Tel Aviv University 30/11/2016

Introduction to Modern Cryptography (0368.3049) – Ex. 3

Benny Chor and Orit Moskovich

Submission in singles or pairs to Orr Fischer's Schreiber mailbox (289) until 14/12/2016, 23:59 (IST)

- Appeals/missing grade issues: bdikacs AT gmail.com
- Issues regarding missing/unchecked assignments will be addressed only if a soft copy will be submitted on time to: crypto.f16 AT gmail.com.
Subject of the email: Ex.3, ID

1. We say that an adversary succeeds in forging a MAC, if it can choose a small (constant) number of messages $m_1, ..., m_s$, obtain their MACs under the unknown secret key k, and then produce a new message $m \notin \{m_1, ..., m_2\}$ together with it MAC $MAC_k(m)$.

   Let $F_k : \{0,1\}^n \to \{0,1\}^n$ be a collection of pseudo random functions (PRF).

   Define the following MAC schemes. Prove that all these MACs are *insecure* for messages of variable length:

   (a) $MAC_k(m) = F_k(m_1) \oplus F_k(m_2) \oplus ... \oplus F_k(m_\ell)$, where $m = m_1...m_\ell$ ($m_i \in \{0,1\}^n$)

   (b) $MAC_k(m) = (F_k(m_1), F_k(F_k(m_2))$, where $m = m_1m_2$ ($m_1, m_2 \in \{0,1\}^n$)

   (c) $MAC_k(m) = (F_k(0||m_1), F_k(1||m_2))$, where $m = m_1m_2$ ($m_1, m_2 \in \{0,1\}^{n-1}$)

2. Consider the following modification to the CBC-MAC:

   $$MAC_k(m_1, ..., m_\ell) = CBC\text{-}MAC(m_1, ..., m_\ell, \ell)$$

   i.e., execute the regular CBC-MAC on each of the blocks m_i, and on ℓ (the number of blocks).

   Show how to break the suggested MAC with a constant number of queries.

3. (a) Let $h : \{0,1\}^n \to \{0,1\}^n$ be a collision resistant function, that is not compressing. Is h necessarily a OWF?

   (b) Let $h : \{0,1\}^{2n} \to \{0,1\}^n$ be a collision resistant hash function (h is compressing). In addition, assume h is a regular function, namely for every $y \in \{0,1\}^n$ that is an image under h, the number of $x \in \{0,1\}^{2n}$ such that $h(x) = y$ is *exactly* 2^n.

   Let A be a PPT algorithm that inverts h w.p. $\geq \varepsilon$:

   $$Pr_{x \leftarrow U_{2n}}[A(h(x)) = x' \text{ such that } h(x) = h(x')] \geq \varepsilon$$

   Show that there exists a PPT algorithm A' that can find a collision in h with probability $\geq \varepsilon(1 - \frac{1}{2^n})$.

   (c) Let $h : \{0,1\}^{n+s} \to \{0,1\}^n$ be a collision resistant hash function (h is compressing). Let A be a PPT algorithm that inverts h w.p. $\geq \varepsilon$:

   $$Pr_{x \leftarrow U_{n+s}}[A(h(x)) = x' \text{ such that } h(x) = h(x')] \geq \varepsilon$$

   Show that there exists a PPT algorithm A' that can find a collision in h with probability $\geq \frac{\varepsilon}{2} - \frac{1}{2^s}$.

   Hint: (1) What is the probability that a random x does not collide with any other x', i.e., $\forall x' \neq x.\ h(x) \neq h(x')$? (2) Given that there exists $x' \neq x$ such that $h(x) = h(x')$, what is the probability that $A(h(x)) \neq x$?

4. In lecture 5, we have seen how to construct a hash function for variable length messages, based on a hash function for fixed length messages (see the diagram in slide 56).

   (a) Show that the scheme, as presented in class, is not secure for variable length messages. (That is, demonstrate how to create a collision.)

   (b) Suggest a modification for the scheme that will fix the issue (no need to formally prove that the new scheme is indeed a collision resistant hash function, but do supply a short textual argument).

5. In this question we show that any public-key encryption is not perfectly secure. Specifically, Let $\mathcal{E} = (Gen, Enc, Dec)$ be a PKE. Show that there exists an unbounded attacker A such that, for any message m:

   $$Pr_{\substack{(sk,pk) \leftarrow Gen \\ c \leftarrow Enc_{pk}(m)}}[A(c, pk) = m] = 1$$

6. Intuitively, a trapdoor function is a function that is easy to compute in one direction, yet difficult to invert, without some secret key.

   Definition: A function $F : \{0,1\}^n \to \{0,1\}^n$ is a trapdoor function with a key sampling algorithm Gen, and an inverting algorithm Inv if: Gen samples a pair of keys (sk, pk), such that given sk, it is possible to compute $x = Inv_{sk}(F_{pk}(x))$. On the other hand, given only pk, F_{pk} is one-way.

   Definition: ε-HCB for a trapdoor function is a predicate $B : \{0,1\}^n \to \{0,1\}$ such that: $pk, F_{pk}(x), B(x) \approx_{c,\varepsilon} pk, F_{pk}(x), u$ where $pk \leftarrow Gen, x \leftarrow U_n, u \leftarrow U_1$

Let $F : \{0,1\}^n \to \{0,1\}^n$ be a trapdoor function. Assume also that F has an ε-hardcore-bit $B : \{0,1\}^n \to \{0,1\}$. Show how to construct from F a public-key bit-encryption scheme. Prove it is 2ε-semantically-secure.

Hint: $Enc_{pk}(b,r) = F_{pk}(r), B(r) \oplus b$

7. **Multiplicative generators in Z_m^*.** In this problem we will make a mild usage of programming to explore the existence and abundance of multiplicative generators in three groups Z_m^*.

   Let $(G, \cdot)$ be a finite group with k elements, and denote by 1 its unit element. Recall that G is called *cyclic* if there is a $g \in G$ such that $< g >= G$, namely $\{g, g^2, \ldots, g^{k-1}, g^k\} = G$. Such g is called a *multiplicative generator* of G. Testing if a given g is a multiplicative generator using the equality above is feasible for small groups, but infeasible for large ones.

   Suppose we know the factorization of k: $k = p_1^{e_1} p_2^{e_2} \ldots p_\ell^{e_\ell}, e_i \geq 1$ (in particular, k has ℓ distinct prime factors). It is known that in such case, g is a generator iff $g^{k/p_1} \neq 1, g^{k/p_2} \neq 1, \ldots g^{k/p_\ell} \neq 1$ (recall that by Lagrange theorem, $g^k = 1$).

   (a) Write a *short and readable* code in your favorite programming language (you can choose either Python or Sage) which tries all elements in G and determines if each of them is a multiplicative generator or not. You should state if G is cyclic or not. If G is cyclic, your code should output the list of all multiplicative generators in G. Otherwise, the code should output the list of all elements in G having maximum order. Submit the code and the requested outputs.

   Recall that the elements in the group $(Z_m^*, \cdot_{\bmod m})$ are all integers in the set $\{1, \ldots, m-1\}$ that are relatively prime to m. There are $\phi(m)$ many elements in Z_m^*. Run your code for $m = 35, 37, 38$.

   Hint: if you use Python, `pow(a,b,m)` computes $a^b \pmod{m}$.

   (b) It is known that if m is a prime, Z_m^* has a multiplicative generator. In fact, such group has many multiplicative generators. For $m = 2^{107} - 1$, it is not feasible to try all $g \in G$. Instead, write a code that samples N elements $g \in G$ *uniformly at random*, and for each of them, tests if it is a multiplicative generator. Count the number of multiplicative generators, A, and output A, N, and the first 10 multiplicative generators your code finds.

   Use A and N to estimate the number of multiplicative generators in $Z_{2^{107}-1}^*$. N should be at least 100,000. Compare your estimate to $\phi(2^{107} - 2)$, which is the exact number of such generators. How good would you say your estimate is. Submit the code and the requested outputs.

   The function $\phi(m)$ was defined in class.

   In both Python and Sage, calling `import random` (once) imports the appropriate package for pseudo random generation. Then `g=random.randint(1,2**107-2)` is a pseudo random generator that produces a new `g` in `G` each time it is invoked. Serious distinguishers will probably tell it apart from a truly random sequence, but it is just fine for our needs.

   Oops: To solve the problem, the prime factorization of $2^{107} - 2$ is required. Well,

$$2^{107} - 2 = 2 \cdot 3 \cdot 107 \cdot 6361 \cdot 69431 \cdot 20394401 \cdot 28059810762433 \ .$$

8. In this problem, you will implement the arithmetic of $GF(3^4)$, using Sage, and look for multiplicative generators. Recall that $GF(3^4)$, which has 81 elements, is **not** the same as Z_{81}!

   - Find an irreducible polynomial $f(x)$ of degree 4 over $GF(3)$.
   - Write a Boolean Sage function `is_generator(y)`, which checks if a given $y \in GF(3^4)$ is a multiplicative generator.
   - Apply the Sage command `K.<a>=GF(3**4, name='a',modulus=f(x))` to implement $GF(3^4)$ arithmetic. Write a program that goes over all $GF(3^4)$ elements, identifies the multiplicative generators, and inserts all multiplicative generators into a set. The program should output the set of generators and its size.
   - Submit your code (including the irreducible polynomial $f(x)$ and a Sage "proof" it is indeed irreducible), the set of generators, and its size.

9. (a 10-Point bonus problem). A student in class proposed the following variant of Naor's bit commitment:

   a) The receiver Bob sends two random strings $w_0, w_1 \in \{0,1\}^n$ to the sender Alice.

   b) To commit to a bit $b \in \{0,1\}$, Alice chooses a random $s \in \{0,1\}^n$, and sends to Bob $G(s \circ w_b)$ ($\circ$ denotes concatenation), where $G : \{0,1\}^{2n} \to \{0,1\}^{10n}$ is a pseudo random generator.

   To decommit to b, Alice sends (b, s). Show:

   a) There exists a PRG G, such that the above scheme is not binding.

   b) There exists a PRG G, such that the above scheme is not hiding.

   (You can assume that for any polynomial length-functions $l(n) < l'(n)$, there exist PRGs that stretch $l(n)$ bits to $l'(n)$ bits.)