CS 171: Problem Set 9

Due Date: April 18th, 2024 at 8:59pm via Gradescope

1 Bounded Collusion Identity-Based Encryption (10 Points)

In Discussion 10, we gave a candidate construction of IBE that is insecure if the attacker is allowed to make two queries to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.

Question: Prove that if DDH is hard for $\mathbb{G}$ and if the attacker is only allowed to make one query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$, then the attacker cannot break CPA security for this IBE scheme.

Note: You may assume that the adversary outputs the IDs used in its encryption and KeyGen queries at the start of the security game.[1]

Security Definition

Here is the definition of security that we will use in this problem.

Definition 1.1 (Weak CPA Security Game for Bounded Collusion IBE) *Let $n \in \mathbb{N}$ be the security parameter, and let $\mathcal{A}$ be the adversary.*

$\underline{\mathcal{G}(n, \mathcal{A})}$*:*

1. *The adversary outputs two different IDs $(\mathsf{ID}_E, \mathsf{ID}_K)$, which will be used for the encryption and KeyGen queries respectively.*[2] *Note that $\mathsf{ID}_E \neq \mathsf{ID}_K$.*
2. *The challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^n)$ and $b \leftarrow \{0, 1\}$. Then they send mpk to the adversary $\mathcal{A}$.*
3. *$\mathcal{A}$ can make at most 1 encryption query and 1 KeyGen query, which are defined below. The queries can be made in any order.*
   (a) ***Encryption Query:*** *$\mathcal{A}$ outputs ID_E along with two messages (m_0, m_1) of the same length. The challenger encrypts m_b as follows:*
   $$\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, \mathsf{ID}_E, m_b)$$
   The challenger returns ct to $\mathcal{A}$.

   (b) ***KeyGen Query:*** *$\mathcal{A}$ queries $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ on ID_K and receives $\mathsf{sk}_{\mathsf{ID}_K}$.*
4. *$\mathcal{A}$ outputs a bit b'. The output of $\mathcal{G}(n, \mathcal{A})$ is 1 if $b' = b$ and 0 otherwise.*

Definition 1.2 (Weak CPA Security for Bounded Collusion IBE) *We say that the IBE scheme is **weakly CPA-secure with collusion bound** 1 if for all PPT adversaries $\mathcal{A}$,*

$$\Pr[\mathcal{G}(n, \mathcal{A}) \to 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

[1] It is possible to prove security without this restriction, but that would require reprogramming the random oracle, which is an advanced technique that we won't cover in this class.

[2] In the regular CPA security game for IBE, the adversary can choose $\mathsf{ID}_E, \mathsf{ID}_K$ later on.

Solution

1. Key Idea: If the adversary is given just one $\mathsf{sk}_{\mathsf{ID}}$, then this is basically the El Gamal encryption scheme. Our security proof will resemble the proof of security for El Gamal encryption.

2. Assume toward contradiction that there's an adversary $\mathcal{A}_{CPA}$ that breaks weak CPA security for this encryption scheme. Then we can construct an adversary $\mathcal{A}_{DDH}$ to solve DDH with non-negligible advantage.

   $\mathcal{A}_{DDH}$:

   (a) The DDH challenger samples $\mathsf{pp} = (\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^n)$. They also sample $x, y, r \leftarrow \mathbb{Z}_q$ independently, and they sample $\beta \leftarrow \{0,1\}$. Then they send the following values to $\mathcal{A}_{DDH}$:
   $$(\mathsf{pp}, g^x, g^y, g^{xy+r\beta})$$

   (b) $\mathcal{A}_{DDH}$ will simulate the CPA security game.
      i. They run $\mathcal{A}_{CPA}$ until it outputs $(\mathsf{ID}_E, \mathsf{ID}_K)$.
      ii. They compute $r_E = H(\mathsf{ID}_E)$ and $r_K = H(\mathsf{ID}_K)$ and sample $s_K \leftarrow \mathbb{Z}_q$.
      iii. They compute:
      $$h_0 = (g^x \cdot g^{-s_K})^{(r_E - r_K)^{-1}}$$
      $$h_1 = g^{s_K} \cdot (h_0)^{-r_K}$$
      and send $\mathsf{mpk} = (\mathsf{pp}, h_0, h_1)$ to $\mathcal{A}_{CPA}$.
      iv. They also sample $b \leftarrow \{0,1\}$.

   (c) Queries:
      i. **Encryption Query:** When $\mathcal{A}_{CPA}$ outputs ID_E along with two messages (m_0, m_1), $\mathcal{A}_{DDH}$ responds with
      $$\mathsf{ct} = (g^y, g^{xy+r\beta} \cdot m_b)$$
      ii. **KeyGen Query:** When $\mathcal{A}_{CPA}$ outputs ID_K, $\mathcal{A}_{DDH}$ responds with
      $$\mathsf{sk}_{\mathsf{ID}_K} = (\mathsf{ID}_K, s_K)$$

   (d) When $\mathcal{A}_{CPA}$ outputs b', $\mathcal{A}_{DDH}$ checks whether $b' = b$. If so, $\mathcal{A}_{DDH}$ outputs $\beta' = 0$, and if not, $\mathcal{A}_{DDH}$ outputs $\beta' = 1$.

3. Let $s_E = x$. Then the distribution of $(h_0, h_1, r_E, r_K, s_E, s_K)$ are the same as they would be in the weak CPA security game.

   When $\mathcal{A}_{DDH}$ computes h_0 and h_1, they are implicitly fixing the values of a and b because $h_0 = g^a$ and $h_1 = g^b$. We will show that they choose the unique values of (a, b) such that:
   $$s_E = a \cdot r_E + b$$
   $$s_K = a \cdot r_K + b$$

Recall that the values of (r_E, r_K, s_E, s_K) are fixed before (h_0, h_1) are computed.

Let us calculate the values of (a, b):

$$\begin{aligned} h_0 &= (g^x \cdot g^{-s_K})^{(r_E - r_K)^{-1}} \\ g^a &= g^{(x - s_K)/(r_E - r_K)} \\ a &= \frac{x - s_K}{r_E - r_K} \end{aligned}$$

$$\begin{aligned} h_1 &= g^{s_K} \cdot (h_0)^{(-r_K)} \\ g^b &= g^{s_K - a \cdot r_K} \\ b &= s_K - a \cdot r_K \end{aligned}$$

Note that (a, b) are completely determined by (r_E, r_K, s_E, s_K), and these are the unique values of (a, b) that satisfy:

$$s_K = a \cdot r_K + b$$

and

$$\begin{aligned} s_E = x &= a \cdot r_E - a \cdot r_K + s_K \\ &= a \cdot r_E + b \end{aligned}$$

Finally, if we fix (r_E, r_K) such that $r_E \neq r_K$, then over the randomness of (s_E, s_K), the values of (a, b) are independent and uniformly random in $\mathbb{Z}_q$.

Therefore, the distribution of $(a, b, r_E, r_K, s_E, s_K)$ are the same as in the weak CPA security game.

4. If $\beta = 0$, then $\mathcal{A}_{DDH}$ correctly simulates the weak CPA security game with $s_E = x = a \cdot r_E + b$.

   This is because $\mathcal{A}_{DDH}$ simulates the encryption query correctly. Recall that when $\beta = 0$, $\mathcal{A}_{DDH}$ outputs the ciphertext:

$$\begin{aligned} \mathsf{ct} &= (g^y, g^{xy} \cdot m_b) \\ &= (g^y, (g^{ar_E + b})^y \cdot m_b) \\ &= (g^y, h_0^{y \cdot r_E} \cdot h_1^y \cdot m_b) \\ &= \mathsf{Enc}(\mathsf{mpk}, \mathsf{ID}_E, m_b) \end{aligned}$$

   In this case, $\mathcal{A}_{CPA}$ will guess $b' = b$ with non-negligible advantage:

$$\Pr[b' = b | \beta = 0] \geq \frac{1}{2} + \mathsf{non\text{-}negl}(n)$$

5. If $\beta = 1$, then ct gives $\mathcal{A}_{CPA}$ no information about b. In this case:

$$\mathsf{ct} = (g^y, g^{xy+r} \cdot m_b)$$

For any given (b, x, y), g^{xy+r} is a uniformly random group element due to the randomness of r. So $(g^{xy+r} \cdot m_b)$ is uniformly random and independent of (b, x, y). Therefore, $\mathcal{A}_{CPA}$ has no information about b. Then

$$\Pr[b' = b | \beta = 1] = \frac{1}{2}$$

6. Recall that $\beta' = 0$ if $b' = b$ and $\beta' = 1$ if $b' \neq b$. Then:

$$\Pr[\beta' = \beta | \beta = 0] = \Pr[b' = b | \beta = 0] \geq \frac{1}{2} + \mathsf{non\text{-}negl}(n)$$
$$\Pr[\beta' = \beta | \beta = 1] = \Pr[b' \neq b | \beta = 1] = \frac{1}{2}$$

$$\begin{aligned}\Pr[\beta' = \beta] &= \frac{1}{2} \cdot \Pr[\beta' = \beta | \beta = 0] + \frac{1}{2} \cdot \Pr[\beta' = \beta | \beta = 1] \\ &\geq \frac{1}{2} \cdot \left(\frac{1}{2} + \mathsf{non\text{-}negl}(n)\right) + \frac{1}{2} \cdot \frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{2} \cdot \mathsf{non\text{-}negl}(n)\end{aligned}$$

Note that $\frac{1}{2} \cdot \mathsf{non\text{-}negl}(n)$ is still non-negligible.

This means that $\mathcal{A}_{DDH}$ correctly guesses β with non-negligible advantage.

7. This is a contradiction because DDH is hard. Therefore, the original assumption was false, and in fact, there is no PPT adversary $\mathcal{A}_{CPA}$ that breaks weak CPA security for this encryption scheme.

■

2 Digital Signatures From Bilinear Maps (10 Points)

We will construct a digital signature scheme using a bilinear map and a random oracle.

Let $\mathcal{G}(1^n)$ generate the parameters of a bilinear map – $(\mathbb{G}, \mathbb{G}_T, q, g, e)$ – for which the *decisional bilinear Diffie-Hellman* problem (DBDH) is hard. Let $\mathbb{G}$ be the message space, and let $H : \mathbb{G} \to \mathbb{G}$ be a random oracle (a truly random function that all parties have access to).

Consider the following digital signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

1. $\mathsf{Gen}(1^n)$:
   (a) Generate the parameters of a bilinear map: $\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, q, g, e) \leftarrow \mathcal{G}(1^n)$.
   (b) Sample $x \leftarrow \mathbb{Z}_q$ independently, and compute $h = g^x$.
   (c) Output $\mathsf{pk} = (\mathsf{pp}, h)$ and $\mathsf{sk} = (\mathsf{pp}, x)$.
2. $\mathsf{Sign}(\mathsf{sk}, m)$: Let $m \in \mathbb{G}$. Then output
$$\sigma = H(m)^x$$
3. $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: TBD

Questions:

1. Fill in $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ so that the scheme is both correct and secure.
2. Prove that Π is correct, that any honestly generated signature will be accepted by $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$.
3. Let us modify the construction so that H is now just the identity function: $H(m) = m$ for all $m \in \mathbb{G}$. Prove that with this modification, the signature scheme is insecure.

Note: We won't have you prove the security of Π since the proof is a little more advanced than what we cover in this course.

Solution This is the BLS signature scheme. You can read more about the BLS scheme and find a sketch of the security proof in Boneh & Shoup, Section 15.5.

1. $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: Check whether
$$e(H(m), h) = e(g, \sigma)$$
If so, output 1 (accept). If not, output 0 (reject).
2. **Claim 2.1** *The signature scheme Π is correct.*

   Proof We can express $H(m)$ as $H(m) = g^y$ for some $y \in \mathbb{Z}_q$. Then
$$e(H(m), h) = e(g^y, g^x) = e(g, g)^{y \cdot x}$$

Furthermore, when σ is generated honestly,

$$\sigma = H(m)^x = (g^y)^x = g^{y \cdot x}$$

Then

$$\begin{aligned} e(g, \sigma) &= e(g, g^{y \cdot x}) = e(g, g)^{y \cdot x} \\ &= e(H(m), h) \end{aligned}$$

Therefore, σ will be accepted by $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ with certainty. ■

3. **Claim 2.2** *If we modify H so that H is the identity function, then the signature scheme is insecure.*

   Proof

   (a) The following adversary can forge a signature:

      i. The adversary chooses $m \in \mathbb{G} \backslash \{1\}$ and queries $\mathsf{Sign}(\mathsf{sk}, m)$ to obtain $\sigma = m^x$.

      ii. The adversary outputs $m^* = m^2$ and $\sigma^* = \sigma^2$ as its forgery.

   (b) Note that m^* was not previously queried to $\mathsf{Sign}(\mathsf{sk}, \cdot)$ because $m^* \neq m$.

   (c) Next, $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*)$ will accept. Let us express m as $m = g^y$ for some $y \in \mathbb{Z}_q \backslash \{0\}$. Then $m^* = g^{2y}$, and

   $$e(H(m^*), h) = e(m^*, h) = e(g^{2y}, g^x) = e(g, g)^{2y \cdot x}$$

   (d) Next

   $$\begin{aligned} e(g, \sigma^*) &= e(g, \sigma^2) = e(g, m^{2x}) = e(g, g^{2y \cdot x}) = e(g, g)^{2y \cdot x} \\ &= e(H(m^*), h) \end{aligned}$$

   Then $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*)$ will accept.

■

■

3 Merkle Proofs (10 Points)[3]

You will write a Python function to generate a Merkle proof.

You can learn more about Merkle proofs here, and you can download the starter code here. The starter code folder contains the following files:

- `prover.py`: This script generates the Merkle proof and writes it to a file for the verifier to read. Specifically it writes the leaf position, the leaf value, and the hashes used to prove the leaf's presence at the given position in the Merkle tree.

  Your job is to implement the function `gen_merkle_proof()`. The missing code can be implemented in less than ten lines of Python.

  Example of running `prover.py`: Run "`python3 prover.py 683`" from the command line. This script first calls the function `gen_leaves_for_merkle_tree()` to generate a thousand strings that will make up the leaves of a Merkle tree. Next it calls the method `gen_merkle_proof()` to generate the hashes for the Merkle proof for leaf number 683. Finally, it writes the Merkle proof to a text file `merkle_proof.txt`.

- `verifier.py`: The script reads in the Merkle proof generated by the prover and verifies that the leaf is at the stated position. Note that the value `ROOT` is hardcoded into this script. `ROOT` is the root for the Merkle tree whose leaves were generated by `gen_leaves_for_merkle_tree()`.

  Do not make any changes to this file.

- `merkle_utils.py`: This python script contains helpers used for generating and verifying the proof.

  Do not make any changes to this file.

- `proof-for-leaf-95.txt`: This is an example Merkle proof for leaf #95.

  Once you've finished editing `prover.py`, try generating the Merkle proof for leaf #95 with the command "`python3 prover.py 95`", and then compare the result to the expected output provided in `proof-for-leaf-95.txt`.

Another Test: After you implement the function `gen_merkle_proof()` in `prover.py`, run the following two scripts and check that your output matches the output below:

```
$ python3 prover.py 683
   I generated 1000 leaves for a Merkle tree of height 10.
   I generated a Merkle proof for leaf #683 in file merkle_proof.txt

$ python3 verifier.py 683
   I verified the Merkle proof:  leaf #683 in the committed tree is "data item
683".
```

[3] This problem is adapted from this project.

Try changing one character in `merkle_proof.txt` and check that the verifier now rejects the proof.

Deliverables: Please submit your file `prover.py` on Gradescope. The autograder will test your prover on random leaves.

Tips: To help you understand the starter code, try to answer the following questions for yourself. You do not need to submit your answers to these questions:

- What does the verifier expect you to include in the proof?
- How is height defined?
- What is the purpose of the padding in `gen_merkle_proof()`?

Solution

```
def gen_merkle_proof(leaves, pos):
    """Takes as input a list of leaves and a leaf position.
    Returns the list of hashes that prove the leaf is in
    the tree at position pos."""


    height = math.ceil(math.log(len(leaves),2))
    assert height < MAXHEIGHT, "Too many leaves."

    # hash all the leaves
    state = list(map(hash_leaf, leaves))

    # Pad the list of hashed leaves to a power of two
    padlen = (2**height)-len(leaves)
    state += [b"\x00"] * padlen

    # initialize a list that will contain the hashes in the proof
    hashes = []

    level_pos = pos    # local copy of pos

    for level in range(height):
        is_right_node = level_pos % 2
        sibling_pos = (level_pos - 1) if is_right_node else (level_pos + 1)

        if sibling_pos < len(state):
            sibling_hash = state[sibling_pos]
            hashes.append(sibling_hash)

        new_state = []
        for i in range(0, len(state), 2):
            left = state[i]
            right = state[i + 1] if (i + 1) < len(state) else left
# Handle the case where the number of nodes at this level is odd
            new_state.append(hash_internal_node(left, right))

        state = new_state
        level_pos //= 2

    return hashes
```

■