CS 355: Topics in Cryptography **Spring 2018**

Problem Set 5

Due: June 8, 2018 at 5pm (submit via Gradescope)

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

https://crypto.stanford.edu/cs355/homework.tex

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **9KY4BB** to sign up. Note that Gradescope requires that the solution to each problem starts on a **new page**.

Problem 1: Conceptual Questions [12 points]. For each of the following statements, say whether it is TRUE or FALSE. Write *at most one sentence* to justify your answer.

(a) Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the sort of bilinear map we use to instantiate BLS signatures. Say (TRUE or FALSE) whether there are *known* efficient attacks on the following computational problems:

- CDH in $\mathbb{G}$
- DDH in $\mathbb{G}$
- CDH in $\mathbb{G}_T$
- DDH in $\mathbb{G}_T$

(b) Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q, and suppose there exists a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ of the sort we use to instantiate BLS signatures. Then, the group $\mathbb{G}$ is safe to use for standard two-party Diffie-Hellman key agreement:

That is, Alice sends $g^a \in \mathbb{G}$ for $a \xleftarrow{R} \mathbb{Z}_q$, Bob sends $g^b \in \mathbb{G}$ for $b \xleftarrow{R} \mathbb{Z}_q$, and Alice and Bob use $H(g^{ab})$ as their shared secret, where $H : \mathbb{G} \to \mathbb{G}$ is a hash function that we model as a random oracle.

(c) There exists succinct non-interactive arguments for the class BPP in the standard model (from plausible computational assumptions).

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

$$f(x) = 0 \bmod p \qquad \text{and} \qquad |x| \leq N^{\beta^2/\delta}.$$

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, q are identically-distributed primes), 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.

Extra credit [5 points]. Show that if you are given the middle $n/3$ bits of p you can also factor N.

(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 $N/2 \le M < N$, 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$\cdots$000.")

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. Show that the signer need only output signatures of $2n/3$ 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: SNARGs in the Random Oracle Model [12 points]. In this problem, we will show how to leverage probabilistically-checkable proofs (PCPs) to construct a succinct non-interactive argument (SNARG) in the random oracle model. We will rely on the following adaptation of the famous PCP theorem:

[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.

Theorem (PCP)**.** Let $\mathcal{L}$ be an NP language. There exists two efficient algorithms $(\mathcal{P}, \mathcal{V})$ defined as follows:

- The prover algorithm $\mathcal{P}$ is a deterministic algorithm that takes as input a statement $x \in \{0,1\}^n$, a witness $w \in \{0,1\}^h$ and outputs a bitstring $\pi \in \{0,1\}^m$, where $h, m = \text{poly}(n)$. We refer to π as the proof string.
- The verifier algorithm $\mathcal{V}^\pi$ is a *randomized* algorithm that takes as input a statement $x \in \{0,1\}^n$ and has oracle access to a proof string $\pi \in \{0,1\}^m$. The verifier reads $O(1)$ bits of π. The verifier chooses the bits it reads *nonadaptively* (i.e., they can depend on the statement x, but *not* on the values of any bit in π).

Moreover, $(\mathcal{P}, \mathcal{V})$ satisfy the following properties:

- **Completeness:** For all $x \in \mathcal{L}$, if w is a valid witness for x, then

$$\Pr[\pi \leftarrow \mathcal{P}(x, w) : \mathcal{V}^\pi(x) = 1] = 1.$$

- **Soundness:** If $x \notin \mathcal{L}$, then for all $\pi \in \{0,1\}^m$,

$$\Pr[\mathcal{V}^\pi(x) = 1] \leq 1/2.$$

(a) Let λ be a security parameter and let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant hash function. Use H to construct a commitment scheme $(\mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Commit}(x) \to c$: The commitment algorithm should take a message $x \in \{0,1\}^m$ and output a commitment $c \in \{0,1\}^\lambda$.
- $\mathsf{Open}(x, c, i) \to \sigma$: The open algorithm takes a message $x \in \{0,1\}^m$, a commitment $c \in \{0,1\}^\lambda$, and an index $i \in [m]$, and outputs an opening σ.
- $\mathsf{Verify}(c, i, b, \sigma) \to \{0,1\}$: The verification algorithm takes a commitment $c \in \{0,1\}^\lambda$, an index $i \in [m]$, a value $b \in \{0,1\}$, and an opening σ, and outputs a bit.

Show that your commitment scheme satisfies the following properties:

- **Completeness:** For all $x \in \{0,1\}^m$ and $i \in [m]$,

$$\Pr[c \leftarrow \mathsf{Commit}(x); \sigma \leftarrow \mathsf{Open}(x, c, i) : \mathsf{Verify}(c, i, x_i, \sigma) = 1] = 1.$$

- **Binding:** For all efficient adversaries $\mathcal{A}$, if we set $(c, i, (b, \sigma), (b', \sigma')) \leftarrow \mathcal{A}(1^\lambda)$, then

$$\Pr[b \neq b' \text{ and } \mathsf{Verify}(c, i, b, \sigma) = 1 = \mathsf{Verify}(c, i, b', \sigma')] = \text{negl}(\lambda).$$

- **Succinctness:** The commitment c output by Commit and opening σ output by Open satisfy $|c| = O(\lambda)$ and $|\sigma| = O(\lambda \log m)$.

In other words, the commitment scheme $(\mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ allows a user to succinctly commit to a long bitstring and then selectively open up a single bit of the committed string.

(b) Let $\mathcal{L}$ be an NP language (with statements of length n). Show how to construct a 3-round succinct argument system for $\mathcal{L}$ using your commitment scheme from Part (a). Specifically, your argument system should satisfy perfect completeness, have soundness error $\text{negl}(\lambda)$ against computationally-bounded provers, and the total communication complexity between the prover and the verifier should be $\text{poly}(\lambda, \log n)$. In particular, the communication complexity scales *polylogarithmically* with the length of the NP statement. [**Hint:** Use the PCP theorem.]

(c) Show how to convert your succinct argument from Part (b) into a SNARG in the random oracle model. [**Hint:** It suffices to invoke a theorem from class here for your security analysis.]

Problem 4: Generic Discrete Log [12 points]. You will need the "Birthday Bound" for these problems (see Boneh-Shoup, Appendix B.1). Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q.

(a) Many times this quarter, we have mentioned that there is a algorithm for discrete log that runs in time $\tilde{O}(\sqrt{q})$.[2] This algorithm, which is shockingly simple, is the best known algorithm for discrete-log in standard elliptic curve groups.

You are given a discrete-log challenge $h = g^x \in \mathbb{G}$. Show that by computing two tables of values: one of the form g^{r_i} for $r_i \xleftarrow{R} \mathbb{Z}_q$ and one of the form h^{s_j} for $s_j \xleftarrow{R} \mathbb{Z}_q$, you can recover the discrete log $x \in \mathbb{Z}_q$ in time $\tilde{O}(\sqrt{q})$.

This algorithm uses space $\tilde{\Omega}(\sqrt{q})$. A clever trick, due to Pollard, gets the space usage down to $O(\text{poly}\log q)$.

(b) The B-bounded discrete-log problem is the problem of recovering $x \in \{0, \ldots, B\}$, given $g^x \in \mathbb{G}$. That is, the group $\mathbb{G}$ is of order q, but we sample the exponent x from a range of size $B \ll q$. Modify your algorithm of part (a) to solve the B-bounded discrete-log problem in time $\tilde{O}(\sqrt{B})$.

Problem 5: Discrete Log in $\mathbb{Z}_p^*$ [12 points]. The algorithms in Problem 4 are *generic* algorithms, in that they work for every group. There are beautiful special-purpose algorithms for solving discrete log faster in $\mathbb{Z}_p^*$ (for prime p). We sketch some of the ideas behind these non-generic discrete-log algorithms, and we hope to explain why they have these funny sub-exponential running times.

Let $p = 2q + 1$ be a prime such that q is also prime. Let $g \in \mathbb{Z}_p^*$ be a generator of the order-q subgroup of $\mathbb{Z}_p^*$.

(a) Say that an integer is B-smooth if it factors into primes less than B. A good approximation is that:

$$\Pr_{x \xleftarrow{R} \mathbb{Z}_q} [(g^x \bmod p) \text{ is } B\text{-smooth}] \approx u^{-u} \qquad \text{where} \qquad u = \frac{\ln p}{\ln B}.$$

Let $B(p) = \exp(\sqrt{\ln p \cdot \ln\ln p})$. Show that the probability that a random number in $\mathbb{Z}_p$ is $B(p)$-smooth is at least $\Omega(1/B(p)))$.

(b) You are given:

- an integer $h = g^x \in \mathbb{Z}_p^*$,

[2] The big-$\tilde{O}$ and big-$\tilde{\Omega}$ notations are just like the normal big-O and big-Ω, except that they also suppress log factors. So, a running time of $\sqrt{q} \cdot \log^4 q$ is $\tilde{O}(\sqrt{q})$.

- all of the primes $(\pi_1, \ldots, \pi_k)$ of size at most $B(p)$, and
- the discrete logs $(\log_g \pi_1, \ldots, \log_g \pi_k)$ of these small primes modulo p (assume for simplicity that all of these discrete logs exist).

Give an algorithm that uses this information to recover $x \in \mathbb{Z}_q$ with constant probability in time $\text{poly}(B(p))$. You should show that your algorithm is correct and that it runs in the stated time.

(c) **Extra credit [6 points].** Show how to generate the discrete logs needed for part (b) in time $\text{poly}(B(p))$. This shows that it's possible to compute discrete logs in $\mathbb{Z}_p^*$ in time $\text{poly}(B(p)) = \exp(O(\sqrt{\log p \log\log p}))$.

Problem 6: Time Spent [3 points 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.

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?