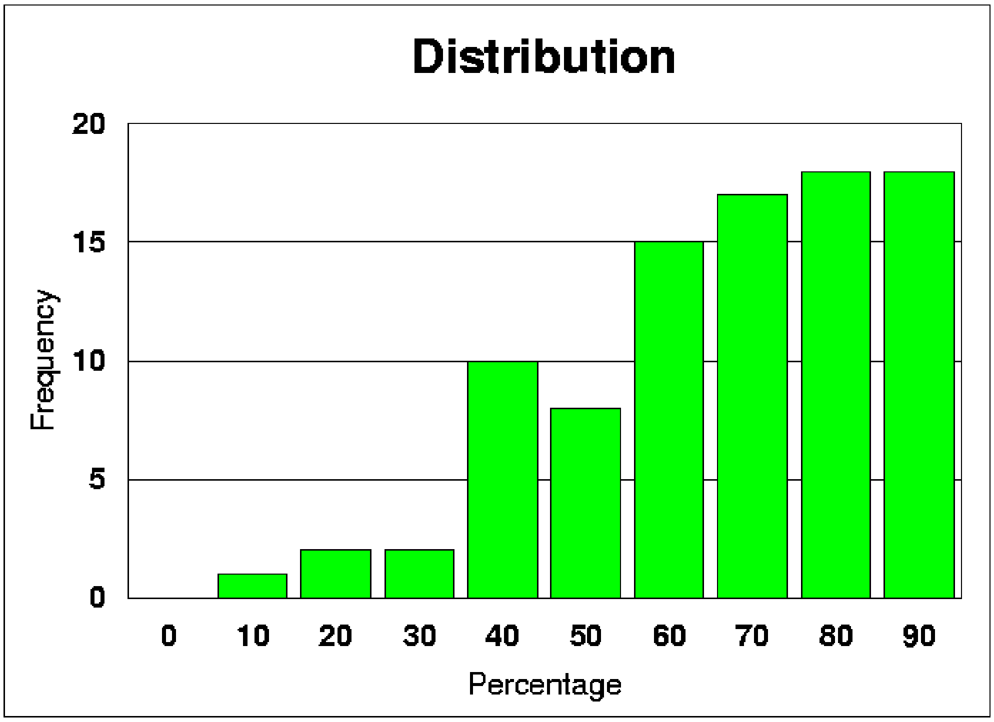
Massachusetts Institute of Technology | Handout 33
6.857: Network and Computer Security | December 3rd, 1998
Professor Ronald L. Rivest

Midterm Solutions



Class Average	74
Media	77
Std. Deviation	16

Problem Q-1. Perfect Forward Secrecy

Show how you can achieve perfect forward secrecy in key agreement using only traditional public-key techniquues. That is, your solution should not depend upon use of the standard Diffie-Hellman key agreement mathematics exchanging g^x and g^y and using g^{xy} as the key, but only on the existence of public-key encryption and signature algorithms as necessary.

For notation, please use $E_{Ap}(M)$ to denote the public-key encryption of M using Alice's public key, $S_{As}(M)$ to denote the public-key signature of M using Alice's secret key.

Total : 15 points

The following is a possible solution (definitely a simple one).

Requirements:

- Forward Secrecy: If an adversary compromises the system in the future, he will learn nothing of the current communications.
- Key Agreement: Alice and Bob have a single secret session key K which they can use in an efficient secret-key cryptosystem.

- $A \to B$: PK_r – Alice generates a random public/private key pair. She sends Bob the public key.
- $B \to A$: $E_{PK_r}(K)$ – Bob generates a random session key, and sends it encrypted to alice in her public key. Only Alice can decrypt it, she does, and then discards her PK pair.

At the end of their conversation, all of the keys are thrown away. Since Alice was the only one who could decrypt the key, and she throws away that ability at the end of the key exchange, nobody can decrypt the key. Once Alice and Bob discard the key, there is no record of the key left, so this scheme exhibits forward secrecy in key exchange

Notes: People lost points for having flawed protocols. Many people added authentication (unrequired) by requiring that Alice and Bob have pre-agreed PK pairs. This was not required, and the assumption makes the scheme less useful (DH doesn't require pre-established keys). Some people noted that Bob has full control over the key and tried to come up with ways of making BOTH people involved. Also, some people forgot to generate a session key, and ended up with multiple public key pairs.

Problem Q-2. Strengthening RSA

The best factoring algorithm known to date (the *number field sieve*) runs in

$$e^{O(\log^{1/3} n \log\log^{2/3} n)}$$

That is, the running time does not depend on the size of the smallest factor, but rather in the size of the whole composite number.

The above observation seem to suggest that in order to preserve the security of RSA, it may not be necessary to increase the size of both prime factors, but only of one of them.

The following idea is designed to strengthen RSA: Choose the RSA modulus n to be 5,000 bits long, the product of a 500-bit prime p and a 4,500-bit prime q. Since RSA is usually used just to exchange DES keys we assume here that the message being encrypted is smaller than p.

(a) How would you choose the public exponent e? Explain why 3 is a bad choice?

(b) Once the public exponent e is chosen, one computes $d = e^{-1} \bmod \phi(n)$ and keep it secret. The problem with such a big modulus n, is that decrypting a ciphertext $c = m^e \bmod n$ may take a long time (since one has to compute $c^d \bmod n$.) But since we know that $m < p$ we can just use the Chinese Remainder Theorem and compute $m_1 = c^d \bmod p = m$. This scheme appears to protect us better against the number-field sieve, while still allowing our computations to be efficient.

Show how with a single chosen ciphertext attack (i.e. obtaining the decryption of a ciphertext of your choice) you can completely break this variant RSA scheme, by factoring n.

Total: 20 points

Solution to (a): 10 points.

$n = pq$. Assume w.l.o.g. that $p << q$, $m < p$. If $e = 3$, $m^3 < p^3 < n$. This means we can compute the decryption of m^3 by taking the integer cubic root (no modulo necessary, since we never 'wrapped around'). So the choice of $e = 3$ is VERY bad. One must choose e to be fairly large.

Large in this case is enough to make messages wrap around an unpredictable number of times. Since the messages can vary from 1 to $p - 1$, having a large choice of e with many zeros in it (such as $2^{16} + 1$) is a good idea.

Solutions to (b): 10 points.

If we choose the message $-1 \bmod n \equiv (n-1)$, we know that its decryption is going to be -1 as well (since d is odd). However, we are going to be given $-1 \bmod p = p - 1$, from which we can quickly compute p. Note this creates a ciphertext that would normally not happen since we assumed that $m < p$, but it is nevertheless a valid attack on this system.

Problem Q-3. Favorite stuff

What for you has been the most surprising and/or interesting material that you have seen in this course this term? Describe it and explain why, in AT MOST one page.

Total: 10 points

There is no real solution to this problem. You were expected to write up something that demonstrated an understanding of some material, as well as explain what you found interesting about it.

Problem Q-4. Hash function security

Design a hash function h for which it is easy to find many pairs x, x' where $x \neq x'$ such that $h(x) = h(x')$, but for which it is arguably hard, given only a value $h(x)$, to find *any* value x' such that $h(x) = h(x')$. (Your function $h(x)$ must depend on all of the bits of x, in the sense that flipping any bit of x is likely to change the value of $h(x)$.)

Total: 15 points

There are many possible solutions to this, some using mathematics, and some basing a scheme on another already-designed hash function.

One good solution involved computing the following hash function:

$$H(m) = g^m \bmod p$$

We believe that this is hard to invert given the discrete log assumption. However, finding collisions is fairly trivial (just add $p - 1$ to any message to get a collision).

Problem Q-5. Rock-Papers-Scissors

In the two-person game "rock paper scissors" the two players simultaneously either show a pair of fingers (scissors), a fist (rock), or a flat hand (paper). If both show the same, it is a tie (they do it over), otherwise "scissors cuts paper," "paper wraps rock", and "rock breaks scissors" (that is, scissors beats paper, etc.).

Consider implementing this game remotely over a network. Describe a high-level approach to implementing this game. State clearly the security requirements that must be met in this approach. Give a plausible way of implementing your approach.

Total: 20 points

This is probably the most free-form problem on the midterm. The following is a guideline for what we were looking for.

An analysis of requirements. These include:

- Nobody may change their mind (i.e. we need a commit) after some point.
- Nobody may 'decide' based on other people's decisions. This includes worries about being able to choose your hand as a function of an encryption or commit of somebody else's.
- There is a clear winner or tie at the end (both can verify it).
- When playing multiple games, "rock" shouldn't always look like the same message (randomization).

There should also be others that indicate clear thought about the problem at hand.

A protocol description. One possibility is to use bit commitment functions to commit two bits ("11" can be deemed invalid). Then, reveal them after seeing a commitment of the other player's hand. A description of the actual protocol is useful here.

An analysis of why the protocol meets the 'requirements' posed, or where it fails and why. Some common observations include a problem with the fact that somebody always reveals last, giving them an opportunity to not reveal.

Note: People lost points for using a third party in this system, because that is no longer the same as RPS. While it is true that using a third party prevents the reveal problem (which is the only argument for using it), it introduces many more problems of trust. Furthermore, it is not the same game when it requires a referee (which real RPS doesn't).

Problem Q-6. Discrete-log based signature schemes

Consider the following discrete log based signature scheme. Let p be a large prime and g a generator of Z_p^*. The private key is $x < p$. The public key is $y = g^x \bmod p$.

To sign a message M, calculate the hash $h = H(M)$. If $gcd(h, p-1)$ is different than 1 then append h to M and hash again. Repeat this until $gcd(h, p-1) = 1$. Then solve for z in

$$zh = x \bmod (p-1)$$

The signature of the message is $s = g^z \bmod p$. To verify the signature, a user checks that $s^h \equiv y \pmod p$ and that h was obtained from M using the method outlined above.

(a) Show that valid signatures are always accepted.

(b) Is this scheme secure? Explain your reasoning.

Total: 20 points

For part (a): 10 points.

For a valid signature, we need to check that $(\sigma(M))^h \equiv y \pmod p$. And we know that $zh \equiv x \pmod{\phi(p)}$, so, by a corollary to fermat's theorem, we have $(\sigma(M))^h \equiv g^{zh} \equiv g^x \equiv y \pmod p$, and we therefore accept.

For part (b): 10 points.

Unfortunately, this scheme is not secure. It is possible to forge a signature for ANY message.

While it's true that $z \equiv xh^{-1} \pmod{p-1}$ is difficult to compute, $g^z \equiv g^{xh^{-1}} \equiv y^{h^{-1}}$ is easy.

So, to get the signature for any message M, repeat the process to get h until $gcd(h, p-1) = 1$. Then, compute $h^{-1} \bmod (p-1)$, and the signature is just $y^{h^{-1}} \bmod p$.