Introduction to Modern Cryptography, Assignment #2

Benny Chor and Rani Hod

Published: 15/11/2009; due: 2/12/2009, in Rani's mailbox (Schreiber, 2nd floor).

This assignment contains six "dry" problems and one "wet" problem. Efficient solutions are always sought, but a solution that works inefficiently is better than none. The answers to the the "wet" problem should be given as the output of a `Sage`, `maple` or `WolframAlpha` session.

1. **Enhancing DES**

   The following two keys enhancements to DES were proposed in order to increase the complexity of finding the keys by exhaustive search.

$$\begin{aligned} \mathrm{DES^V}_{k,k_1}(M) &= \mathrm{DES}_k(M) \oplus k_1, \\ \mathrm{DES^W}_{k,k_1}(M) &= \mathrm{DES}_k(M \oplus k_1) \end{aligned}$$

   The keys' lengths are $|k| = 56$ and $|k_1| = 64$ (k_1 has the same length as the block length). Show that both these proposals do not increase the complexity of breaking the cryptosystem using brute-force key search. That is, show how to break these schemes using on the order of 2^{56} DES encryptions/decryptions. You may assume that you have a moderate number of plaintext-ciphertext pairs, $C_i = \mathrm{DES^V}/\mathrm{DES^W}_{k,k_1}(M_i)$.

2. **Meet in the Middle**

   In lecture 4 we described a *meet in the middle* attack against *double* DES. The attack required 2^{56} decryptions, 2^{56} encryptions and storage for 2^{56} messages (the decryptions of the ciphertext under all possible keys), 64 bits each. The attack used a small number of plaintext/ciphertext pairs: M_i and $C_i = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(M_i))$.

   You were hired to perform the same task, but your employer, hurt by the recent market trends, has supplied you with a machine capable of storing only 2^{40} words of 64 bits each. How many encryption and decryption operations do you need in order to recover the secret key k_1, k_2 with high probability? Does the number of required plaintext/ciphertext pairs increase?

3. **Cryptographic Hash Functions**

   Let $m = m_1 m_2 \ldots m_n$, where every m_i, $i = 1, \ldots, n$, is a 128 bits binary string. We define a hash function H that operates on messages of this form.

   - h_0 is defined as the all zero string of length 128.
   - For every i, $1 \le i \le n$, define $h_i = \mathrm{AES}_{m_i}(h_{i-1})$.
   - $H(m) = h_n$.

   (a) Show how to find collisions for H (namely two different messages that are mapped by H to the same string) using approximately 2^{64} AES applications.

   (b) Given a random string m, show how to find a different string m' such that $H(m) = H(m')$, using approximately 2^{64} AES applications.
   Hint: Recall the attack on double DES.

4. **CBC-MACs and variable length messages**

In this problem we will explore the security of CBC-MACs when the length of the message is allowed to vary. The constructions use a block cipher, $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, which you should assume to be secure ($E_K(t)$ is the encryption of a block t of length n under the key K of length k).

In general, let $\mathbf{x} = x_1 x_2 \cdots x_\ell$, where $x_i \in \{0,1\}^n$ for each $i = 1, \ldots, \ell$. For all the variants considered in this problem, the authentication of the message $\mathbf{x}$ is defined as the concatenation of $\mathbf{x}$ with $\mathrm{MAC}_K(\mathbf{x})$, where K is the secret key (shared by Alice and Bob), and $\mathrm{MAC}_K(\mathbf{x})$ is of length n.

- We say that Fred, the forging adversary, succeeds if after seeing a small number of messages $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_s$ *of his choice* and their MACs under the *unknown* secret key K, he can produce a new message $\mathbf{w} \notin \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_s\}$ together with $\mathrm{MAC}_K(\mathbf{w})$.
- We emphasize that $\mathbf{w}$ can (and typically will) be constructed out of pieces depending on the $\mathbf{z}_i$'s.
- By small number we mean s is either a constant or at most a fixed polynomial in n, the block length of $\mathbf{x}$. In addition to the number s of message/MAC pairs, Fred is also limited to polynomial (in n) time computations.

Remark: This type of forgery is called *adaptive existential forgery* (adaptive since the choice of $\mathbf{z}_{i+1}$ can depend on all previous i message/MAC pairs, and existential because it demonstrates the existence of a message whose MAC can be forged). This is the strongest form of "reasonable adversary" considered in the crypto world.

(a) Consider the application of 'standard' CBC-MAC to messages of arbitrary length. Formally, given $\mathbf{x} = x_1 x_2 \cdots x_\ell$, we define $y_0 = 0^n$ and $y_i = E_K(y_{i-1} \oplus x_i)$ for $1 \le i \le \ell$. Then $\text{CBC-MAC}_K(\mathbf{x}) = y_\ell$.
Show that this MAC is completely insecure; break it with a constant number of queries.

(b) In order to overcome the problem of applying 'standard' CBC-MAC to messages of arbitrary length, consider the following patch.

$$MAC_K(x_1, x_2, \ldots, x_\ell) = \text{CBC-MAC}_K(x_1, x_2, \ldots, x_\ell, \ell) \ ,$$

where the number ℓ of blocks in $\mathbf{x}$ is written in binary using n bits.
Show that this patch does not hold water either; break it with a constant number of queries.

(c) Consider the following attempt to allow one to MAC messages of arbitrary length. The domain for the MAC is $(\{0,1\}^n)^+$. To MAC the message $\mathbf{x} = x_1 x_2 \cdots x_\ell$ under the secret key (K, K'), compute $\text{CBC-MAC}_K(\mathbf{x}) \oplus K'$, where K has k bits and K' has n bits.
Show that this MAC is completely insecure; break it with a constant number of queries.

5. **Orders**

(a) Let a, m be two positive integers, with $1 \le a \le m-1$. The order of a modulo m, $ord_m a$, is defined as the minimum positive integer ℓ such that $a^\ell = 1 \bmod m$, and ∞ if no such ℓ exists.
Prove that $ord_m a$ is finite if and only if $\gcd(a, m) = 1$.

(b) Let x be an integer and let p be an odd prime divisor of $x^{16} + 1$. Prove that $p = 1 \bmod 32$.

6. **Primitive Elements in $GF\left(p^k\right)$**

In assignment #1 we saw that in some cases (e.g., $p = 2$ and $k = 5$) $GF\left(p^k\right)$ has $p^k - 2$ primitive roots (that is, all elements of $GF^*\left(p^k\right)$ except 1 are primitive).

(a) Show that, for $p > 2$ and $k > 1$, $GF^*\left(p^k\right)$ always has some non-primitive element that is not 1.

(b) Find $p - 2$ such elements explicitly (using the representation of finite fields arithmetic).

7. **Primitive Elements in $\mathbb{Z}_p$**

Let $p > 2$ be a prime number. Recall the algorithm we saw in class to efficiently test whether $g \in \mathbb{Z}_p^*$ is a primitive element given the list $p_1, \ldots, p_k$ of all prime factors of $p-1$: test whether there exists no $1 \leq i \leq k$ such that $g^{(p-1)/p_i} = 1 \bmod p$.

(a) Unfortunately, factoring integers is a hard problem, even if they are of the special form $p-1$. Take $p = 249 \cdot 2^{249} - 1$. Using `Sage`'s `is_prime` function, verify that p is indeed a prime. Now apply `factor` to $p-1$. This procedure tries to factor its integer argument. For large integers it obviously not always succeeds, neither does it always succeed for primes minus 1. However for our p, `factor` produces the complete factorization in a few minutes (even on Benny's MacBook). Produce this factorization.

(b) Implement `is_primitive_root`. This function should take p and g as arguments and return True iff g is primitive in $\mathbb{Z}_p$. You can use `Sage`'s built-in `mod(g, p).multiplicative_order()` to check your code.

(c) Use the code from (b) to find a *random* integer $g > 10^7$ such that g is a primitive element of $\mathbb{Z}_p$ (for $p = 249 \times 2^{249} - 1$) but $g+1$ is *not* a primitive element of $\mathbb{Z}_p$.[1]

Hint: you may want to add the factorization of $p-1$ as an optional arguement to `is_primitive_root`, so you wouldn't have to recompute it every iteration.

(d) Instead of looking for a prime p and trying to factor $p-1$, a different procedure is to look at random for a prime q and then test if $p = 2q+1$ is also a prime. In that case, a complete factorization of $p-1$ is $2 \times q$. You may think that having both q and $2q+1$ being primes is such a rare event that we will never run into one. Write a short `Sage` code that prints out *two* such random pairs, with $q > 2^{400}$ in both cases. You can definitely use `is_prime` here, as well as `next_prime`.

8. **Claw Free Permutations**

Two permutations $f_0, f_1 : D \to D$ are called *claw free*[2] if it is infeasible to calculate $x, y \in D$ such that $f_0(x) = f_1(y)$.

(a) Let p be a prime number, g be a primitive element in $\mathbb{Z}_p^*$, and $a \in \mathbb{Z}_p^*$. Define two permutations $f_0, f_1 : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ by $f_0(x) = g^x \bmod p$ and $f_1(y) = ag^y \bmod p$.

Assuming it is infeasible to calculate a z such that $g^z = a$, prove that f_0, f_1 are claw free permutations.

(b) Let $m = b_1 b_2 \ldots b_n$ be an n bit message (b_i are bits) and let f_0, f_1 be claw free permutations on D. Define the function H by

$$H(m) = f_{b_1}(f_{b_2} \ldots (f_{b_n}(IV) \ldots)),$$

where IV is the all zero string in D. For example, if $m = 011$ then $H(m) = f_0(f_1(f_1(IV)))$.

Assume that it is infeasible to find a $z \in D$ such that $f_0(z) = IV$ or $f_1(z) = IV$. Prove that H is a collision resistant hash function. In other words, show that if $m_1 \neq m_2$ and $H(m_1) = H(m_2)$, then we can *efficiently* either find a pair $x, y \in D$ such that $f_0(x) = f_1(y)$, or a $z \in D$ such that $f_0(z) = IV$ or $f_1(z) = IV$. Note that m_1 and m_2 can have different lengths.

[1] The probability that among the fewer than 100 values of g submitted by the course participants there will be a collision is so small that we will interpret such collision as a collusion and will act accordingly.

[2] This is not the same definition of *claw freeness* we saw in lecture 4.