Massachusetts Institute of Technology — Handout 5
6.857: Applied Cryptography and Security — April 4, 2022
Professors Ronald L. Rivest and Yael Tauman Kalai — **Due:** April 19, 2022

Problem Set 4

This problem set is due on *Tuesday, April 19, 2022* at **11:59 PM**. Please note our late submission penalty policy in the course information handout. Please submit your problem set, in PDF format, on Gradescope. *Each problem should be in a separate page.* Have **one and only one group member** submit the finished problem writeups. Please title each PDF with the Kerberos of your group members as well as the problem set number and problem number (i.e. *kerberos1_kerberos2_kerberos3_pset1_problem1.pdf*).

You are to work on this problem set in groups. For problem sets 1, 2, and 3, we will randomly assign the groups for the problem set. After problem set 3, you are to work on the following problem sets with groups of your choosing of size three or four. If you need help finding a group, try posting on Piazza or email `6.857-staff@mit.edu`. You don't have to tell us your group members, just make sure you indicate them on Gradescope. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must be provided as a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we may distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on your homework submission.

Problem 4-1. Optimal Asymmetric Encryption Padding (OAEP)

In class, we learned about the (vanilla) RSA encryption scheme, which is a deterministic encryption scheme, and as such not CPA (or CCA) secure. We saw that one can make the RSA scheme CCA secure by adding the OAEP encoding to the message. (See `https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding`).

(a) This encoding seems quite complicated. Consider the following simpler proposal:

To encrypt a message m, choose a random $r \in \{0,1\}^{k_0}$, and send r together with an RSA encryption of $m \cdot G(r)$. Assume G is a random oracle that takes r to $\mathbb{Z}_n^*$ (where n is the public key), and assume the RSA problem is hard. Suppose the message space is $\mathbb{Z}_n^*$.

Is this scheme CPA secure, CCA secure or neither? Explain your answer.

(b) Consider the following alternative variation, which is closer to the OAEP encoding, but significantly simpler: To encrypt a message $m \in \{0,1\}^{k-k_0}$, choose a random $r \in \{0,1\}^{k_0}$, and send an RSA encryption of $(m \oplus G(r), r)$. Assume G is a random oracle that maps $\{0,1\}^{k_0}$ to $\{0,1\}^{k-k_0}$, where the RSA modulus n is a $k+1$-bit number, and assume the RSA problem is hard.

Is this scheme CPA secure, CCA secure or neither? Explain your answer.

For your CCA analysis, consider that the adversary's decryption oracle is the RSA decryption oracle. Namely, it takes a ciphetext C and outputs $C^d \pmod n$, where $d \times e \equiv 1 (\bmod\ \phi(n))$.

Problem 4-2. RSA Variants In this problem, we will be looking at various variants of RSA. Recall that RSA consists of the following algorithms:

- Gen(1^λ) : find two primes p, q of length λ, compute $n = p \cdot q$, sample some e in $\mathbb{Z}^*_{\varphi(n)}$, and compute $d = e^{-1}$ $(\texttt{mod}\varphi(n))$. Then, the public key is $pk := (n, e)$, the secret key is $sk := (d, p, q)$, and the message and ciphertext spaces are $\mathbb{Z}_n$.

- Enc(pk, m) : output m^e $(\texttt{mod}\ n)$ as the ciphertext.
- Dec(sk, c) : output c^d $(\texttt{mod}\ n)$ as the message.

We will now tweak RSA in various ways and analyze the (in)security of these variants. Note that each problem part is independent of the other ones (i.e., they all modify the basic version of RSA instead of each part modifying the prior part). For each of your yes/no answers below, **justify your answer**.

(a) We modify the Gen algorithm such that, to sample p and q, we first choose p randomly, and then compute $q = \texttt{nextprime}(H(p))$, where H is a hash function (modelled as a random oracle) mapping n-bit strings to n-bit strings, and `nextprime` returns the next prime greater than its input. The rest of Gen is unchanged. Is the RSA problem still hard for this scheme? I.e., given a ciphertext and the public key, is it hard to find m?

(b) Recall that RSA is not CPA-secure, since it is deterministic. Consider a variant of RSA where, instead of having a fixed encryption exponent e, we sample it every time we encrypt. That is, the public key is now just n, and the ciphertext is the pair $(e, m^e\ (\texttt{mod}\ n))$ (e is sampled uniformly at random (from Z_n, say) on every encryption). Is this variant CPA secure?

(c) Consider a variant of RSA where the public key is now $pk := (n, e, p + q)$. You can think of this as an implementation of RSA that "leaks" $p + q$ to the adversary. Is the RSA problem still hard for this scheme?

Problem 4-3. Public Key Infrastructure

For this problem we are giving you fifteen X.509 certificates that form a mini PKI. These certificates are of the same format that you would see live for websites on the internet, but are, unsurprisingly, not signed by a real certificate authority. Instead, we are designating a local *root store* for you that contains three certificates: `cert1`, `cert2`, and `cert3`. This root store, much like the one that exists in your browser, tells you which certificates should be trusted to be the *root* of a chain of certificates. A root certificate, like the name implies, does not have an issuer but is self-signed. Because *anyone* can generate a self-signed certificate, the root store is used to designate a set of root certificates that are believed to be trustworthy.

The certificates we give you for this problem have the following fields:

- `subject_name`: the name of the entity that owns this certificate
- `issuer_name`: the name of the entity that issued (signed) this certificate
- `public_key`: the (RSA) public key for this certificate
- `serial_number`: unique identifier for certificate (not used in this question)
- `not_valid_before`: certificate is only valid *after* this date
- `not_valid_after`: certificate is only valid *before* this date
- `signature`: a signature on a hash of the certificate using the issuer's (RSA) private key

Your task for this problem is to determine the relationship between the certificates we give you and to decide whether they should be trusted. We are providing you with code to read the certificates we give you and validate their signatures[1]. Note that signature verification does not tell you anything about *other* fields of the certificate, only that it was indeed signed by the key that you provided to the verification function.

(a) Please write out or diagram the relationship between all 15 of the certs. As an example, if `certA` is a self-signed root cert that is used to sign `certB` you might indicate that as: `certA` → `certB`. If `certA` is used to sign more than one certificate you are welcome to write that as two chains: `certA` → `certB` and `certA` → `certC`. Note that this is just a suggestion and you are welcome to indicate relationships between certificates however you like (as long as it's clear).

[1]This code also includes a function to build certificates, though you will not need it to solve this problem.

(b) For every leaf certificate (a certificate that is not used to sign any other certificates) please indicate whether you think this certificate should be considered trustworthy. For certificates that you think are **not** trustworthy, please explain why[2]. When determining trustworthiness, consider both:

1. whether the contents of any of the certificate's fields are invalid according to the definitions we provided above
2. whether you, if visiting the domain (`example.com`) listed in the certificate's subject field, might have any reason to believe that you were not accessing the *real* `example.com`, but have instead been presented with a certificate from an imposter

(c) The certificates we provide for you are very abbreviated compared to what you might find in the wild. Find a live certificate for some website and look at the fields it has. Pick one element of the real certificate that isn't present in our examples and explain why that field might be useful in practice.

[2]Some certificates might be untrustworthy for multiple reasons. Please include all of them in your explanation!