Code-Snippet

```
# EchoPulse Hash Function & Input Encoding Specification (Document A4)
**Version:** 1.0 **Date:** May 11, 2025 **Author:** EchoPulse
Initiative This document specifies the exact encoding, input structure,
and usage of the SHA3-256 cryptographic hash function within the
EchoPulse Key Encapsulation Mechanism (KEM) protocol. Consistent
application of this specification is crucial for ensuring
interoperability and security across different EchoPulse
implementations. ## 1. Hash Function Selection The EchoPulse KEM
employs the SHA3-256 cryptographic hash function, as defined in FIPS
202. * **Algorithm:** SHA3-256 * **Output Length:** 32 bytes (256 bits)
* **Security Properties:** SHA3-256 is a collision-resistant hash
function and is considered to offer strong security properties against
known quantum algorithms. * **Implementation Standard:** All EchoPulse
implementations must adhere to the NIST reference specification for
SHA3-256. ## 2. Input Encoding Structure The input to the SHA3-256 hash
function within the EchoPulse protocol is constructed by the
concatenation of two byte sequences: the encoded final state vertex and
the random symbol sequence.
```

Hash_Input = v_enc_bytes || r_bytes

```
Where: * **`v_enc_bytes`:** Represents the 2-byte little-endian
encoding of the final state vertex ($v$) reached during the
encapsulation process. The vertex $v$ is an element of the state set
$V$ of the graph $G(V, E)$. * **`r_bytes`:** Represents the 28-byte raw
symbol sequence ($r$) that serves as the public payload transmitted
during key exchange. The total length of the `Hash_Input` byte string
is fixed at 30 bytes (2 bytes for $v_{enc}$ and 28 bytes for $r$). ##
3. Encoding Details The following specific encoding rules must be
followed when constructing the `Hash_Input`: * **Encoding of v_enc:**
The final state vertex $v_{enc}$ (a 2-byte unsigned integer) is encoded
in little-endian format. This means the least significant byte of $v$
is placed at the lower memory address (first byte), and the most
significant byte is placed at the higher memory address (second byte).
For a vertex $v$, the encoding is: * Byte 1: `v & 0xFF` (Least
significant byte) * Byte 2: `v >> 8` (Most significant byte) *
**Encoding of r:** The random symbol sequence $r$ is treated as a
sequence of 28 raw bytes. Each byte in $r$ represents a symbol and
falls within the range `0x00` to `0xFF`. These bytes are directly
appended to the `v_enc_bytes` without any modification. * **No Padding
or Compression:** The `Hash_Input` must be formed by the direct
concatenation of the encoded $v_{enc}$ and the raw $r$ without any
additional padding, compression, or other alterations. * **Constant-
Time Hashing:** The SHA3-256 hashing operation must be implemented
using constant-time techniques to mitigate potential timing-based side-
channel attacks. ## 4. Hashing Implementation Notes Implementers should
adhere to the following guidelines when implementing the SHA3-256
hashing: * **Cryptographic Libraries:** Utilize well-vetted and
established cryptographic libraries that provide constant-time
implementations of the SHA3 family of hash functions. * **No
Intermediate Inspection:** The individual bytes of `v_enc_bytes` should
not be accessible or observable in a data-dependent manner before being
processed by the hash function. * **Input Validation:** Implementations
should reject any input to the SHA3-256 function that does not conform
to the specified length of 30 bytes. Handling of malformed inputs
should be done securely and without revealing information about the
expected input structure. ## 5. Worked Example Consider the following
example values: * Encapsulation final state vertex: $v_{enc} = 0x03FA$
(decimal 1018) * Random symbol sequence $r$: `BC 22 91 00 99 F0 A1 B2
C3 D4 E5 F6 07 18 29 3A 4B 5C 6D 7E 8F 90 A1 B2 C3` (28 bytes in
hexadecimal representation) The encoding process would be: * **Encoding
of v_enc:** * Byte 1 (`0x03FA & 0xFF`): `FA` * Byte 2 (`0x03FA >> 8`):
`03` * `v_enc_bytes = FA 03` * **Encoding of r:** `r_bytes = BC 22 91
00 99 F0 A1 B2 C3 D4 E5 F6 07 18 29 3A 4B 5C 6D 7E 8F 90 A1 B2 C3` *
**Construction of Hash_Input:** * `Hash_Input = FA 03 BC 22 91 00 99 F0
A1 B2 C3 D4 E5 F6 07 18 29 3A 4B 5C 6D 7E 8F 90 A1 B2 C3` (30 bytes) *
**SHA3-256 Output:** * `$K = SHA3\_256(FA 03 BC 22 91 00 99 F0 A1 B2 C3
D4 E5 F6 07 18 29 3A 4B 5C 6D 7E 8F 90 A1 B2 C3)$` * Example result
```