CS 355: Topics in Cryptography **Spring 2024**

Problem Set 2

Due: Friday, 26 April 2024 (submit via Gradescope)

Instructions: You **must** typeset your solution in LaTeX using the provided template:

`https://crypto.stanford.edu/cs355/24sp/homework.tex`

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **RKN4PX** to sign up. Note that Gradescope requires that the solution to each problem starts on a **new page**.

Bugs: We make mistakes! If it looks like there might be a mistake in the statement of a problem, please ask a clarifying question on Ed.

Note: The following two documents may help with number theory background on this assignment.

1. `https://crypto.stanford.edu/~dabo/cs255/handouts/numth1.pdf`
2. `https://crypto.stanford.edu/~dabo/cs255/handouts/numth2.pdf`

Problem 1: For each of the following statements, say whether it is TRUE or FALSE. Write *at most one sentence* to justify your answer [7 points].

1. Let p, q, r, and r' be distinct large primes. Let $N = pqr$ and $N' = pqr'$. Assume that there does *not* exist an efficient (probabilistic polynomial time) factoring algorithm. Say whether each of the following statements are TRUE or FALSE.

   (a) There is an efficient algorithm that takes N as input and outputs r.

   (b) There is an efficient algorithm that takes N and N' as input and outputs r.

   (c) There is an efficient algorithm that takes N and N' as input and outputs q.

2. Let $\mathbb{G}$ be a group of prime order q. Consider the following special cases of the discrete-log problem. For each of them, say TRUE if an efficient (polynomial in $\log q$) algorithm for the special case can be used to construct an efficient algorithm for the general case of the discrete-log problem, and FALSE otherwise.

   (a) An algorithm that correctly outputs the discrete log only when it is smaller than $q/\log q$.

   (b) An algorithm that correctly outputs the discrete log only when it is smaller than $\log q$.

3. Given $g \in \mathbb{G}$ and a positive integer n, a generic group algorithm requires $\Omega(n)$ time to compute g^n.

4. Let $\mathbb{G}$ be a cyclic group of prime order q with a generator $g \in \mathbb{G}$ and $H\colon \mathbb{G} \to \{1,2,3\}$ be a random function. A walk on $\mathbb{G}$ defined as $x_0 \xleftarrow{\mathrm{R}} \mathbb{G}$ and $x_{i+1} \leftarrow x_i \cdot g^{H(x_i)}$ collides in $O(\sqrt{q})$ steps in expectation (i.e., if $i_{\text{col}} = \min\{i \in \mathbb{N}\colon \exists j < i \text{ s.t. } x_i = x_j\}$, then $\mathbb{E}_{x_0,H}[i_{\text{col}}] \le O(\sqrt{q})$).

Problem 2: Coppersmith Attacks on RSA [15 points]. In this problem, we will explore what are known as "Coppersmith" attacks on RSA-style cryptosystems. As you will see, these attacks are very powerful and very general. We will use the following theorem:

> **Theorem** (Coppersmith, Howgrave-Graham, May)**.** Let N be an integer of unknown factorization. Let p be a divisor of N such that $p \geq N^{\beta}$ for some constant $0 < \beta \leq 1$. Let $f \in \mathbb{Z}_N[x]$ be a monic polynomial of degree δ. Then there is an efficient algorithm that outputs all integers x such that
>
> $$f(x) = 0 \bmod p \qquad \text{and} \qquad |x| \leq N^{\beta^2/\delta}.$$
>
> Here $|x| \leq B$ indicates that $x \in \{-B, \ldots, -1, 0, 1, \ldots, B\}$.

In the statement of the theorem, when we write $f \in \mathbb{Z}_N[x]$, we mean that f is a polynomial in an indeterminate x with coefficients in $\mathbb{Z}_N$. A *monic* polynomial is one whose leading coefficient is 1.

When $N = pq$ is an RSA modulus (where p and q are random primes of equal bit-length with $p > q$), the interesting instantiations of the theorem have either $\beta = 1/2$ (i.e., we are looking for solutions modulo a prime factor of N) or $\beta = 1$ (i.e., we are looking for small solutions modulo N).

For this problem, let N be an RSA modulus with $\gcd(\phi(N), 3) = 1$ and let $F_{\text{RSA}}(m) := m^3 \pmod N$ be the RSA one-way function.

(a) Let $n = \lceil \log_2 N \rceil$. Show that you can factor an RSA modulus $N = pq$ if you are given:

- the low-order $n/3$ bits of p,
- the high-order $n/3$ bits of p, or
- the high-end $n/6$ bits of p *and* the low-end $n/6$ bits of p.

(b) In the dark ages of cryptography, people would encrypt messages directly using F_{RSA}. That is, they would encrypt an arbitrary bitstring $m \in \{0,1\}^{\lfloor \log_2 N \rfloor/5}$ by

- setting $M \leftarrow 2^{\ell} + m$ for some integer ℓ to make $N/2 \leq M < N$, and
- computing the ciphertext as $c \leftarrow F_{\text{RSA}}(M)$.

(Note that the first step corresponds to padding the message M by prepending it with a binary string "$10000\cdots000$.")

Show that this public-key encryption scheme is very broken. In particular, give an efficient algorithm that takes as input (N, c) and outputs m.

(c) To avoid the problem with the padding scheme above, your friend proposes instead encrypting the short message $m \in \{0,1\}^{\lfloor \log_2 N \rfloor/5}$ by setting $M \leftarrow (m \| m \| m \| m \| m) \in \{0,1\}^{\lfloor \log_2 N \rfloor}$ and outputting $c \leftarrow F_{\text{RSA}}(M)$. Show that this "fix" is still broken.

(d) The RSA-FDH signature scheme uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N$. The signature on a message $m \in \{0,1\}^*$ is the value $\sigma \leftarrow F_{\text{RSA}}^{-1}(H(m)) \in \mathbb{Z}_N$. As we discussed in lecture, the signature σ is $n = \lceil \log_2 N \rceil$ bits long. Give a modification to RSA-FDH such that the signature is only $2n/3$ bits while still

- retaining exactly the same level of security (i.e., using the same size modulus), and

- having the verifier run in polynomial time.[1]

Problem 3: A Special PRF [10 points]. One can show that $F(k,x) = H(x)^k$ is a PRF, if H is modeled as a random oracle to a group where the discrete logarithm problem is hard. This PRF has many special properties. In this problem, we will explore two applications of this PRF.

(a) Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a PRF defined over groups $(\mathcal{K}, +)$ and $(\mathcal{Y}, \otimes)$, where $+$ and $\otimes$ are the respective group operations in those groups. We say F is *key-homomorphic* if it holds that

$$F(k_1 + k_2, x) = F(k_1, x) \otimes F(k_2, x).$$

Is the PRF $F(k,x) = H(x)^k$ defined with a random oracle $H : \mathcal{X} \to G$ (where G is a group of prime order p) a key-homomorphic PRF? Please prove your answer one way or the other.

(b) *Key rotation* is a common problem encountered in cloud storage: how to change the key under which data is encrypted without sending the keys to the storage provider? A naive solution is to download the encrypted data, decrypt it, re-encrypt it under a new key, and re-upload the new ciphertext. We will now see how this process can be made more efficient with a key-homomorphic PRF.

Suppose you have a ciphertext c made up of blocks $c_1, \ldots, c_N$ that corresponds to a message $m = (m_1, \ldots, m_N)$ encrypted under a key k_1 using a key-homomorphic PRF F in counter mode, i.e., $c_i = m_i \otimes F(k_1, i)$. Now you want to rotate to a key k_2. It turns out you can send the storage provider a single element $k_{\mathsf{update}} \in \mathcal{K}$ which it can then use to generate c', an encryption of m under k_2. Please tell us how you can compute k_{update} and how the storage provider can use k_{update} and c to compute c'. Prove that your protocol is correct (you need not prove security).

(c) An *oblivious PRF* is an interactive protocol between a client who holds a message x and a server who holds a key k. The protocol allows the client to learn the PRF evaluation $F(k,x)$ without the server learning anything about x. Oblivious PRFs are used in many advanced crypto protocols.

It turns out that there is an oblivious PRF protocol for the PRF $F(k,x) = H(x)^k$. Please show us how a client holding x and a server holding k can interact so that the client learns $H(x)^k$ while the server learns nothing about x. Prove that your protocol is correct (you need not prove security).

Problem 4: Random Oracle Commitments [5 points]. Let $H : \mathcal{M} \times R \to \mathcal{C}$ be a function where $\mathcal{R} = \mathcal{C} = \{0,1\}^\lambda$. Define $\mathsf{Commit}(m, r) = H(m, r)$. Show that Commit is binding and hiding if H is modeled as a random oracle.

Problem 5: Multi-Commitments [10 points]. Let $\mathbb{G}$ be a group of prime order q in which the discrete logarithm problem is hard. Let g and h be generators of $\mathbb{G}$. As we saw in class, the Pedersen commitment scheme commits to a message $m \in \mathbb{Z}_q$ using randomness $r \in \mathbb{Z}_q$ as $\mathsf{Commit}(m; r) := g^m h^r \in \mathbb{G}$. Moreover, we saw that Pedersen commitments are *additively homomorphic*, meaning that given commitments to m_1 and m_2, one can compute a commitment to $m_1 + m_2$. The "public parameters" associated with the Pedersen commitment scheme are the description of the prime-order group $\mathbb{G}$ and the group elements g and h.

[1] We don't use this optimization in practice since (1) Schnorr signatures are so much shorter and (2) the verification time here is polynomial, but still much larger than the normal RSA-FDH verification time. Still, it's a cool trick to know.

(a) Use $\mathbb{G}$ to construct an additively homomorphic commitment scheme $\mathsf{Commit}_n(m_1, \dots, m_n; r)$ that commits to a length-n vector of messages $(m_1, \dots, m_n) \in \mathbb{Z}_q$ using randomness $r \in \mathbb{Z}_q$. The output of the commitment should be short – only a single group element. You should specify both the public parameters of your scheme (which may be different from that of the basic Pedersen commitment scheme) as well as the description of the Commit_n function.

(b) Prove that your commitment scheme is perfectly hiding and computationally binding (assuming hardness of discrete log in $\mathbb{G}$).

(c) Show that if you are given a hash function $H\colon \mathbb{Z}_q \to \mathbb{G}$ (modeled as a random oracle), the public parameters for your construction from Part (a) only needs to consist of the description of the group $\mathbb{G}$ and the description of H. Argue *informally* why your construction is secure. You do *not* need to provide a formal proof. This problem shows that getting rid of public parameters is another reason why random oracles are useful in practice!

(d) **Extra Credit [5 points].** Prove formally that your construction from Part (c) is secure in the random oracle model.

Problem 6: On The Importance of Elliptic-Curve Point Validation [10 points]. In this problem, we will see that all parties in a cryptographic protocol must verify that adversarially chosen points are on the right curve, and failing to do so may break security. We exemplify this by considering a variant of elliptic-curve Diffie-Hellman key exchange in which the server uses the same key pair across multiple sessions. More specifically, let $E\colon y^2 = x^3 + Ax + B$ be an elliptic curve over $\mathbb{F}_p$, where $q := |E(\mathbb{F}_p)|$ is a prime number and $P \in E(\mathbb{F}_p)$ is a generator. The server holds a *fixed* secret key $\alpha \xleftarrow{R} \mathbb{Z}_q$ and advertises (e.g., in its TLS certificate) the corresponding fixed public key $\alpha P \in E(\mathbb{F}_p)$. A client connects to the server by choosing $\beta \xleftarrow{R} \mathbb{Z}_q$, computing $V = \beta P$, and sending V to the server. Both sides then compute the shared secret $W = \alpha\beta P$. For simplicity, we assume that the server then sends the message $E_s(W, \text{“Hello!”})$ to the client, where (E_s, D_s) is some symmetric cipher.

(a) Explain how the server can check that the point V it receives from the client is indeed in $E(\mathbb{F}_p)$.

Observe that the elliptic-curve group addition formulae are *independent of the parameter B* of the curve equation. In particular, for every curve $\hat{E}\colon y^2 = x^3 + Ax + \hat{B}$ for some $\hat{B} \in \mathbb{F}_p$, applying the formulae for addition in $E(\mathbb{F}_p)$ to any two points $\hat{V}, \hat{W} \in \hat{E}(\mathbb{F}_p)$ gives the point $\hat{V} \boxplus \hat{W} \in \hat{E}(\mathbb{F}_p)$.

(b) Suppose there exists a curve $\hat{E}\colon y^2 = x^3 + Ax + \hat{B}$ such that $|\hat{E}(\mathbb{F}_p)|$ is divisible by a small prime t (i.e., $t = O(\text{polylog}(q))$). Show that if the server *does not check* that $V \in E(\mathbb{F}_p)$, a malicious client can efficiently learn $\alpha \bmod t$. You may assume one can efficiently find a point of order t in $\hat{E}(\mathbb{F}_p)$.

(c) Use Part (b) to show how a malicious client can efficiently learn the secret key α, if the server *does not check* that $V \in E(\mathbb{F}_p)$. You may assume that if $\hat{B} \xleftarrow{R} \mathbb{F}_p$, then $|\hat{E}(\mathbb{F}_p)|$ is uniform in $\left[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}\right]$ and is efficiently computable. (As in Part (b), you may assume that whenever the order of a curve has a small prime factor t, one can efficiently find a point of order t on that curve.)

Optional Feedback [0 points]. Please answer the following questions to help us design future problem sets. You do not need to answer these questions, and if you would prefer to answer anonymously, please use this form. However, we do encourage you to provide us feedback on how to improve the course experience.

(a) What was your favorite problem on this problem set? Why?

(b) What was your least favorite problem on this problem set? Why?

(c) Do you have any other feedback for this problem set?

(d) Do you have any other feedback on the course so far?

Problem 7: Time Spent [1 point for answering]. How long did you spend on this problem set? This is for calibration purposes, and the response you provide will not affect your score.