set pre-agreed modulus) but do NOT send either of the two over (i.e. the public key is still kept private).
 - Alice creates fifty-two messages, each corresponding to a particular card (plus some random string). She sends Bob (in some random order) the encryption of all cards: $E_A(M_1), E_A(M_2), \dots, E_A(M_{52})$
 - Bob picks a ten messages at random (he can't decrypt), encrypts them, and sends them to Alice. That will be the hand he is “dealt”: $E_B(E_A(M_1)), E_B(E_A(M_3)), \dots$
 - Alice decrypts them using her private key, to get ten messages encrypted under Bob's key (she can't tell which they are): $D_A(E_B(E_A(M_x))) = E_B(M_x)$. She sends the ten cards back to Bob.
 - Bob decrypts the messages with his private and reveal his hand. He *does not* send his hand to Alice.

Bob now has ten cards. At the end of the game, the keys must be revealed so that all the messages can be verified to be correct. Since Bob only sees encryptions of the cards until they are revealed, and since Alice can't tell which cards Bob has passed her, this scheme works.

- (b) To deal more cards to Bob, he merely selects more cards from the remaining encryptions of the cards, encrypts them with his key, and sends them over to Alice. Alice will decrypt them, send them over, and Bob will decrypt to find out what cards he got.
- (c) Instead of letting Alice deal cards, we simply let Bob choose another ten cards from the remaining set which he will hand to Alice (without encrypting). She will decrypt and discover which cards she has. Since Bob could not decrypt those cards, he has no knowledge which cards he gave her, and so she can trust them to be random.
- (d) Again, dealing more cards to Alice is easy. Simply let Bob send more cards to Alice when she wants them. For the same reasons as above, this is a fair deal.

Note: In reality, it turns out that using some cryptosystems (such as RSA), still allows one to cheat slightly if you let Alice select her own cards (some solutions suggested Bob encrypt all the cards he has, send them to Alice, and then let Alice choose which ones she wants and send them to Bob for decryption). Alice can mark some cards. She does this by relying on the fact that quadratic residues (i.e. squares), remain squares when raised to any power, and non-residues only become squares when raised to an even power.

She can mark the cards by simply setting all the cards she wants to be quadratic residues, and all the ones she doesn't as non-residues. This can be avoided in several ways, but is problematic otherwise.

Problem Q-2. Timestamping

A timestamping service provides users with the ability to place a “signature” on a document which indicates the time it was seen by the service provider. This service can be useful in proving that a patent is invalid due to prior art, or that a paper was written before the deadline (and yes, the student really was sick and couldn’t come to class). There have been several proposed systems that achieve some form of timestamping.

List five different requirements that are desirable in such a service. Explain what they mean, and briefly indicate one potential approach for achieving each requirement.

1. **Easy Verifiability** is necessary so that the receiver of the timestamp can believe in the validity of his timestamp instantly, and can demonstrate the validity of the timestamp to others when desired (hopefully without the interaction of any authority).

This can be achieved by using a public-key cryptosystem whereupon an authority signs the certificate, and anyone can verify the signature using a published public key.

2. **Authenticity** of the timestamp certificate needs to be assured. This means that people should trust that nobody else could have generated the timestamp other than the authority.

Again, this can be achieved using a public-key signature on the document and time by the authority.

3. certificates should be **efficient to compute** if the service wishes to scale with demand. A timestamping authority should not get bogged down with demands causing a backlog and therefore incorrect timestamps.

One speedup is to sign the hash of the document and time instead of the entire document. Another approach is to allow for multiple signing authorities, though this would require synchronizing clocks, which is in general not easy to do.

4. **Authorities can’t cheat.** It is very desirable in a system which can decide cases where lots of money is involved that the system can’t cheat. A student should not be able to team with his friend at the authority to get him to backdate his homework.

One way to achieve this is to link all the signatures in some fashion (potentially a tree or just a chain), so that in order to demonstrate a valid timestamp, you need to find the documents signed before and after for several hops. Also, some sort of checkpointing to some easy to find document (perhaps the NY Times) can be done to commit strongly.

5. **Documents should not be revealed** unless the writer wishes to do so. Many situations require proof that a document was written at a certain time, but without revealing the contents of the document (trade secret perhaps).

Instead of signing a document, the service can sign a strongly collision-free hash function of the document, plus the current time.

There are many other features people listed, and ways of achieving them. The general idea was that it should apply to the topic of timestamping, and that the solution made sense.

Problem Q-3. Chosen Ciphertext Attack

In a *chosen ciphertext* attack (CC), an adversary is given access to a decryption black box to which he may submit strings of his choice, and for which he will receive decryptions or an indication that the input was not a valid ciphertext.

The following cryptosystem is based on the difficulty of finding the square root of a number in a field $\text{mod}(pq)$. This problem has been proven to be equivalent to factoring. Any number that is a square in $\text{mod}(pq)$ has four square roots in the field: $x, -x, y, -y$. For example, in $\text{mod}(35) = \text{mod}(5 \cdot 7)$, the square roots of 16 are 4, $-4(= 31)$, 11, $-11(= 24)$.

Computing these square roots is easy if you know the factoring of n , but hard if you don't. Similarly, having two unique square roots (x, y) where $x \neq \pm y$, makes factoring easy ($\text{gcd}(x \pm y, n) = p$ or q).

Encrypt in the following manner:

$$E(x) = x(x + 2) \pmod{n}$$

where n is the public information, and the factors p, q make up the private info.

To decrypt, complete the square and find the right square root (remember there are four of them):

$$D(E(x)) = \sqrt{E(x) + 1} - 1 = \sqrt{x^2 + 2x + 1} - 1 = \sqrt{(x + 1)^2} - 1 = x \pmod{n}$$

This operation will yield four potential ciphertexts. Usually, one of these will make more sense than any of the others, and so it is chosen. There are better approaches to solving the problem of four potential plaintexts, but for this problem, just assume that the one most closely resembling ASCII text is the one that is chosen.

Given access to a black-box decryptor (which has the keys built in), can you devise a way, using a chosen ciphertext attack, to yield the factoring of the public modulus n ? Explain how it works, and why. Note that the black box will yield only ONE decryption, not all four. If the message submitted wasn't a possible ciphertext, it will just result in an error message.

Yes, it is relatively easy to attack this cryptosystem.

What you need to do is discover two unique square roots. The approach we use is to start with one, and see if the box will give us the other. If this box computed the function $E(x) = x^2 \pmod{n}$ (and so $D(c) = \sqrt{c} \pmod{n}$), we could start with a random $x \in_R Z_n^*$, compute $y = D(x^2)$. If $y = \pm x \pmod{n}$ then try again. Otherwise, we have found two unique square roots. Since we chose x at random and squared it, the decryptor has no way of knowing if we started with $x, -x, y, -y$, since they will all yield $x^2 \pmod{n}$ when squared. So even though the decryptor may be clever and try and give us what we started with, it can't, and

so we have at worst a 50% chance of getting a useful square root. This means that after an expected 2 attempts, we will have found a useful square root, and have factored n .

Now, our encryption system is a little funkier, but the basic idea is the same:

$$D(x) = \sqrt{x+1} - 1$$

So, select $u \in_R Z_n^*$. let:

$$v = D(u(u+2)) = \sqrt{u^2 + 2u + 1} - 1 = \sqrt{(u+1)^2} - 1 \pmod{n}$$

The resulting v will be one of u , $-(u+1)-1 = -(u+2)$, or a completely new pair of numbers. If we get the above, then we try again. The other numbers correspond to the other square roots thus: $v+1 = \sqrt{(u+1)^2}$. So we have found unique square roots of the same number: $u+1$, $-(u+1)$, $v+1$, $-(v+1)$. Then we can simply compute : $\gcd((u+1) - (v+1), n)$ or $\gcd((u+1), (v+1), n)$ to get a factor of n , and we're done.

Problem Q-4. Information Theoretic Cryptography

Consider the following proposed encryption/authentication scheme. Messages are elements of Z_p^* , for a suitable prime p . (Here p does not need to be very large, as we are not aiming for computational security, but for information-theoretic security.)

Alice and Bob have pre-arranged random key values a_i and b_i for $i = 1, 2, \dots$, where $a_i \in Z_p$ and $b_i \in Z_p^*$. For each message M_i they wish to send, they use a_i and b_i as the encryption/authentication key. That is, the keys are used only once; fresh keys are used for each message. To encrypt/authenticate a message $M_i \in Z_p^*$, Alice sends Bob the indexed ciphertext:

$$(i, c_i, d_i) = (i, M_i + a_i, M_i b_i),$$

where addition and multiplication are performed modulo p . (Note that d_i is non-zero.) When Bob's decryption equipment receives the indexed ciphertext (i, c_i, d_i) , it first checks to see that the i used is the correct one (that is, the next one in sequence; if not, it discards the input), then computes $M_i = c_i - a_i$, and checks that this result is equal to d_i/b_i . If they are equal, it returns the message M_i as the authenticated plaintext. Otherwise, it erases all of its intermediate results, including a_i and b_i , increments the internal i , and waits for the next input.

- (a) Argue that the encryption scheme is information-theoretically secure: a passive adversary learns nothing about M_i by hearing (i, c_i, d_i) .
 - (b) Argue that an "impersonation" attack succeeds with probability $1/p$. (Here the adversary tries to generate an indexed ciphertext that Bob will accept.)
 - (c) Argue that a "substitution" attack can be mounted successfully. Here the adversary overhears an indexed ciphertext (i, c_i, d_i) for which she *knows* (by some side channel) the corresponding message M_i , and substitutes the indexed ciphertext with another one of her own construction. How could you modify the scheme to provide protection against a substitution attack?
 - (d) Can you describe another attack that can be mounted against this system? How could you modify the scheme to provide protection against this attack?
-

Problem Q-5. Diffie-Hellman With Passwords

Let p be a fixed public prime and let g be a fixed public generator modulo p . Assume that p is large enough so that finding a discrete logarithm to the base g , modulo p , is infeasible for anyone.

Alice and Bob share a secret password w . They wish to securely establish a shared secret key between them and use w to authenticate each one to the other. They are concerned about “dictionary attacks” on the part of a passive eavesdropper who overhears the key-establishment dialogue between Alice and Bob, and also about the possible disclosure of the password to an active eavesdropper who initiates a key-establishment dialogue with Alice or Bob (pretending to be the other), or who tries a “man-in-the-middle” attack. They would like to use a variant of the basic Diffie-Hellman protocol, wherein Alice sends Bob g^x for a randomly chosen x , Bob sends Alice g^y for a randomly chosen y , and they agree upon the key $K = g^{xy}$. (All computed modulo p , of course.) Once K is established, they will use RC5 or another suitable block cipher to encrypt messages, and MAC the messages with a variant K' of K (say by using $K' = K + 1$.)

Alice and Bob have two possible schemes for accomplishing their goal.

- (Scheme 1.) Use $K = g^{xy} + w$ rather than $K = g^{xy}$, and then proceed as before. (Here it is presumed that the password can be interpreted as a residue modulo p in some natural manner.)
- (Scheme 2.) Instead of sending g^x to Bob, Alice sends $\text{RC5}(w, g^x)$. That is, she encrypts g^x with RC5 using w as the encryption key. (Bob can, of course, decrypt this.) Similarly, Bob sends Alice $\text{RC5}(w, g^y)$, which Alice decrypts. Everything else works as in the basic Diffie-Hellman scheme.

Discuss and compare the security of these two schemes. What recommendations, if any, can you make to Alice and Bob?
