Algorithmic Specifications
Dynamic Multi-Primitive Cryptographic Hopping Protocol (DMP-CHP)

January 26, 2026

1 Core Algorithms

1.1 Stateless Schedule Derivation (Resilient Hopping)

To handle packet loss and out-of-order delivery (e.g., in UDP/QUIC streams), the hopping state depends on a monotonic sequence number rather than strictly the previous internal state.

Algorithm 1 Get_Hop_Parameters

Require: Master secret K_{session}, packet sequence number SeqID
Require: Library size N_{algo}, mode Mode
Ensure: Algorithm Index idx, Packet Key k_{pkt}

1: $\mathsf{Salt} \leftarrow \mathsf{Mode}.\text{salt}$
2: $\mathsf{Seed} \leftarrow \text{HMAC}(K_{\text{session}}, \text{"}\mathsf{HOP}\text{"} \parallel \mathsf{SeqID} \parallel \mathsf{Salt})$ ▷ Use the first 32 bits of Seed in network byte order for index derivation
3: $idx \leftarrow \text{Trunc32}(\mathsf{Seed}[0{:}4] \text{ (network byte order)}) \bmod N_{\text{algo}}$
4: $k_{pkt} \leftarrow \text{HKDF}(\mathsf{Seed}, \mathsf{info} = \text{"}\mathsf{KEYGEN}\text{"})$
return (idx, k_{pkt})

1.2 Holographic Entropy Dispersion (Encryption)

1.3 Active Decoy Injection

1.4 Neuro-Cognitive Adaptation

Algorithm 2 Holographic_Encrypt

Require: Session secret K_{session}
Require: Plaintext payload P, threshold k, total shares n, base sequence $\mathsf{BaseSeqID}$
Require: Mode Mode, max orthogonality retries $R_{\max}$
Require: Orthogonality library Λ (array of algorithms), library size N_{algo}
Ensure: Composite Ciphertext C_{bundle}

$Shares \leftarrow \text{Shamir\_Split}(P, k, n)$ ▷ Generate n shares
$C_{bundle} \leftarrow []$
$\mathsf{prev\_class} \leftarrow \perp$ ▷ No previous hard-problem class for first share
for $i \leftarrow 1$ **to** n **do**
    $s_i \leftarrow Shares[i]$
    $\mathsf{SeqID} \leftarrow \mathsf{BaseSeqID} + i$ ▷ Transmitted / authenticated monotonic sequence identifier
    $\mathsf{SelCtr} \leftarrow \mathsf{SeqID}$ ▷ Selection counter used for enforcing orthogonality
    $(idx, key) \leftarrow \text{Get\_Hop\_Parameters}(K_{\text{session}}, \mathsf{SelCtr}, N_{\text{algo}}, \mathsf{Mode})$
    $Algo \leftarrow \Lambda[idx]$ ▷ Select algorithm for this share
    ▷ Ensure Orthogonality: retry until hard-problem class differs from previous
    $\mathsf{retry} \leftarrow 0$
    while $Algo.\text{hard\_problem\_class} = \mathsf{prev\_class}$ **do**
        $\mathsf{SelCtr} \leftarrow \mathsf{SelCtr} + 1$ ▷ Deterministic bump to escape same-class selection
        $(idx, key) \leftarrow \text{Get\_Hop\_Parameters}(K_{\text{session}}, \mathsf{SelCtr}, N_{\text{algo}}, \mathsf{Mode})$
        $Algo \leftarrow \Lambda[idx]$
        $\mathsf{retry} \leftarrow \mathsf{retry} + 1$
        if $\mathsf{retry} > R_{\max}$ **then**
            break ▷ Fallback after $R_{\max}$ attempts; orthogonality may be relaxed
        end if
    end while
    $c_i \leftarrow Algo.\text{Encrypt}(key, s_i, \text{AD} = \text{Header}(\mathsf{SeqID}))$ ▷ Use AEAD-style Encrypt(key, plaintext, AD) to bind SeqID
    $C_{bundle}.\text{append}(\{c_i, \text{Header}(\mathsf{SeqID})\})$ ▷ Header contains authenticated SeqID bound as AD
    $\mathsf{prev\_class} \leftarrow Algo.\text{hard\_problem\_class}$
end for
return C_{bundle}

Algorithm 3 Generate_Decoy

Require: Legitimate traffic distribution $\mathcal{D}_{\text{traffic}}$, current schedule
Require: Session secret K_{session}
Require: Administrative/control key K_{admin} (or a control key derivable from K_{session})
Require: Orthogonality library Λ (array of algorithms), library size N_{algo}, mode Mode
Require: Ghost sequence counter $\mathsf{GhostSeqID}$ (monotonic counter stored in state; initialize to 0 if undefined)
Ensure: Decoy Packet P_{decoy}

$\mathsf{TargetLen} \leftarrow \text{Sample}(\mathcal{D}_{\text{traffic}}.\text{length})$
$\mathsf{TargetInterarrival} \leftarrow \text{Sample}(\mathcal{D}_{\text{traffic}}.\text{timing})$
$\mathsf{Noise} \leftarrow \text{TRNG}(\mathsf{TargetLen})$
$\mathsf{GhostSeqID} \leftarrow \mathsf{GhostSeqID} + 1$ ▷ Monotonic counter reserved for decoys (stateful)
$(idx, key) \leftarrow \text{Get\_Hop\_Parameters}(K_{\text{session}}, \mathsf{GhostSeqID}, N_{\text{algo}}, \mathsf{Mode})$
$Algo \leftarrow \Lambda[idx]$
$P_{decoy} \leftarrow Algo.\text{Encrypt}(key, \mathsf{Noise})$
$P_{decoy}.\text{Header.Flag} \leftarrow \text{Encrypted}(\text{"}\mathsf{DECOY}\text{"}, K_{\text{admin}})$
return P_{decoy}

Algorithm 4 Update_Threat_State

Require: Current State $State_t$, Network Metrics M_t, Global Model θ_{global}

Ensure: New Mode $Mode_{t+1}$

```
Features ← Extract_Features(M_t)
ThreatScore ← NeuralNet_{θ_global}(Features)
LocalEntropy ← Measure_Jitter()
if ThreatScore > T_paranoid then
    Mode_{t+1} ← "NANO_HOPPING"
    Inject_Chaff(Rate = HIGH)
else if ThreatScore > T_alert then
    Mode_{t+1} ← "MICRO_HOPPING"
else
    Mode_{t+1} ← "MACRO_HOPPING"
end if
Update_Locally_Learned_Weights()          ▷ Federated Learning Step return Mode_{t+1}
```