1.

sha3-256.py > ...

```python
import hashlib
s = b"abcdefgh"                    # bytes literal, exact bytes
h = hashlib.new("sha3_256", s).hexdigest()
print(h)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell

```
PS C:\Users\takoo\CS415\Assignment3> python sha3-256.py
3e2020725a38a48eb3bbf75767f03a22c6b3f41f459c831309b06433ec649779
PS C:\Users\takoo\CS415\Assignment3>
```

2. $G_1(x) = \underbrace{G(x)}_{256 \text{ bits}} \,||\, \underbrace{x}_{128 \text{ bits}}$

Split in u = first 256 bits, v = last 128

so $z = u || v$

For every probablistic polynomial time distinguisher D,

$|\Pr_{x \leftarrow U_{128}}[D(G(x)) = 1] - \Pr_{r \leftarrow U_{256}}[D(r) = 1]|$

construct $G_1 : \{0,1\}^{128} \to \{0,1\}^{384}$, $G_1(x) = G(x) || x$

$|\Pr[D'(G_1(U_{128})) = 1] - \Pr[D'(U_{384}) = 1]|$

$D'(z) = \begin{cases} 1 \text{ if } G(v) = u \\ 0 \text{ otherwise} \end{cases}$

$Adv_{G_1}^{PRG}(D') = |\Pr_x[D'(G_1(x)) = 1] - \Pr_z[D'(z) = 1]|$

case 1: $z = G_1(x) = G(x) || x$

$u = G(x)$ $v = x$

$G(v) = G(x)$, so $G(v) = u$

$\Pr_x[D'(G_1(x)) = 1] = 1$

case 2: $z = u || v$ where z is uniform in $\{0,1\}^{384}$

distinguisher outputs 1 iff $G(v) = u$

$\Pr_z[D'(z) = 1] = 2^{-256}$

$Adv_{G_1}^{PRG}(D') = |1 - 2^{-256}| \approx 1$ ∴ generator G_1 cannot be pseudorandom

∴ G_1 is not a secure PRG

2.

$G_2 : \{0,1\}^{128} \to \{0,1\}^{255}$

There exists a PPT distinguisher D' and a non-negligible $\epsilon(\cdot)$ such that

$$|\Pr_x[D'(G_2(x))=1] - \Pr_{u \leftarrow U_{255}}[D'(u)=1]| \ge \epsilon(n)$$

if $w = G(x)$, then $w_{[1..255]} = Trunc(G(x))$ so D accepts with probability $\Pr_x[D'(G_2(x))=1]$

if w is uniform U_{256}, then $w_{[1..255]}$ is uniform U_{255} so D accepts with probability $\Pr_{u \leftarrow U_{255}}[D'(u)=1]$

$\therefore Adv_G(D) = Adv_{G_2}(D') \ge \epsilon(n)$, contradicting security of G, hence no D' exists; G_2 is a secure PRG

$G_3 : \{0,1\}^{256} \to \{0,1\}^{512}$

1. For every efficient distinguisher D',

$$|\Pr[D'(G_3(U_{128}, U_{128}))=1] - \Pr[D'(U_{512})=1]|$$

2. Hybrid => $H_0 = G(x) \| G(y)$ $H_1 = U_{256} \| G(y)$ $H_2 = U_{256} \| U_{256} = U_{512}$

3. prove

$$|\Pr[D'(H_0)=1] - \Pr[D'(H_2)=1]| \le |\Pr[D'(H_0)=1 - \Pr[D'(H_1)=1]| + |\Pr[D'(H_1)=1] - \Pr[D'$$

4. construct D_1 for G

D_1 receives 256-bit string z

D_1 generates random seed $y \leftarrow U_{128}$

D_1 computes $G(y)$ and forms 512-bit string and feeds to D'

D_1 outputs whatever D' outputs

if $z = G(x)$, D_1 input to D' is $G(x) \| G(y) = H_0$

if z is uniform, D_1 input to D' is $U_{256} \| G(y) = H_1$

$\therefore T_1 \le negl(n)$

5. Construct D_2

D_2 receives 256-bit challenge string z

Random 256-bit string $u \leftarrow U_{256}$

input $= u \| z$ for D'

D_2 outputs whatever D' outputs

if $z = G(y) = H_1$

if $z = U_{256}$ $= U_{256} \| U_{256} = H_2$

$\therefore T_2 \le negl(n)$

6. Conclude, $|Pr[D'(H_0)=1] - Pr[D'(H_2)=1]| \le T_1 + T_2 \le negl(n) + negl(n)$

$\therefore G_3$ is a secure PRG

3. Assume adversary has encryption oracle $Enc(SK, \cdot)$

1. Before challenge, query encryption oracle on all zero messages, receive $c0 = Enc(sk, 0^{256})$
$= 0^{256} \oplus r^* = r^*$

2. Submit any two distinct messages $m\text{-}0, m\text{-}1 \in \{0,1\}^{256}$, pick $b \in \{0,1\}$ return
$c^* = Enc(sk, m\text{-}b) = m\text{-}b \oplus r^*$

3. Compute $c^* \oplus c^0 = (m\text{-}b \oplus r^*) \oplus r^* = m\text{-}b$, output recovered $m\text{-}b$ and guess $\hat{b}$ accordingly

4. r^* cancels, returns probability 1 $\therefore$ construction is insecure

3.

4. 1. Configuring Challenger Services

The challenger selects a secret bit b ∈ {0,1} at random for every experiment. It uses the pseudorandom function Fλ(k,·) to answer queries and chooses a secret key k if b = 1. It reacts with a consistent, genuinely random function if b = 0. For every experiment, the challenger securely and independently keeps this state.

2. Interface for Attacker Interaction

An outside attacker has the ability to (a) launch a fresh experiment, (b) send in several queries, and (c) guess the answer to b. AWS API Gateway, which routes requests to backend functions while enforcing authentication and rate limits, exposes these actions via HTTPS endpoints.

3. Design of the AWS Framework

Services:

All computation (creating experiments, processing queries, and verifying results) is handled by AWS Lambda.

Per-experiment state, including encrypted secrets, counters, and status, is stored in DynamoDB.

Bit values and secret key encryption are managed by AWS KMS.

S3 and CloudWatch track performance and record outcomes.

State, Randomness, and Security: Using a CSPRNG, Lambda generates secrets and randomness, which are then encrypted and stored using KMS.

Cost control and query limits are enforced by API Gateway usage plans and DynamoDB counters, which also guard against overuse.

Scalability and Fairness: While usage quotas and throttling guarantee equitable resource sharing, serverless lambda scaling accommodates numerous concurrent attackers.