Princeton University
Due: May 1, 2017, 11:59pm

Homework 7

1 Problem 1 (10 points)

(a) Let F_0, F_1 be two supposed one-way functions. Say you know that one of F_0, F_1 is a secure one-way function, but the other is not. However, you do not know which one. Construct a new one-way function F that is secure as long as at least one of F_0, F_1 are secure, but not necessarily both. Prove the one-wayness of F relying on just the security of F_0 *or* F_1

(b) Let $(\mathsf{Gen}_0, F_0, F_0^{-1}), (\mathsf{Gen}_1, F_1^{-1}, F_1^{-1})$ be two supposed trapdoor permutations, and suppose the domain for both trapdoor permutations is the same set $\mathcal{X}$ (since they are permutations, the co-domain is also $\mathcal{X}$). Suppose you are guaranteed that both are in fact permutations, but one of the two may be insecure. You do not know which one. Construct a new trapdoor permutation $(\mathsf{Gen}, F, F^{-1})$ that is secure as long as at least one of $(\mathsf{Gen}_0, F_0, F_0^{-1}), (\mathsf{Gen}_1, F_1, F_1^{-1})$ is secure, but not necessarily both.

(c) Let $(\mathsf{Gen}_0, \mathsf{Enc}_0, \mathsf{Dec}_0), (\mathsf{Gen}_1, \mathsf{Enc}_1, \mathsf{Dec}_1)$ be two public key encryption schemes. Suppose you are guaranteed that both are correct, in that decrypting an encryption of m recovers m. However, only one of the schemes is CPA-secure, and you don't know which. Construct a new encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that is CPA secure, provided at least one of the two schemes is CPA-secure.

(d) Let $(\mathsf{Gen}_0, \mathsf{Sign}_0, \mathsf{Ver}_0), (\mathsf{Gen}_1, \mathsf{Sign}_1, \mathsf{Ver}_1)$ be two digital signature schemes. Suppose you are guaranteed that both are correct, in that signatures will verify. However, only one of the schemes is CMA-secure, and you don't know which. Construct a new signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ that is CMA-secure, provided at least one of the two schemes is CMA-secure.

The constructions you present above are called *combiners*. With some extra work, the construction from part (a) can be turned into a *universal* one-way function: a one-way function that is secure, provided that *some* one-way function exists (but you don't need to know the one-way function). The same goes for the encryption combiner. Unfortunately, these universal constructions are of little use in practice.

2 Problem 2 (20 Points)

A *random self reduction* is a procedure which turns any instance of a problem into a *random* instance of the problem.

For example, let p a prime, and $\mathbb{G}$ a group of order p. Suppose you are given a discrete log instance $(g, h = g^a)$, and suppose that $g \neq 1$ (so that g is a generator). Choose a random $r, s \in \mathbb{Z}_p$ such that $r \neq 0$ and let $g' = g^r$ and $h' = h^r \times g^s$.

(a) Show that (g', h') is a random discrete log instance with $g' \neq 1$ (meaning g is a random generator, and h is a random group element).

(b) Suppose you give someone (g', h') and they give you a discrete log b such that $(g')^b = h'$. Explain how to recover the discrete log of h, namely a.

A random self reduction therefore shows that, if there is *any* discrete log instance that is hard, a *random* discrete log instance is also hard. This means that there are no "extra hard" instances, since no instance is harder than the average case. The fact that discrete log admits a random self reduction means we can actually base hardness of the *worst case* version of the problem, rather than an average case problem. It can also be used to amplify the success probability of attacks:

(c) Suppose you have a discrete log adversary A that runs in time t, and solves random discrete log instances with probability ϵ. You know nothing about A except this fact: in particular, maybe A is deterministic, or maybe A is randomized.

Show how to use A to derive an adversary A' which solves discrete log with probability 99/100, but is allowed to run in time about $O(t/\epsilon)$.

Part (c) shows that it is actually sufficient to assume that no time-bounded adversary can solve discrete log with high probability.

(d) Show such a random self reduction for DDH. That is, you are given a tuple $(g, u = g^a, v = g^b, w = g^c)$ where g is a generator, a and b are in $\mathbb{Z}_p$, and c is either ab mod p or different than ab. We will call the $c = ab$ case a DDH tuple.

You must come up with a new tuple (g', u', v', w') such that:

- If (g, u, v, w) is a DDH tuple, then (g', u', v', w') is a *random* DDH tuple (it should be random even if (g, u, v, w) is a fixed tuple)
- If (g, u, v, w) is *not* a DDH tuple, then (g', u', v', w') is a truly random tuple of group elements, conditioned on g' being a generator and the tuple being *not* a DDH tuple.

The transformation from (g, u, v, w) to (g, u', v', w') must be efficient: you cannot compute discrete logs as part of the transformation.

Note that for part (d), the following simple transformation will not work: (g, u^r, v, w^r). This is a DDH tuple if (g, u, v, w) was a DDH tuple, and isn't a DDH tuple if (g, u, v, w) isn't. However, for a fixed tuple (g, u, v, w), (g, u^r, v, w^r) is not random: for example, the third component is fixed as v. While this transformation won't work, it is a useful starting point to think about.

3 Problem 3 (20 points)

In class, we saw informally how obfuscation can be used to turn a MAC into a signature scheme. While in general, obfuscation (at least the kind that is sufficiently strong for crypto) is extremely inefficient. However, sometimes we can design a special purpose obfuscator that will work for certain constructions.

In this problem, we will consider the following types of programs. Let p be a prime, and $d > 0$. We will let $P(x)$ denote a degree-d polynomial defined over $\mathbb{Z}_p$. We will obfuscate programs of the form $T_P(x, z) = \begin{cases} 1 & \text{if } P(x) = z \\ 0 & \text{if } P(x) \neq z \end{cases}$.

(a) One way to obfuscate such programs in the following. Let $\mathbb{G}$ be a cyclic group of order p. Choose a random generator g. The description of the obfuscated program will consist of $(g, g^{a_0}, \ldots, g^{a_d})$, where a_i is the coefficient of x^i in P.

By the discrete log assumption, it is impossible to recover the description of P from the obfuscated program. Nonetheless, it is still possible to evaluate T_P. **Explain how, given $(g, g^{a_0}, \ldots, g^{a_d})$, to evaluate $T_P(x, z)$.**

(b) Use the above construction to construct a d-time signature scheme. Each signature should be a single element in $\mathbb{Z}_p$. The public key should be an obfuscated program as above, namely consisting of $d+2$ group elements. *[Hint: think about how we created d-time MACs]*

(c) Unfortunately, we do not know how to prove the above construction is d-time secure under the definition seen in class. However, we will consider a weaker definition. We will consider a non-interactive CMA attack model, where the adversary's d chosen message queries must be made all at once, and before the adversary sees the public key. That is, the adversary submits d messages $m_1, \ldots, m_d$, and gets as response the public key and the d signatures on $m_1, \ldots, m_d$. Finally, the adversary chooses an $m^* \notin \{m_1, \ldots, m_d\}$ and tries to forge a signature on m^*. We will say that a scheme is (t, d, ϵ) secure if any adversary running in time at most t has at most a probability ϵ of forging a signature on m^*.

Show that if the discrete log assumption holds on $\mathbb{G}$, then your scheme from part (c) is secure under a non-interactive CMA attack.

For this part, the following fact from Lagrange interpolation will be helpful: Let P be a degree d polynomial with coefficients $a_0, \ldots, a_d$. For any list of $d+1$ inputs $x_0, \ldots, x_d$, it is possible to efficiently compute values $r_{i,j} \in \mathbb{Z}_p$ such that $a_i = \sum_j r_{i,j} P(x_j)$.

(d) Explain how to extend the scheme to messages in $\mathbb{Z}_p^\ell$. The signatures should still be in $\mathbb{Z}_p$

(e) Explain why the scheme is not $d+1$-time secure

4 Problem 4 (10 points)

Suppose Alice and Bob each have signed the same message m (with their own secret keys), obtaining signatures σ_A, σ_B, which they have given to Charlie. Charlie now wished to prove to Donald that Alice and Bob signed m. Clearly, he can simply give σ_A, σ_B to Donald. However, in some situations, Charlie would like to provide a single signature $\sigma_{A,B}$ which proves to Donald that both Alice and Bob signed m.

More generally, aggregate signatures allow for the following: if n users $1, \ldots, n$ have signed the same message m, producing n signatures $\sigma_1, \ldots, \sigma_n$, anyone with these n signatures can construct an aggregate signature $\sigma_{\{1,\ldots,n\}}$ that attests to all n users signing the message m. The size of $\sigma_{\{1,\ldots,n\}}$ should be independent of the number of users.

Show that the signature scheme from **Problem 3** can be aggregated very easily. That is, assume all n users have public/secret keys chosen according to the scheme, all using the same group $\mathbb{G}$. We will additionally require they all use the same generator g. To aggregate several signatures on the same message, simply add the signatures together in $\mathbb{Z}_p$.

(a) Explain how, given the public keys for the n users, to verify the aggregate signature $\sigma_{\{1,\ldots,n\}} = \sigma_1 + \cdots + \sigma_n$

(b) Explain why, if user i did not sign m, that it is computationally infeasble to construct an aggregate signature which verifies