CS 601.442/642 – Modern Cryptography Instructors: Aditya Hegde and Harry Eldridge

Homework 8 Solutions

Deadline: April 16; 2026, 11:59pm ET

Instructions

- The solutions must be submitted via Canvas.
- You must typeset your solutions. We suggest using LaTeX or Typst.

Problems

1. (30 points) In class, we saw an interactive zero-knowledge proof for graph isomorphism: given a pair of graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$, the prover convinces the verifier that there exists a permutation $\phi : V_0 \to V_1$ such that $G_1 = \phi(G_0)$ without revealing the permutation ϕ. Let $G_0 \simeq G_1$ be a shorthand to denote that the two graphs are isomorphic.

   In this problem, we consider an extension of the graph-isomorphism language. The statement consists of *two* pairs of graphs, and the claim is that **at least one** of the two pairs is isomorphic, i.e., the language L is defined as

   $$L = \left\{ \Big((G_0, G_1), (H_0, H_1) \Big) \ : \ G_0 \simeq G_1 \ \vee \ H_0 \simeq H_1 \right\}.$$

   A *zero-knowledge* proof system for this language must prevent the verifier from learning anything beyond the fact that at least one pair is isomorphic. In particular, it must not learn *which* pair is isomorphic (or if only one or both pairs are isomorphic) nor the permutation between the isomorphic pair(s).

   Figure 1 describes a protocol for this language. Answer the following questions.

   - (5 points) Prove that the protocol satisfies **completeness**.
   - (10 points) Prove that the protocol satisfies **soundness**.
   - (15 points) Prove that the protocol satisfies **zero-knowledge**.

2. (30 points) Suppose a client stores n files $m_1, \ldots, m_n$ on a cloud server. Later, the client downloads a file m_i and wants to verify that the server returned the correct file. One approach is for the client to keep a local copy of all files and compare, but this defeats the purpose of cloud storage.

   A more economical approach is to store a short *hash* $h_i = H(m_i)$ for each file. When the client downloads m_i, it checks that $H(m_i) = h_i$. If H is collision-resistant, a computationally-bounded cheating server cannot produce a different file $m_i' \neq m_i$ with the same hash. This is better since the client stores n short hashes instead of n large files; but the storage still grows linearly with n.

   At the other extreme, the client could hash *everything* into a single digest $h = H(m_1 \| \cdots \| m_n)$. Now the client stores only one hash, but verifying any single file requires downloading *all* n files to recompute h.

   Merkle trees provide a trade-off between these extremes. Figure 2 describes the Merkle tree construction and Figure 3 provides an example for $n = 8$ messages.

   Answer the following questions.

- (10 points) Analyze the runtime of $\mathsf{MT.H}$, $\mathsf{MT.Open}$, and $\mathsf{MT.Verify}$. If the client uses a Merkle tree to verify downloaded files, how much must the client store locally? How many values must the server send for the client to verify a single file m_i? Compare both quantities to the two extremes described above.
- (20 points) Prove that the Merkle tree ensures binding at each position. Specifically, prove that if CRH is a collision resistant hash function then for any non-uniform PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{c} \mathsf{MT.Verify}(s, \mathsf{h_{rt}}, i, m, \pi) = 1 \\ \wedge\, \mathsf{MT.Verify}(s, \mathsf{h_{rt}}, i, m', \pi') = 1 \\ \wedge\, m \neq m' \end{array} : \begin{array}{r} s \leftarrow \mathsf{MT.Gen}(1^\lambda) \\ (\mathsf{h_{rt}}, i, m, \pi, m', \pi') \leftarrow \mathcal{A}(1^\lambda, s) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

3. (20 points) In this problem we will construct a symmetric encryption scheme that is IND-CPA secure, but *not* IND-CCA-1 secure.

   (a) (5 points) Let $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA secure encryption scheme. Construct an encryption scheme $(\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$. Briefly justify in words why your scheme is *correct*. You do not need to write a full proof.

   (b) (5 points) Prove via reduction that your scheme satisfies IND-CPA security.

   (c) (10 points) Prove that your scheme does *not* satisfy IND-CCA-1 security.

4. (20 points) Let $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme, and $(\mathsf{MACKeyGen}, \mathsf{Tag}, \mathsf{Ver})$ be a strong UF-CMA secure MAC scheme. Consider the "Encrypt *and* MAC" method of encryption below.

$\mathsf{KGen}'(1^\lambda)$	$\mathsf{Enc}'((k_{\mathsf{mac}}, k_{\mathsf{enc}}), m)$	$\mathsf{Dec}'((k_{\mathsf{mac}}, k_{\mathsf{enc}}), (\mathsf{ct}, \sigma))$
$k_{\mathsf{mac}} \leftarrow \mathsf{MACKeyGen}(1^\lambda)$	$\mathsf{ct} \leftarrow \mathsf{Enc}(k_{\mathsf{enc}}, m)$	$m \leftarrow \mathsf{Dec}(k_{\mathsf{enc}}, \mathsf{ct})$
$k_{\mathsf{enc}} \leftarrow \mathsf{KGen}(1^\lambda)$	$\sigma \leftarrow \mathsf{Tag}(k_{\mathsf{mac}}, m)$	if $\mathsf{Ver}(k_{\mathsf{mac}}, m, \sigma) \neq 1$ then return $\perp$
return $(k_{\mathsf{mac}}, k_{\mathsf{enc}})$	return (ct, σ)	else return m

   In this question, we will construct an IND-CCA-1 secure encryption scheme such that when it is used as the underlying encryption scheme by the above construction, the result is not IND-CCA-2 secure.

   (a) (5 points) Let $(\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ be an IND-CCA-1 secure encryption scheme. Construct an IND-CCA-1 secure encryption scheme $(\mathsf{KGen}'', \mathsf{Enc}'', \mathsf{Dec}'')$. Briefly justify why your scheme is correct and satisfies IND-CCA-1 security. You do not need to do a formal proof.

   (b) (15 points) Prove that when the "Encrypt *and* MAC" scheme above is instantiated with your scheme, the resulting scheme is *not* IND-CCA-2 secure.

Zero-Knowledge Proof for L

Statement:. $(G_0, G_1), (H_0, H_1)$, where all four graphs are on $n := \mathsf{poly}(\lambda)$ vertices.

Protocol: The prover first computes the permutation for the isomorphic pair, i.e., if $G_0 \simeq G_1$, it computes a permutation ϕ_G such that $G_1 = \phi_G(G_0)$. Else, if $H_0 \simeq H_1$, it computes a permutation ϕ_H such that $H_1 = \phi_H(H_0)$.

Repeat the following procedure λ times using fresh randomness.

- **Prover $\to$ Verifier:**
  1: Sample $b_1 \leftarrow \{0,1\}$ and $b_2 \leftarrow \{0,1\}$.
  2: Sample $\pi_1 \leftarrow \mathsf{Perm}_n$ and $\pi_2 \leftarrow \mathsf{Perm}_n$.
  3: Compute $G := \pi_1(G_{b_1})$ and $H := \pi_2(H_{b_2})$.
  4: Send (G, H).

- **Verifier $\to$ Prover:**
  1: Sample $b \leftarrow \{0,1\}$ and send b.

- **Prover $\to$ Verifier:**
  1: Choose b'_1 and b'_2 with $b'_1 \oplus b'_2 = b$ as follows:
  - If $G_0 \not\simeq G_1$: set $b'_1 := b_1$ and $b'_2 := b \oplus b_1$.
  - If $H_0 \not\simeq H_1$: set $b'_2 := b_2$ and $b'_1 := b \oplus b_2$.
  - If both $G_0 \simeq G_1$ and $H_0 \simeq H_1$: choose any b'_1, b'_2 with $b'_1 \oplus b'_2 = b$.
  2: Compute π'_1 and π'_2 such that $G = \pi'_1(G_{b'_1})$ and $H = \pi'_2(H_{b'_2})$.
  3: Send $(b'_1, b'_2, \pi'_1, \pi'_2)$.

- **Verifier:**
  1: Check that $b'_1 \oplus b'_2 = b$.
  2: Check that $G = \pi'_1(G_{b'_1})$ and $H = \pi'_2(H_{b'_2})$.
  3: Accept if all checks pass.

Figure 1: Interactive zero-knowledge proof for the disjunction of two graph isomorphism instances.

Merkle Tree Hash

Notation. Let $\mathsf{CRH} = (\mathsf{Gen}, \mathsf{H})$ be a collision resistant hash function[a].

- $\mathsf{MT.Gen}(1^\lambda)$:
  1: Output $s \leftarrow \mathsf{CRH.Gen}(1^\lambda)$.

- $\mathsf{MT.H}(s, m_1, \ldots, m_n)$:
  Let n be a power of 2. Compute the following.
  1: For each $i \in \{1, \ldots, n\}$, compute $\mathsf{h}_i^{(0)} \coloneqq \mathsf{CRH.H}_s(m_i)$.
  2: For each $j \in \{1, \ldots, \log n\}$ and each $i \in \{1, \ldots, n \cdot 2^{-j}\}$ compute

$$\mathsf{h}_i^{(j)} \coloneqq \mathsf{CRH.H}_s\left(\mathsf{h}_{2i-1}^{(j-1)} \| \mathsf{h}_{2i}^{(j-1)}\right).$$

  3: Output $\mathsf{h}_{\mathsf{rt}} \coloneqq \mathsf{h}_1^{(\log n)}$.

- $\mathsf{MT.Open}(s, i, m_1, \ldots, m_n)$:
  1: Let $i_0 \coloneqq i$. For each $j \in \{0, \ldots, \log n - 1\}$ compute

$$\sigma_j \coloneqq \begin{cases} \mathsf{h}_{i_j - 1}^{(j)} & \text{if } i_j \text{ is even,} \\ \mathsf{h}_{i_j + 1}^{(j)} & \text{otherwise} \end{cases}$$

  and set $i_{j+1} \coloneqq \lceil i_j / 2 \rceil$, where each $\mathsf{h}_i^{(j)}$ is computed as in $\mathsf{MT.H}$.
  2: Output $\pi \coloneqq (\sigma_0, \sigma_1, \ldots, \sigma_{\log n - 1})$.

- $\mathsf{MT.Verify}(s, \mathsf{h}_{\mathsf{rt}}, i, m_i, \pi)$:
  1: Parse $\pi = (\sigma_0, \sigma_1, \ldots, \sigma_{\log n - 1})$.
  2: Set $d_0 \coloneqq \mathsf{CRH.H}_s(m_i)$ and $i_0 \coloneqq i$. For each $j \in \{1, \ldots, 0\} \log n - 1$, compute

$$d_{j+1} \coloneqq \begin{cases} \mathsf{CRH.H}_s(\sigma_j \| d_j) & \text{if } i_j \text{ is even,} \\ \mathsf{CRH.H}_s(d_j \| \sigma_j) & \text{otherwise} \end{cases}$$

  and set $i_{j+1} \coloneqq \lceil i_j / 2 \rceil$.
  3: Output 1 if $d_{\log n} = \mathsf{h}_{\mathsf{rt}}$, and 0 otherwise.

[a] The domain of CRH is any arbitrary (polynomial length) bitstring and its range is $\{0,1\}^\lambda$

Figure 2: Merkle tree construction from collision-resistant hash function.

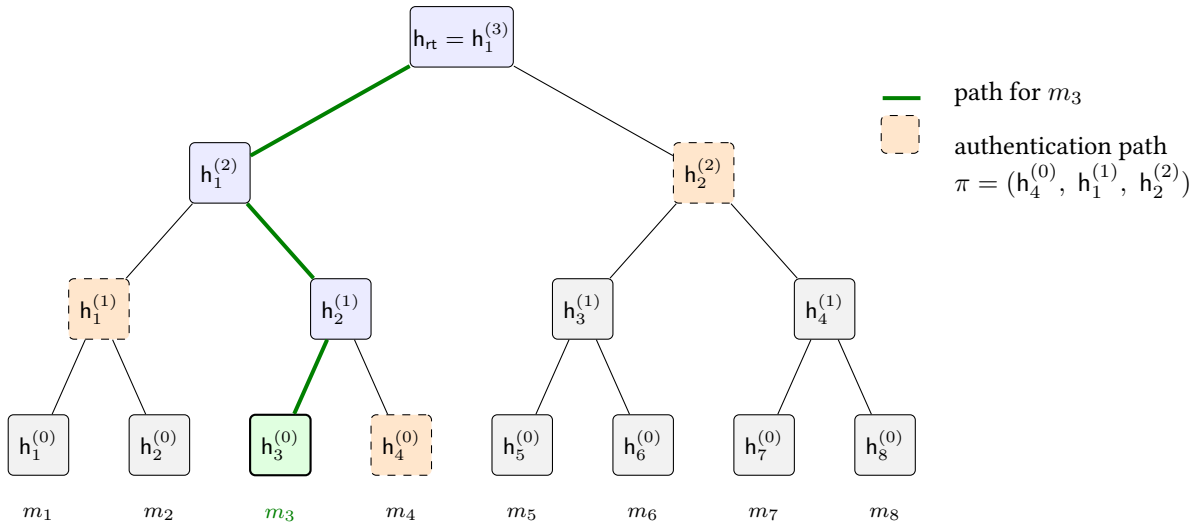

Figure 3: Merkle tree for $n = 8$ messages. To verify m_3, the server provides m_3 and the authentication path $\pi = (\mathsf{h}_4^{(0)}, \mathsf{h}_1^{(1)}, \mathsf{h}_2^{(2)})$ (dashed nodes). The client recomputes $\mathsf{h}_3^{(0)} = \mathsf{CRH.H}_s(m_3)$, then $\mathsf{h}_2^{(1)} = \mathsf{CRH.H}_s(\mathsf{h}_3^{(0)} \| \mathsf{h}_4^{(0)})$, then $\mathsf{h}_1^{(2)} = \mathsf{CRH.H}_s(\mathsf{h}_1^{(1)} \| \mathsf{h}_2^{(1)})$, and checks $\mathsf{CRH.H}_s(\mathsf{h}_1^{(2)} \| \mathsf{h}_2^{(2)}) = \mathsf{h}_{\mathsf{rt}}$.