Advanced Crypto
Final Exam

2019-01-24

Exercise 1: Course questions

The (main) questions of this exercise are independent. The clarity and completeness of the answers will be taken into account for grading.

Q1:

1. Define the followings:
   - The Möbius transform of a Boolean function
   - The $\mathrm{RM}(r, m)$ of binary Reed-Muller codes
2. Explain how it is possible to encode efficiently a message into a Reed-Muller codeword without using a generator matrix. Is this encoding systematic?

Q2: Describe Prange's algorithm for finding a low-weight codeword of a binary linear code, and one of its improvements of your choice.

Q3: Let $\mathrm{P} : \{0,1\}^n \to \{0,1\}^n$ be a public permutation. We define the following block cipher constructions:

- $\mathrm{EM}_1 : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, as $\mathrm{EM}_1(k,p) = \mathrm{P}(k \oplus p) \oplus k$
- $\mathrm{EM}_2 : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, as $\mathrm{EM}_2(k,p) = \mathrm{P}(\mathrm{P}(\mathrm{P}(k \oplus p) \oplus k) \oplus k) \oplus k$
- $\mathrm{EM}_3 : \{0,1\}^{3n} \times \{0,1\}^n \to \{0,1\}^n$, as $\mathrm{EM}_3(k,p) = \mathrm{P}(\mathrm{P}(k_1 \oplus p) \oplus k_2) \oplus k_3$

1. Assuming that P does not have any structural weakness, which of EM_1, EM_2 and EM_3 provides the best security in a single-key attack setting?
2. Give a XOR related-key distinguishing attack for EM_1 that has advantage ≈ 1 and negligible time and query complexity. Can you adapt your attack to EM_2 and EM_3?

Exercise 2: an interpolation attack

The goal of this exercise is to study an *interpolation attack* on a low-degree block cipher.

Let the round function $\rho : \mathbb{F}_{2^{129}} \times \mathbb{F}_{2^{129}} \to \mathbb{F}_{2^{129}}$ be defined as $(k, x) \mapsto (x + k)^3$. It can be shown that ρ is invertible and has the best possible differential uniformity and linearity; this makes it a mapping with optimal protection against standard differential and linear cryptanalysis. However, it is quite obvious that the algebraic expression of ρ over $\mathbb{F}_{2^{129}}$ is simple, insofar as it is a sparse polynomial of low degree. This can be exploited to attack ciphers built from (too) few iterations of ρ.

Let us further define $\mathrm{E}_2 : \{0,1\}^{387} \times \{0,1\}^{129} \to \{0,1\}^{129}$ as $\mathrm{E}_2(k_1||k_2||k_3, x) = \rho(k_2, \rho(k_1, x)) + k_3 = ((x + k_1)^3 + k_2)^3 + k_3$.

Q1:

1. Show that the encryption c of a message m with E_2 under a key $k_1||k_2||k_3$ can be expressed as the following degree-9 polynomial:

$$\begin{aligned} c = m^9 + m^8k_1 + m^6k_2 + m^4k_1^2k_2 + m^3k_2^2 + m^2(k_1k_2^2 + k_1^4k_2) \\ + m(k_1^2k_2^2 + k_1^8) + k_1^3k_2^2 + k_1^6k_2 + k_1^9 + k_2^3 + k_3 \end{aligned} \tag{1}$$

(Hint: Remember that in characteristic-2 fields, $(a+b)^2 = a^2 + b^2$.)

Q2:

1. Propose a linearization of *Equation* 1. How many unknowns do you get?
2. How many known plaintext-ciphertext pairs do you need at minimum to retrieve all the unknowns of the linearization?
3. What is the time and query complexity to solve this system?
4. Does solving this system allow to retrieve the secret key $k_1||k_2||k_3$?
5. Does it allow to compute $\mathrm{E}_2(k_1||k_2||k_3, m')$ for an arbitrary message m' without knowing (or recomputing) $k_1||k_2||k_3$?

Q3: We recall that a degree-d polynomial $P(X)$ can be uniquely interpolated from its evaluation on $d+1$ distinct points $x_0, \ldots x_d$ as:

$$P(X) = \sum_{i=0}^{d} P(x_i) \prod_{0 \le j \ne i \le d} (X - x_j)/(x_i - x_j).$$

1. Explain how to solve the linearization of *Equation* 1 using interpolation.
2. What is the naïve time and query complexity of this method?
3. Do you think it is possible to do better? How?

Q4: We define E_n similarly as E_2, with n compositions of ρ instead of 2.

1. Do you think that the above attack is still efficient on E_6?
2. What about E_{10}?
3. And E_{89}?

(Hint: $\log_2(3) \approx 1.58$)

Remark. The interpolation attack presented here is due to Jakobsen and Knudsen (FSE '97).

Exercise 3: birthday algorithms for the subset sum problem

Given $k+1$ integers $x_1, \ldots, x_k, s \in \mathbb{N}$, a *subset sum problem*[*] of size k consists in finding k binary coefficients $\alpha_1, \ldots, \alpha_k \in \{0,1\}$ (if they exist) s.t. $\sum_{i=1}^{k} \alpha_i x_i = s$ (where this sum is computed over the integers). In other words, one wants to find a subset of a given generating set $\{x_1, \ldots, x_k\}$ that sums to a target s.

In all of the following, we assume random instances where $\#x_{1,\ldots,k}$ (the bitsize of $x_{1,\ldots,k}$) $\approx \#s = n$ and n is "large" (for instance $\approx k$).

In your answers to the questions of this exercise, algorithms must be described as clearly and as concisely as possible. This does not necessarily imply the use of formal pseudo-code.

Q1:

1. Describe an algorithm that solves a subset sum problem of size k with worst-case time complexity $\mathcal{O}(2^k)$ and constant memory complexity.

Q2: Let $\mathcal{S} = \{x_1, \ldots, x_k\}$ be the generating set of a subset sum problem instance. Assume k is even and define $\mathcal{S}_1 = \{x_1, \ldots, x_{k/2}\}$; $\mathcal{S}_2 = \{x_{k/2+1}, \ldots, x_k\}$, so that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$.

1. Show that there is a solution to the problem with generating set $\mathcal{S}$ and target s iff. $\exists s_1, s_2$ s.t.:
   (a) there is a solution to the problem instance of generating set $\mathcal{S}_1$ and target s_1;
   (b) there is a solution to the problem instance of generating set $\mathcal{S}_2$ and target s_2;
   (c) $s = s_1 + s_2$.
2. What is the time complexity required to compute the set $\mathbb{S}_1$ (resp. $\mathbb{S}_2$) of *all* targets s_1 (resp. s_2) that have a solution for the set $\mathcal{S}_1$ (resp. $\mathcal{S}_2$)?
3. Let us store $\mathbb{S}_1$ and $\mathbb{S}_2$ in two tables $\mathcal{T}_1$, $\mathcal{T}_2$. What is the expected size of those tables? How much does it cost to sort them?
4. We now assume that $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) has been sorted in increasing (resp. decreasing) order. Describe an algorithm that solves the subset sum problem in maximum time $\#\mathcal{T}_1 + \#\mathcal{T}_2$.
5. What is the total time and memory complexity of this algorithm (ignoring possible log factors)?

*

The objective of the remainder is to improve the algorithm of the previous question by decreasing its memory complexity without increasing its time complexity.

Q3: We recall that a maximum (resp. minimum) *priority queue* is a data structure that allows efficient (e.g. logarithmic time) insertion (`insert`) of an element and efficient extraction (`pop`) of the maximum (resp. minimum) element for some order relation. We also introduce a function $\texttt{next}(x, \mathcal{T})$ that returns the element following x (if it exists) in a sorted table $\mathcal{T}$.

Let us assume that $k \equiv 0 \mod 4$ so that we may further split $\mathcal{S}_1$ into $\mathcal{S}_{11} = \{x_1, \ldots, x_{k/4}\}$, $\mathcal{S}_{12} = \{x_{k/4+1}, \ldots, x_{k/2}\}$, that naturally define sets $\mathbb{S}_{11}$ and $\mathbb{S}_{12}$ of reachable targets. Let $\mathcal{H}_1$ be a (min.) priority queue initialized with $\{(x_1, x_2) : x_1 = \min \mathbb{S}_{11}, x_2 \in \mathbb{S}_{12}\}$ and $\mathcal{T}_{11}$ be a sorted table (in increasing order) of the elements of $\mathbb{S}_{11}$.

[*] This can also be considered a particular case of the more general family of *knapsack* problems.

1. Show that $\min \mathbb{S}_1 \in \mathcal{H}_1$ (where an element is represented as a pair (x, y) that sums to its value).
2. Show that the last of the following sequence of instruction returns the second smallest element of $\mathbb{S}_1$:
   (a) $(x_1, x_2) = \texttt{pop}(\mathcal{H}_1)$;
   (b) $\texttt{insert}((\texttt{next}(x_1, \mathcal{T}_{11}), x_2,), \mathcal{H}_1)$;
   (c) $(x'_1, x'_2) = \texttt{pop}(\mathcal{H}_1)$.
3. Describe an algorithm that outputs *all* the elements of $\mathbb{S}_1$ in *increasing order* with *memory complexity* $\mathcal{O}(2^{k/4})$. What is its time complexity (again ignoring log. factors)? (You will be careful to prove the emphasized statements.)

Q4:

1. Describe an algorithm that solves a subset sum problem of size k with worst-case time complexity $\mathcal{O}(2^{k/2})$ and memory complexity $\mathcal{O}(2^{k/4})$.

Q5:

1. Is the improvement of the algorithm of **Q3** over the ones of **Q2** and **Q1** practically useful in an implementation?
2. Would it be practically useful to further decrease the memory complexity?
3. Estimate the maximum problem size that can be solved on a reasonably priced server (e.g. *FunFoehn*: 16 cores @ 2.0 GHz; 256 GB of RAM) within one year ($\approx 2^{25}$ seconds) for all three algorithms.

Remark. The algorithm studied in this exercise is due to Schroeppel and Shamir (FOCS'79).