Cryptography

Problem Set

1. The problem set (as on 3/12/19) has **23 questions** with a total score of **295 points**.

2. A lot of the problems refer to [Ros19], which is the 21 Mar 2019 version on its webpage. If you need a copy of this version (and the Joy of Cryptography draft gets updated), you may get one from the slack group for the course, or contact me.

3. You are welcome to discuss with other classmates **as long as these discussion are reasonable**; these are not meant for one to solve the problem for the other. You are eventually expected to find and write your own solutions.

   If you do discuss, you are expected to explicitly mention who you discussed with and which parts of your solution came from these discussions.

4. Solutions may be submitted either as a pdf, or handwritten. If you are creating a pdf, **please do not print**! You can email the pdf to me instead. If you do send me a pdf via email, please make sure the subject of the email includes the word "Crypto", and use the following scheme for the filename:

   `firstname_lastname_YYYY-MM-DD_n.pdf`

   For example, if I were to submit an assignment on 19/09/2019, then I'll name mine it as

   `ramprasad_saptharishi_2019-09-19_1.pdf`.

   If I make a minor correction and resubmit it on the same day, it would be named

   `ramprasad_saptharishi_2019-09-19_2.pdf`

   (If you have already submitted problem set solutions via different file names, don't worry about those. Just use this scheme for all subsequent submissions; makes my life a little easier :-))

Deadline: 9th September 2019

Question 1. [5 points] [Ros19, Problem 1.7]. It might be helpful to have the following `ascii` conversions

`alpha`: 0110000101101100011100000110100001100001

`bravo`: 0110001001110010011000010111011001101111

`delta`: 0110010001100101011011000111010001100001

`gamma`: 0110011101100001011011010110110101100001

Question 2. [10 points] [Ros19, Problem 2.2]

Question 3. [10 points] [Ros19, Problem 2.5]

Question 4. [10 points] [Ros19, Problem 2.14]

Question 5. [5 points] [Ros19, Problem 4.13]

Question 6. [15 points] [Ros19, Problem 5.8]

Deadline: 12th September 2019

Question 7. [10 points] [Ros19, Problem 6.5]

Question 8. [15 points] [Ros19, Problem 6.6]

Deadline: 19th September 2019

Question 9. [10 points] Let F be a secure PRF. Show why a 2-round Feistel structure using F does not yield a PRP by exhibiting an attacker.

Question 10. [15 points] Suppose $F : \{0,1\}^\lambda \times \{0,1\}^b \to \{0,1\}^b$ be a secure PRP. Show that the CBC-mode of encryption with this PRP is CPA$ secure. Recall that the encryption algorithm is given by:

```
function Enc(k,$(m_1, \ldots, m_\ell)$)
    $c_0 \in_R \{0,1\}^b$
    for $i = 1, \ldots, \ell$ do
        $x_i = c_{i-1} \oplus m_i$
        $c_i = F(k, x_i)$
    end for
    return $(c_0, \ldots, c_\ell)$
end function
```

Hint: The following hybrid library may be helpful.

```
$k \in_R \{0,1\}^\lambda$

function CText($m_1, \ldots, m_\ell$)
    $c_0 \in_R \{0,1\}^b$
    for $i = 1, \ldots, \ell$ do
        $x_i = c_{i-1} \oplus m_i$
        $c_i = \text{Lookup}(x_i)$
    end for
    return $(c_0, \ldots, c_\ell)$
end function
```

$\circ$

```
bad = 0

function Lookup($x$)
    if $T[x]$ is undefined then
        $T[x] \in_R \{0,1\}^b \setminus \text{Values}(T)$
    else
        bad = 1
        $r \in_R \{0,1\}^b$
        return r
    end if
    return $T[x]$
end function
```

Can you show that the probability that bad is set to 1 is negligible?

Deadline: 23rd September 2019

Question 11. [10 points] [Ros19, Problem 6.6]. Show that a 3-round Feistel cipher, with any PRF, *cannot* be a strong PRP.

Question 12. [15 points] [Ros19, Problem 9.5]. Show how a padding-check oracle (for CBC-mode encryption, with X.923 padding (the one that ends with `01` or `00 02` or `00 00 00 03` etc.)) can be used to **generate a valid encryption** of *any* chosen plaintext, under the same (secret) key that the padding-check oracle uses.

In this problem, you are *not* given access to an encryption subroutine, or *any* valid ciphertexts — only the padding-check oracle. All you are allowed is to send strings of your choice (say x) to the padding-check oracle and the oracle would tell you if the decryption of x is properly padded or not. Given this, the goal is to generate a valid encryption of any chosen plain text (under the same key).

Question 13. [10 points] [Ros19, Problem 9.7] Suppose you have an encryption Enc with message space $\{0,1\}^n$. Define the encryption $\text{Enc}^{(2)}$, with message space $\{0,1\}^{2n}$ by just encrypting the left and right halves and concatenating the ciphertexts:

$$\text{Enc}^{(2)}(m_1 m_2) = c_1 c_2 \quad \text{where, } c_1 = \text{Enc}(m_1)\ ,\ c_2 = \text{Enc}(m_2).$$

1. Show that if Enc is CPA-secure, then so is $\text{Enc}^{(2)}$.
2. Show that, even if Enc is CCA-secure, $\text{Enc}^{(2)}$ is *not* CCA-secure by producing an attacker.

Deadline: 3rd October 2019

Question 14. [10 points] [Ros19, Problem 10.8] In class, when we studied the "Encrypt-then-MAC" method to make a CPA-secure encryption scheme CCA-secure, we stressed that we sent the MAC of the ciphertext and not the message. (You can check for yourself why that is the case.)

But consider the following variant called "Encrypt-then-Encrypted-message-MAC".

```
function $\textsc{Enc}'(k_{\mathrm{Enc}}, k_{\mathrm{MAC}}, m)$
    $c = \mathrm{Enc}^{\mathrm{CPA}}(k_{\mathrm{Enc}}, m)$
    $t = \mathrm{MAC}(k_{\mathrm{MAC}}, m)$
    $c' = \mathrm{Enc}^{\mathrm{CPA}}(k_{\mathrm{Enc}}, t)$
    return $(c, c')$
end function
```

The decryption is defined in the natural way. Show that this scheme does not result in a CCA-secure scheme by exhibiting an attacker.

Question 15. [15 points]

For the standard "Encrypt-Then-MAC" scheme, we use two separate keys for the encryption scheme and the MAC. The following is an attempt to derive it from a single key using a pseudorandom function $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$.

```
function $\textsc{Enc}'(k, m)$
    $k_{\mathrm{Enc}} = F(k, 00\cdots0)$
    $k_{\mathrm{MAC}} = F(k, 11\cdots1)$
    $c = \mathrm{Enc}^{\mathrm{CPA}}(k_{\mathrm{Enc}}, m)$
    $t = \mathrm{MAC}(k_{\mathrm{MAC}}, c)$
    return $(c, t)$
end function
```

Does this give you a CCA-secure scheme? Justify your answer by either exhibiting an attacker or proving CCA-security.

Question 16. [10 points] [Ros19, Problem 11.13] Let $H : \{0,1\}^n \to \{0,1\}^n$ be a collision resistant hash function (we are ignoring the salt just for ease of writing/reading. And yes, input and output length being n is perhaps silly, but let's work with this). Consider the following CBC-MAC type of function to create a candidate hash function for larger input lengths (but same output length):

```
**function** $H^*(m_1, \ldots m_\ell)$
    // each $m_i$ is $n$-bits long
    $y_0 = 0^n$ // or this could even be a part of the salt
    **for** $i = 1, \ldots, \ell$ **do**
        $y_i = H(m_i \oplus y_{i-1})$
    **end for**
    **return** $y_\ell$
**end function**
```

Show that the function H^* is not collision resistant by showing how you can construct a collision.

Deadline: 18th October 2019

Question 17. [15 points] Prove or disprove the following:

> Suppose $f : \{0,1\}^n \to \{0,1\}^n$ is a one-way function. Then, the function $f' : \{0,1\}^n \to \{0,1\}^n$ defined as $f'(x) = f(x) \oplus x$ is also a one-way function.

(Hint: Say $f(x,y) = (g(x) \oplus y, y)$)

Question 18. [15 points] Let us assume that injective one-way functions exist. In class we studied the notion of hardcore predicates for injective one-way functions, and we saw that the most significant bit of the RSA function is a concrete hardcore predicate. This question explores if the same works for any function. That is, for any injective one-way function, perhaps one of the input bits x_i is itself a hardcore predicate? Prove or disprove the following:

> Assume that injective one-way functions. Then, there exists an injective one-way function $f : \{0,1\}^n \to \{0,1\}^*$ such that, for every $i \in [n]$, there is an efficient algorithm A_i such that
>
> $$\Pr_x [A_i(f(x)) = x_i] \gg \frac{1}{2}.$$
>
> That is, there are injective one-way functions such that none of the coordinate functions $\{h_i(x) = x_i \ : \ i \in [n]\}$ are hardcore predicates for it.

Question 19. [10 points] Let $f : \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable, injective function. Prove or disprove the following:

> If $h : \{0,1\}^n \to \{0,1\}$ is a hardcore predicate for f, then f is a one-way function

Answer the same question in the setting when f is not injective. That is, prove or disprove the following:

> Let $f : \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable (possible non-injective) function. Then, if $h : \{0,1\}^n \to \{0,1\}$ is a hardcore predicate for f, then f is a one-way function.

Deadline: 18th Nov 2019

Question 20. In class, we saw a construction of a digital signature scheme that was secure against chosen message attacks from a scheme that was one-time secure based on the "tree" construction. In the scheme we saw in class, the signer was *stateful* in the sense that needed to "remember" all the public keys he had to reveal in the course of previous signatures. In this problem, we shall attempt to come up with a scheme that avoids this using pseudorandom functions.

For simplicity, let us assume that we are signing messages of length n, the on-time signature scheme S uses a public key of length at most $(n/2)$ and signs messages of length n (so that internal nodes, that have to sign two public keys, end up signing length n messages). Furthermore, assume that the signing scheme S has the property that the key generation algorithm Gen_S uses n random bits to generate two (public key, secret key) pairs, and uses n random bits for each signature it produces (the scheme based on OWFs was deterministic, but one might imagine signatures generated via a randomized algorithm as well).

Assume that $F : \{0,1\}^\lambda \times \{0,1\}^{n/2} \to \{0,1\}^n$ is a secure pseudorandom function. Consider the following signature scheme similar to the "tree" scheme discussed in class.

```
function Gen( )
    $(\mathrm{pk}_\varepsilon, \mathrm{sk}_\varepsilon) = \mathsf{Gen}_S()$
    $k_1, k_2 \in_R \{0,1\}^\lambda$
    return $(\mathrm{pk}_\varepsilon, (\mathrm{sk}_\varepsilon, k_1, k_2))$
end function

function Sign($m, \mathrm{sk}_\varepsilon, k_1, k_2$)
    for $i = 0, \cdots, n-1$ do
        $m_{[i]}$ = prefix of first $i$ bits of $m$
        $r_1 = F(k_1, \mathrm{pk}_{m_{[i]}})$ , $r_2 = F(k_2, \mathrm{pk}_{m_{[i]}})$
        Generate two (public-key, secret-key) pairs $(\mathrm{pk}_{m_{[i]}0}, \mathrm{sk}_{m_{[i]}0})$ and
$(\mathrm{pk}_{m_{[i]}1}, \mathrm{sk}_{m_{[i]}1})$ using $\mathsf{Gen}_S$ with randomness $r_1$.
        Let $\sigma_i' = S(\mathrm{pk}_{m_{[i]}0} || \mathrm{pk}_{m_{[i]}1}\ ,\ \mathrm{sk}_{m_{[i]}})$, using randomness $r_2$
        $\sigma_i = (\sigma_i', \mathrm{pk}_{m_{[i]}0}, \mathrm{pk}_{m_{[i]}1})$
    end for
    Let $r = F(k_2, \mathrm{pk}_m)$
    $\sigma_n = S(m\ ,\ \mathrm{sk}_m)$, using randomness $r$
    return $(\sigma_0, \ldots, \sigma_n)$
end function
```

The verification routine is exactly as it was discussed in class.

[15 points] Show that the above digital signature scheme is secure against chosen message attacks (assuming of course that the scheme S is one-time secure and F is a secure pseudorandom function). [You does not have to be extremely verbose here but the answer should be sufficiently formal and convincing to me.]

Question 21. In class, we saw the Merkle-Damgård transform that, given a collision-resistant hash function (CRHF) $H = \left\{h_s : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}\right\}_s$ to build a CRHF with larger input lengths. Another way to achieve this is via what are called Merkle trees defined as:

$$\begin{aligned} &h_s^0 : \quad x \mapsto x \\ &h_s^i : \{0,1\}^{2^i \ell} \to \{0,1\}^{\ell} \\ &h_s^i : (x,y) \mapsto h_s(h_s^{i-1}(x) || h_s^{i-1}(y)) \quad , \quad \text{for all } i > 0 \end{aligned}$$

[10 points] Show that, for any $i = O(\log n)$, the function $\{h_s^i\}_s$ is a CRHF if h is a CRHF.

Question 22. In class we saw a zero knowledge protocol for 3-colouring, assuming the existence of commitment schemes, However, that protocol required $O(|E|)$ rounds to obtain soundness error at most $1/2$ due to sequential repetition. However, consider the following protocol for the *Hamiltonian Cycle* problem where the language L_{Ham} is the set of all graphs G such that there is a Hamiltonian cycle in G (a cycle that passes through every vertex exactly once). Consider the following protocol for L_{Ham}.

> The Prover (provided the input G, and the cycle C) choose a random permutation $H = \sigma(G)$ of the graph G and sends the verifier a commitment Λ to entries of the adjacency matrix of H.
>
> The verifier chooses a bit $b \in_R \{0, 1\}$, uniformly at random, and sends that to prover.
>
> If $b = 0$, the prover opens all committed bits and reveals σ so that the verifier can check if the prover had honestly committed to a permutation of G.
>
> If $b = 1$, the prover only opens the bits corresponding to the cycle C so that the verifier can check that H indeed has a Hamiltonian cycle.

[15 points] Prove that, assuming the commitment scheme is secure, this protocol has perfect completeness, soundness error at most $(1/2)$ and is computational zero knowledge.

Question 23. It is known that, in the single prover setting (which was the only setting we discussed in class), a k-fold parallel/concurrent repetition of a protocol with soundness error ε results in a protocol with soundness error ε^k. You may assume this result for the purposes of this question.

1. **[10 points]** Recall that Σ-protocol is a Proof Of Knowledge protocol for a relation $R_L(x, y)$, with special soundness and Honest-Verifier-Zero-Knowledge satisfies the following template:

   - Prover sends a "commitment" a,
   - Verifier sends a "challenge" c chosen uniformly at random from a challenge set $\mathcal{C}$,
   - Prover sends a response r. Verifier checks if $P(a, c, r)$ is true, for some deterministic polynomial time function P.

   It satisfies the special soundness property in the sense that there is an efficient algorithm E that, given any pair (a, c, r) and (a, c', r') such that $P(a, c, r) = P(a, c', r') =$ True and $c \neq c'$, can compute a y such that $R_L(x, y) =$ True.

   Furthermore, there is a simulator Sim that on input x and a random $c \in \mathcal{C}$ outputs a triple (a, c, e) such that (a, c, e) is distributed according the same distribution as the transcript of the honest prover with the honest verifier.

   [10 points] Show that a k-fold parallel repetition of such a Σ-protocol with the above properties also results in a Σ-protocol with all the above properties.

2. A slight variant of parallel repetition is the notion of "concurrent repetition" where the verifier is running k protocols together and can choose to "pause" one run, execute a few steps of another run, etc. Consider the following Proof Of Knowledge protocol for discrete-log.

   - Verifier (on input (x, g, p)) attempts to randomly guess a y' and checks if $g^{y'} = x \bmod p$. Verifier sends 1 to the Prover (who has inputs (x, y, g, p)) if he was lucky, and sends 0 otherwise.
   - If Verifier had sent 1, then the Verifier initiates a Proof-Of-Knowledge protocol for discrete log (say Schnorr's protocol we saw in class) to tell the Prover that he did find the discrete log of x. If the prover is convinced, then the prover sends y to the verifier. If the prover is *not* convinced, then the prover sends nothing and the verifier rejects.
   - If the Verifier had sent 0, then the Prover initiates a Proof-Of-Knowledge protocol for discrete log (say Schnorr again) and the verifier accepts if he is convinced.

   [10 points] Show that the above protocol is indeed a Zero Knowledge Proof of Knowledge.

   [10 points] Show that a malicious Verifier, with just two concurrent runs of this protocol, can completely recover y.

References

[Ros19] Mike Rosulek. Joy of cryptography, March 2019. Lecture notes for CS427 in Oregon State University. URL: http://web.engr.oregonstate.edu/~rosulekm/crypto/.

A Typesetting libraries (some hacky way)

I've created a small macro `\LibBlock{}` that takes an algorithm pseudocode (syntax as in the package `algpseudocode`) and puts it inside an inline tikz box. The following is an example of using this macro. The macro requires the use of packages `tikz, varwidth, algpseudocode`. If any of you have a nicer way of typesetting it, please do let me know.

$\mathcal{L}^{\text{pOTP}}_{\text{left}}$:

```
function Eavesdrop($m_L, m_R$)
    $s \leftarrow \{0,1\}^\lambda$
    $z = G(s)$
    $c = z \oplus m_L$
    return $c$
end function
```

$\mathcal{L}_{\text{hyb}_1}$:

```
function Eavesdrop($m_L, m_R$)
    $z = \text{Query}()$
    $c = z \oplus m_L$
    return $c$
end function
```

$\circ$

```
function Query()
    $s \leftarrow \{0,1\}^\lambda$
    $r = G(s)$
    return $r$
end function
```

$\mathcal{L}_{\text{hyb}_2}$:

```
function Eavesdrop($m_L, m_R$)
    $z = \text{Query}()$
    $c = z \oplus m_L$
    return $c$
end function
```

$\circ$

```
function Query()
    $r \leftarrow \{0,1\}^n$
    return $r$
end function
```

$\mathcal{L}^{\text{OTP}}_{\text{left}}$:

```
dummyglobal $= 42$

function Eavesdrop($m_L, m_R$)
    $z \leftarrow \{0,1\}^n$
    $c = z \oplus m_L$
    return $c$
end function
```