HYPHA

Hypha Secure Remote Compute Fabric

A Cybersecurity-Oriented Architecture for Distributed Compute Across Trusted and Untrusted Networks

Audience	Cybersecurity, infrastructure, platform engineering, and technical due diligence teams
Focus	Remote transport, identity, trust, authorization, execution boundaries, and resilience
Disclosure posture	Security architecture and validation evidence without exposing proprietary implementation details

Executive premise

Hypha treats network reachability, cryptographic identity, authorization, execution capability, and control-plane role as distinct security concerns. This separation allows geographically distributed and heterogeneous systems to participate in one governed compute fabric without equating network presence with trust.

This paper intentionally omits private key material, exact signing procedures, scheduler internals, election logic, recovery internals, and proprietary implementation details.

Executive Summary

Hypha is a distributed compute fabric designed to coordinate heterogeneous compute resources across local, remote, and intermittently connected environments while preserving explicit trust boundaries between transport, identity, authorization, execution, and control-plane state.

The architecture does not require every system to share the same physical LAN. Remote systems can participate through authenticated encrypted transport while retaining cryptographic node identity and policy enforcement at the Hypha layer.

The design separates four concerns that are often collapsed into a single remote-access mechanism: network transport establishes encrypted reachability; cryptographic identity establishes which machine or operator is participating; authorization determines what that identity may do; and execution control determines where workloads may run and how results are accepted.

> *Reachability is necessary, but not sufficient, for participation.*

1. Security Objective

The goal is not merely to make remote computers reachable. The goal is to let distributed systems participate in one execution fabric while minimizing implicit trust.

• Remote connectivity does not automatically imply execution authority.

• Possession of an IP address does not establish node identity.

• Nodes do not require unrestricted inbound exposure.

• Execution capabilities can be constrained independently of network access.

• Trust can be changed or revoked without redesigning the network.

• Administrative access and workload execution remain separate concerns.

• Control-plane availability can extend across physical locations.

2. Layered Trust Model

Hypha uses multiple independent trust layers. No single layer is expected to solve transport, identity, authorization, and execution control at once.

Layer	Primary purpose	Security implication
VPN transport	Encrypted routed reachability	Remote systems reach protected services without public exposure
TLS / certificates	Service identity and channel protection	Clients can distinguish legitimate internal services from impostors
Public/private keys	Verifiable cryptographic identity	Verification does not require distribution of the signing secret
Signed authorization	Bound permission for sensitive actions	Network access alone is not sufficient authority
Capability-aware execution	Constrain work to supported nodes	Heterogeneous systems need not be treated as interchangeable

3. Transport Trust: VPN as a Connectivity Layer

Remote systems connect through an encrypted VPN overlay. In the validated environment, UniFi Teleport provided the practical remote-access transport when direct inbound WireGuard was constrained by upstream NAT that was not under local administrative control.

The VPN establishes encrypted transit, remote network reachability, isolation from direct Internet exposure, and authenticated participation. It does not, by itself, grant Hypha execution authority.

Remote system	Encrypted tunnel	Managed gateway	Private office network	Hypha services
ACE / AX16Pro	Teleport / VPN	UniFi gateway	10.11.x.x routed space	Registry + fleet

4. Cryptographic Identity: Public and Private Keys

Public/private key cryptography is a foundational trust primitive. Private keys remain under the control of the owning system or authority; corresponding public keys can be distributed to components that need to verify signatures or identities.

This permits verification without distributing the secret used to create the signature. That is materially safer than architectures in which broad trust depends on a shared administrative secret.

5. Certificates and Trust Roots

Certificates and internal trust roots establish authenticated service identity in addition to encrypted transport. This protects against endpoint impersonation, unauthorized service substitution, accidental trust of the wrong registry, and interception when certificate verification is enforced.

> *Encryption without identity verification is not equivalent to encryption with authenticated endpoint identity.*

Operationally, trusted CA material should be installed explicitly on clients and nodes. Temporary insecure TLS verification may be useful during troubleshooting, but it should not be treated as the production trust posture.

6. Signed Authorization

Hypha separates identity from permission. A known node or operator need not be authorized for every operation. Sensitive actions can be protected with signed authorization objects that bind intent to properties such as target, operation, expiry, nonce, authorization domain, and signer.

The governing principle is that a request should be trusted because it is cryptographically authorized for the requested action, not merely because it originated from an internal IP address.

7. Separation of Transport, Control Plane, and Execution Roles

One of the strongest architectural properties is the deliberate separation between network connectivity, registry/controller participation, and compute-node execution.

Role	What it means	What it does not imply
VPN participant	Can reach protected routed services	Not automatically trusted to execute workloads
Registry/controller	Participates in control-plane coordination	Not automatically a Fleet worker
Fleet node	Advertises capabilities and executes authorized jobs	Not automatically a registry/controller

8. Remote Registry Validation

ACE was used as a remote registry host located away from the primary office environment. The registry was configured to use the existing protected backend path, then the host was physically relocated and remote transport was tested from home.

The key validation was not simply that ACE had Internet connectivity. The test demonstrated that a geographically remote registry host could regain routed access to protected office services without directly publishing Hypha internals to the Internet.

9. NAT and Exposure Boundaries

Testing exposed a realistic constraint: direct inbound WireGuard could not complete because the upstream carrier-controlled gateway performed NAT and did not expose the required UDP forwarding. The design did not respond by weakening the perimeter. Instead, the deployment used a managed remote-access path that did not require opening public service ports.

This preserved the intended exposure boundary: remote access enters through an authenticated encrypted transport, while Hypha services remain private.

10. Remote Node Behavior

A remote Hypha node does not need unrestricted inbound administrative connectivity to participate. Nodes can establish outbound connectivity to the registry, advertise capabilities, request or receive work, and return results through the fabric. This supports systems behind NAT, residential routers, enterprise firewalls, remote offices, and roaming networks.

11. Heterogeneous Fleet Validation

A three-round fleet benchmark was executed across 13 healthy nodes. The benchmark measured end-to-end Hypha job timing rather than raw compute throughput. Most nodes used a harmless noop operation; specialized nodes without that operation exposed different capability contracts.

Node	Median end-to-end time	Result
ubuntu-usb-tpu	2.574 s	3/3 succeeded
ubuntu-a6000	2.656 s	3/3 succeeded
ubuntu	3.075 s	3/3 succeeded
ubuntu-m6000	9.127 s	3/3 succeeded
AX16Pro	12.008 s	3/3 succeeded
hp840	12.820 s	3/3 succeeded
EVO-T1	14.151 s	3/3 succeeded
hypha-r740-01	15.006 s	3/3 succeeded
miniamd16c	18.699 s	3/3 succeeded
hp840-rtx3060	19.003 s	3/3 succeeded
EVO-T1-NPU	-	0/3 on tested contract
NUC	-	0/3 on tested contract
NUC-GTX970	-	0/3 on tested contract

These figures include registry submission, scheduling, lease acquisition, node polling behavior, execution, and result propagation. They are not CPU or accelerator microbenchmarks.

12. The Older Ubuntu System

One result is particularly noteworthy. The older eight-core Ubuntu system completed the benchmark with a median end-to-end time of approximately 3.075 seconds, behind only the TPU-backed node and the A6000-backed node in this specific fabric test.

This does not mean the older system is computationally faster than modern accelerators. It demonstrates a more useful distributed-systems point: hardware age alone is a poor predictor of end-to-end usefulness inside a fabric. Scheduling overhead, polling state, workload fit, software path, and resource availability materially affect observed service time.

A modest system with favorable execution characteristics can remain highly valuable when the fabric treats compute as a heterogeneous resource pool rather than a hierarchy based only on hardware generation.

13. Capability-Aware Execution

Not every node must support every operation. That is intentional. Accelerator-specific nodes may expose different operation sets than general-purpose CPU nodes. This reinforces a capability-aware scheduling model rather than an assumption of uniform worker behavior.

14. Failure as a Security and Reliability Signal

Three nodes did not successfully complete the tested contract. The significance is not that failures occurred; it is that the fabric remained healthy and failures were attributable to specific node or operation paths instead of becoming systemic control-plane failure.

> *Component failure should remain local and observable rather than becoming control-plane failure.*

15. Security Boundary Summary

Boundary	What it controls
Internet / carrier edge	Untrusted external network and upstream NAT
Encrypted VPN transport	Remote reachability into protected network space
TLS + certificates	Authenticated service endpoint and encrypted channel
Cryptographic node identity	Who or what is participating
Signed authorization	What that identity is permitted to do
Capability-aware scheduler	Where work may execute
Controlled execution	How work is performed and results are returned

16. Relevance to Cybersecurity Teams

Many distributed systems implicitly trust one or more of the following: LAN membership, VPN membership, possession of a shared token, administrative SSH access, hostnames, or static IP addresses. Hypha moves toward a more explicit trust model in which network access is only one factor.

• Authenticate explicitly.

• Authorize explicitly.

• Minimize implicit trust.

• Separate identities from network locations.

• Constrain privileges.

• Preserve auditability.

• Treat network location alone as insufficient evidence of trust.

The architecture aligns with zero-trust principles, but this paper does not claim formal zero-trust certification or complete formal verification.

17. What Is Intentionally Not Disclosed

• Private key material or certificate private keys.

• Signing keys or trust-root generation procedures.

• Exact authorization serialization or nonce-generation mechanisms.

• Internal scheduler decision logic.

• Failover and registry election implementation details.

• Recovery internals and proprietary workload-distribution mechanisms.

These details are not required to validate the security architecture and would either weaken operational security or disclose implementation-specific intellectual property.

18. Security Position

Hypha demonstrates a layered distributed-compute architecture in which encrypted remote transport, TLS service identity, cryptographic node identity, signed authorization, capability-aware execution, and control-plane roles are treated as separate security boundaries.

This is a defensible architectural claim. It does not assert invulnerability, formal verification, or security through obscurity.

Conclusion

Hypha's remote-compute architecture is not simply a VPN-connected cluster. The VPN provides transport. Certificates establish service identity. Public/private key cryptography establishes verifiable trust relationships. Signed authorization constrains privileged actions. Persistent node identity distinguishes machines from addresses. Capability discovery allows heterogeneous hardware to participate safely. The registry coordinates the fabric without requiring every participant to be publicly exposed.

The practical result is a distributed execution environment that can span physical sites while preserving explicit security boundaries and accommodating a heterogeneous mix of CPU systems, accelerators, Windows and Linux hosts, remote machines, and older hardware that remains operationally valuable.

HYPHA | Patent Pending

© 2026 TKT, Inc. All Rights Reserved.

HYPHA technology is the subject of pending patent applications. The contents of this white paper, including its text, illustrations, diagrams, and technical descriptions, are protected by applicable copyright laws. No portion may be reproduced, distributed, or transmitted without prior written permission from TKT, Inc.