CS-E4340 Cryptography: Exercise Sheet 10

Submission deadline: November 28, 2022, 11:30, via MyCourses

Each exercise can give up to two participation points, 2 for a mostly correct solution and 1 point for a good attempt. Overall, the exercise sheet gives at most 4 participation points. We encourage to **choose** exercises which seem **interesting** and/or adequately challenging to you.

Exercise Sheet 10 is intended to help...

(a) ...understand oblivious transfer (Ex. 1).
(b) ...analyze secret-sharing protocols (Ex. 2).
(c) ...understand how to run a secret-sharing protocol (Ex. 3).
(d) ...familiarize yourself with simulation-based definitions (Ex. 4).

Exercise 1 (Simple OT)**.** One very useful primitives from secure multi-party computation is *oblivious transfer (OT)*, where receiver and sender have the following inputs and outputs

Sender Inputs:	secret strings m_0, m_1	**Receiver inputs:**	bit b
Sender Output:	none	**Receiver output:**	string m_b

Security of an OT protocol guarantees that

(1) the receiver learns nothing about m_{1-b} and that
(2) the sender learns nothing about the receiver's bit b.

Figure 1 describes a simple OT protocol which is similar to the Diffie-Hellman key exchange (first described in `https://eprint.iacr.org/2015/267.pdf`).
Task: Explain why (2) holds information-theoretically, i.e., the sender learns nothing about the receiver's secret bit. In addition, elaborate on the problems which need to be difficult for (1) to hold, i.e., that the receiver does not learn any information about the sender's secret bit.

Solution 1. To understand why (2) holds, notice that the Sender sends both secrets s_0, s_1 - only each secret is encrypted under a different key, meaning it has no information about b. Now the crucial security aspect is that (1) holds, that is, even with the Sender sending both secrets, the Receiver should only be able to decrypt one of them, more specifically the secret corresponding to its choice b. It is useful to compute k_0, k_1 in both possible cases, when $b = 0, b = 1$. For simplicity we ignore the application of the hash function H on our key agreements; it is not relevant until we try to derive symmetric keys. Now let us turn to computing the following table:

	$\mathbf{b = 0}$	$\mathbf{b = 1}$
k_0	g^{yx}	g^{x^2+xy}
k_1	g^{xy-x^2}	g^{xy}

Note that the Receiver always computes $k_b = X^y = g^{xy}$, so in the end the Sender should somehow ensure that the right secret s_b is encrypted under g^{xy} (for the Receiver to be able to recover it). Observing our table, we see that the Sender ensures this property ($k_b = g^{xy}$ for $b \in \{0,1\}$), since $b = 0 \implies k_0 = g^{xy}$ and $b = 1 \implies k_1 = g^{xy}$. We have now basically verified the *correctness* of the protocol, but what about its security? Note that the calculation of the keys k_{1-b} (on the anti-diagonal of the table) result in values that the Receiver would have a harder time deriving. To be more precise, we believe that the Receiver would not be able to derive those values g^{xy-x^2} and g^{x^2+xy} (efficiently) given only $g, X = g^x, Y = g^y$ and $Xg^y = g^{x+y}$. Technically, we assume that any efficient adversary cannot even distinguish between values g^{xy-x^2} or g^{x^2+xy}, with g^r where r is a random group element.

Exercise 2 (Penguin C Attack)**.** In the lecture, we discussed a secure communication protocol where penguin A, B, and C each hold a secret bit b_A, b_B, and b_C, respectively, and want to compute $b = b_A \wedge b_B \wedge b_C$ without revealing to each other anything except for the result b.
Task: Explain how penguin C can distinguish between the case that $b_A = b_B = 0$ and $b_A = b_B = 1$, regardless of whether C's own bit b_C is 0 or 1.

Solution 2. In the case that $b_A = b_B = 1$, we have $r_0^A = r_1^A$ and $r_0^B = r_1^B$. Then the second message C recieves from B will be

$$m = r_0^A \oplus r_0^B \oplus r_0^C \oplus r_0^A \oplus r_0^B = r_0^C.$$

In the case that $b_A = b_B = 0$, there is a high probability that the message differs from r_0^C. $m = r_0^C$ if and only if

$$r_0^A \oplus r_0^B \oplus r_1^A \oplus r_1^B = 0.$$

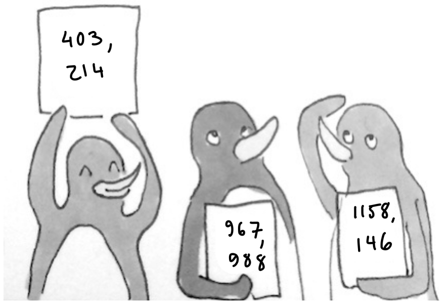
This, in turn, is true if and only if an even number out of $r_0^A[i], r_1^A[i], r_0^B[i]$ and $r_1^B[i]$ are equal to one, for each $i = 1, 2, \ldots, \lambda$ (here $r[i]$ denotes the ith bit of r). Since all bits are random, this happens with probability $\frac{1}{2}$. Therefore we can conclude that

$$\Pr_{b_A = b_B = 0}\left[m = r_0^C\right] = \left(\frac{1}{2}\right)^\lambda$$

which is negligible in λ.

Exercise 3 (Shamir's Secret Sharing)**.** Study the section on Shamir's Secret Sharing in the lecture notes and answer the following:

- why the adversary cannot learn the secret when only $t-1$ shares are leaked?
- is the scheme secure only against polynomial time (polynomial in the bit-length of the secret) adversaries or can we allow exponential runtime adversary? Why?
- Consider Shamir's Secret Sharing with threshold $t = 3$ and $q = 1361$. Suppose Penguin A has share (403, 214), Penguin B has share (967, 988) and Penguin C has share (1158, 146), where the first value of the pair is x and second value is y.

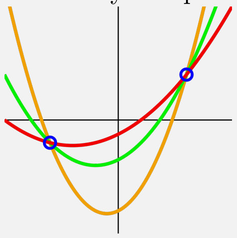


What is the secret? Please explain all the steps.

Hint: You may use any technique for polynomial interpolation that you are familiar with and you may tools such as Wolfram Alpha to invert the Vandermonde Matrix or use some other tool to solve Gaussian elimination.

Hint: The secret is our most favorite code.

Solution 3. – Because there are many degree $t-1$ polynomials that match the given $t-1$ points, hence there is no unique value a_0. Consider for example degree 2 polynomials defined by two points only:



(picture from Wikipedia)

– No, actually the scheme is secure against arbitrary adversaries, namely, it is information theoretically secure. As reasoned above, if the adversary learns only $t-1$ shares, it cannot find a_0, no matter how long the runtime, since there is no way for the adversary to check whether a guess for a_0 is correct (for any guess a_0, there is a polynomial that matches that point and the $t-1$ others).

– The Vandermonde matrix for the given shares is (recall that calculations are done modulo q):

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} = \begin{bmatrix} 1 & 403 & 450 \\ 1 & 967 & 82 \\ 1 & 1158 & 379 \end{bmatrix} \tag{1}$$

Let $a = (a_0, a_1, a_2)^T$ be the column vector of the polynomial's coefficients and $y = (y_0, y_1, y_2)^T$, then we need to solve $Xa = y \Leftrightarrow a = X^{-1}y$. The inverse of the matrix (computed using your favourite tool in GF(1361)) is

$$X^{-1} = \begin{bmatrix} 702 & 1136 & 995 \\ 1035 & 1196 & 491 \\ 118 & 239 & 1004 \end{bmatrix} \tag{2}$$

And hence $a = X^{-1}y = (1337, 861, 1029)^T$ So the secret is a_0 =1337, which is a way of writing LEET, a code that replaces English characters with numbers.

Exercise 4 (Simulation-Based Security of Symmetric Encryption)**.** We want to write IND-CPA security of a symmetric encryption scheme using the simulation-paradigm. Below, we give the outline of such a simulation-based definition. Decide which inputs the simulator gets and explain why. Explain why you think that the resulting notion is (or is not) equivalent to the standard notion of IND-CPA security. Optional: Sketch a reduction argument.

Remark: In the definition sim-IND-CPA security, the game maintains the state variable *st* to keep the state of the simulator.

Definition 4 (sim-IND-CPA). *A symmetric encryption scheme* $\texttt{se}$ *is sim-IND-CPA-secure if there exists a PPT simulator* $\mathcal{S}$ *such that the real game* $\texttt{Gsimind-cpa}^0$ *and the ideal game* $\texttt{Gsimind-cpa}^1$ *are computationally indistinguishable, that is, for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{A}}^{\texttt{Gsimind-cpa}^0,\texttt{Gsimind-cpa}^1}(\lambda)\\ &:= \left|\Pr\left[1 = \mathcal{A} \to \texttt{Gsimind-cpa}^0\right] - \Pr\left[1 = \mathcal{A} \to \texttt{Gsimind-cpa}^1\right]\right|\end{aligned}$$

is negligible in λ.

```
Gsimind-cpa^0_se                Gsimind-cpa^1_S

Parameters                      Parameters
λ:   sec. parameter             λ:   sec. parameter
se:  sym. enc. sch.             S:   simulator

Package State                   Package State
k:   key                        st:  simulator state

ENC(x)                          ENC(x)
if $k = \bot$ :                 —your code here—
    $k \leftarrow\$ \{0,1\}^\lambda$       $(c, st) \leftarrow\$ \mathcal{S}($—your input here$, st)$
                                return $c$
$c \leftarrow\$ \mathsf{se.enc}(k, x)$
return $c$
```

Solution 5. Here is the simulation based definition:

```
Gsimind-cpa^0_se                Gsimind-cpa^1_S

Parameters                      Parameters
λ:   sec. parameter             λ:   sec. parameter
se:  sym. enc. sch.             S:   simulator

Package State                   Package State
k:   key                        st:  simulator state

ENC(x)                          ENC(x)
if $k = \bot$ :                 $l \leftarrow |x|$
    $k \leftarrow\$ \{0,1\}^\lambda$       $(c, st) \leftarrow\$ \mathcal{S}(l, st)$
                                return $c$
$c \leftarrow\$ \mathsf{se.enc}(k, x)$
return $c$
```

Simulator only learns the length of the input x. Now, we could have the following simulator. The simulator stores the key in the state and always encrypts just the all zeroes string of length l using the key in the state. Now it is easy to see, that if we replace the simulator $\mathcal{S}$ by such functionality, then the resulting simulator based security definition is code equivalent to the IND-CPA security definition that we had in the first half of the course. So certainly, if enc is secure by IND-CPA, then it is also secure by the simulator based security definition (use as simulator the simulator described above).
Now we still need to argue why the other direction is true, i.e. prove the following claim:

Claim. If enc satisfies the simulator based security definition, then enc is IND-CPA secure.

This can be proved using reduction. Ask in zulip if you want to see the details.

Sender		**Receiver**
Input: $m_0, m_1 \in \{0,1\}^k$		Input: $b \in \{0,1\}$
$x \leftarrow\!\!\$\ \mathbb{Z}_q$		
$X \leftarrow g^x$		
	$\xrightarrow{\quad X \quad}$	
		$y \leftarrow\!\!\$\ \mathbb{Z}_q$
		$Y \leftarrow g^y$ if b=0,
		$Y \leftarrow Xg^y$ if b=1
	$\xleftarrow{\quad Y \quad}$	
$k_0 \leftarrow H\left(Y^x\right)$		$k_b \leftarrow H\left(X^y\right)$
$k_1 \leftarrow H\left(\left(\frac{Y}{X}\right)^x\right)$		
$c_0 \leftarrow\!\!\$\ enc(k_0, m_0)$		
$c_1 \leftarrow\!\!\$\ enc(k_1, m_1)$		
	$\xrightarrow{\quad c_0, c_1 \quad}$	
		$m_b \leftarrow dec(k_b, c_b)$

Fig. 1: Simple OT protocol. All arithmetic operations are modulo q.