18733: Applied Cryptography S17

Homework 3

due March 10, 2017, 11:59pm EST

1 Cyclic Groups Warmup [3 points]

Let a prime p be of the form $p = 2q + 1$ where q is also prime. Prove that $g \neq 1$, $gcd(g, |\mathbb{Z}_p^*|) = 1$ is a generator of $\mathbb{Z}_p^*$ if and only if $g^q = -1$.

2 Security of Provable Compression Function [3 points]

In the recitation we saw some examples of symmetric key constructions which uses number theoretic guarantees. One example of constructions was 'provable compression function', which is defined as follows:

$$h(H, m) = u^H \cdot v^m \pmod{p},$$

where p is a large prime, u, v is a random numbers in Z_p^*, m, H being inputs to the compression function.

Prove from reduction that $h(\cdot, \cdot)$ is collision-resistant if discrete log problem of (g, p) is hard.

3 Taking Roots Modulo Composite Numbers is Hard [4 points]

In class we discussed some efficient algorithms for computing $x \equiv \sqrt{x^2}$ *mod* p where p is prime. For this problem, you are asked to show that taking the square root of a number modulo a composite is at least as hard as factoring an RSA modulus $N = pq$ where p and q are prime.

Let A be a polynomial time (in n, the number of bits in the binary representation of the modulus) randomized algorithm that given a uniformly random quadratic residue $y \equiv x^2$ *mod* N outputs a square root of y with non-negligible probability.

Use a reduction proof to show that taking the square root of a number modulo N is at least as hard as factoring N by describing an efficient algorithm using A to factor N efficiently with non-negligibe probability.

4 PRG from the Discrete Log Problem [4 points]

Recall the discrete log problem that we discussed in class. Let G_{2q} be a group of order $2q$ formed by taking the group of $\mathbb{Z}_p$ were $p = 2q + 1$ is a strong prime, then given any generator of this group, say g, and an element of the group g^x *mod* p, it is hard to find x. More formally:

$$Pr[q \leftarrow \Pi_n; g \leftarrow Gen_{G_{2q}}; x \leftarrow \mathbb{Z}_p : A(g^x) = x] < \epsilon$$

Here Π_n is an algorithm that randomly draws from the set of n bit prime numbers, $Gen_{G_{2q}}$ is an algorithm that draws a generator g from a group of order $2q$, x is randomly drawn from the integers mod p and A is any probabalistic polynomial time adversary (PPT).

Intuitively, if going from g^x to x is hard, then some bits of x or some function of the bits of x are difficult to find, we call these bits a *hardcore predicate* for the exponentiation function $f_{q,g}(x) = g^x$ *mod* p in the discrete log setting.

Let $\mathbf{half}_n(g^x, g, p)$ be defined as 0 iff $0 \leq x < \frac{n}{2}$, and 1 otherwise. This intiutively says if x is in the first half of the closed interval $[0, n]$ or not.

We want to prove that $\mathbf{half}_{p-1}(g^x, g, p)$ is a *hardcore predicate* for the discrete log problem – that is, output of the function half is difficult to find. To show this, use a reduction proof that assumes the existance

an adversary A that can perform $\mathbf{half}_{p-1}(g^x, g, p)$ in the discrete log setting and use adversary A to construct another adversary B who can easily find x given $g^x mod p, g, p$.

Hint: Think of an algorithm that takes the square root of g^x, i.e. $\sqrt{g^x}\ mod\ p$. Notice that if the square root of g^x is defined then g^x can be expressed as $g^{2y} = (g^y)^2$ and g^x has two square roots, namely g^y and $g^{y+\frac{p-1}{2}}$. Think about how to reason about these two square roots using the function $\mathbf{half}_{p-1}(g^x, g, p)$.

5 (Programming) Oracle Padding Attack [6 points]

You are an adversary Eve that can eavesdrop on a communication channel between Alice and her banking website. Her bank's website exports a page, banking.php which takes in a CBC encrypted secret token. The CBC encryption mode uses 128 bit blocks, and the token is padded upto the correct length using a padding scheme known as PKCS7. PKCS7 means that if n bytes need to be appended, then the value of those bytes is n. The resulting ciphertext is $C = IV||CBC(IV, m, k)$.

Using your own account with the bank you have learned a few aspects of how the system works. You noticed that if banking.php is requested with a correctly encrypted token, then an encrypted page is returned. You have also noticed that if you request the page with the plaintext token, a plaintext version of the page is returned instead. Finally you have learned that if you request the page with a ciphertext such that the attempted decryption has an invalid pad, then the page responds with an HTTP 404 error. If the padding is correct but the tag is incorrect then an HTTP 403 is generated. **Additionally, you have observed that at the start of every hour the bank refreshes all of its keys and tokens.**

Your goal is to obtain the decrypted token and therefore obtain the decrypted banking page. The ciphertext that you eavesdrop may be obtained by from:
http://possibility.cylab.cmu.edu/18733/eavesdrop.php
You can then make your requests to the banking page as:
http://possibility.cylab.cmu.edu/18733/banking.php?arg=<token>

The following example code will show you how to interact with the server using python, although any reasonable programming language can be used for this assignment:

```
import urllib2

def strxor(a, b):
    if len(a) > len(b):
            return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])
        else:
                return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])


#0 = Incorrect padding
#1 = Correct padding but malformed message
#2 = No error generated, either correct message or correct ciphertext
def oracle(decimal_token):
        try:
                page = "http://possibility.cylab.cmu.edu/18733/banking.php?arg="
                url = page + decimal_token.encode('hex')
            request = urllib2.urlopen(url)
        except urllib2.HTTPError, e:
            if e.code == 404:
                return 0
            if e.code == 403:
                return 1
        return 2
```

```
def solution(ciphertext):
    plaintext = ""

    #Ask the oracle questions and do some things here to return the plaintext
    return plaintext

print "The token is : " + solution(<ciphertext>)
```

(a) **[4 points]** Submit your code that finds a valid decrypted token as well as the value of the token that you obtained. Remember to check that it is valid by requesting the page with it.

(b) **[1 point]** Suppose that instead of an oracle padding attack that uses the HTTP code as a side-channel, there was a timing side channel. Let the runtime of the algorithm be 2ms if the token is correct, 3ms if the padding is correct but the messsage is malformed, and 4 ms if the padding is malformed. Would an attack against this still be possible? If so, how would the time to perform this attack depend on Eve's position within the network, i.e. how geographically far away she is from her targets.

(c) **[1 point]** Describe at least 1 possible mitigation for the oracle padding attack **without removing the side channel that is being used.**