THE PAST IS PROLOGUE:
WHAT WORKLOAD IDENTITY'S HISTORY TELLS US ABOUT THE FUTURE

Sal Kimmich

Core Concepts

- **Sovereignty**: Not just state control, but individual digital autonomy.
- **Zero-Knowledge Proofs (ZKPs)**: Prove something is true without revealing data.
- **Verifiable Trust**: Trust must replaced by cryptographic mathematical proof.
- Open Source recursion over the secure kernel is **unique, important, and worthy of your understanding**

What We'll Cover

- Understand what workload identity is and why it's essential.
- Trace how historical authentication protocols shaped today's systems.
- Explore zero trust principles in action.
- See how SPIFFE/SPIRE, cloud-native tools, and confidential computing are applied today.
- Discover how AI and federation solve modern identity challenges.
- Connect with today's talks on securing agents, multi-cloud workloads, and identity migration.

Defining Workload Identity

A workload identity is a cryptographic identity assigned to a software workload, used to authenticate securely without relying on static secrets.

Examples:

- SPIFFE ID: spiffe://org/service-name
- Azure Managed Identity
- AWS IAM Role or certificate (via Roles Anywhere)

Anatomy of Workload Identity

- **Issuance** - SPIRE issues identity based on workload attestation.
- **Authentication** - Proves identity using X.509 or JWT.
- **Authorization** - Enforced via RBAC/OPA/ABAC.
- **Rotation** - Automated lifecycle with expiration (e.g., every 1h).
- **Revocation** - Immediate invalidation of misused credentials.

Historical Foundations to Now

- 1978 – Needham-Schroeder
- 1983 – Kerberos
- 1988 – X.509 Certificates
- 2010 – OAuth2 / OIDC
- 2018 – SPIFFE
- 2022+ – Confidential Computing / Federated Identity

Lessons from the Past

- Needham-Schroeder: basis of mutual authentication → used in SPIFFE's mTLS handshake.
- Kerberos: short-lived, ticket-based credentials → mirrored in SPIRE's SVID lifecycle.
- X.509: foundation of PKI → used by cert-manager, AWS Private CA, SPIRE.

Every "modern" solution inherits principles from 30+ years of security evolution.

Effect of Distributed Systems

- VMs → Containers → Serverless
  - Challenge: Secrets in code/config break under fast autoscaling, ephemeral pods.
  - Opportunity: Move to dynamic, verifiable, short-lived identity models.

- Cloud-native Identity Requires:
  - Auto-issuance
  - Policy enforcement
  - Ephemeral trust binding

Federation and SPIFFE

1. Key Concept: Federation
    a. Trust across domains (SAML, OIDC).
    b. Federated tokens = single sign-on for workloads.

1. Key Technology: SPIFFE
    a. Platform-agnostic identity (spiffe://).
    b. SPIRE enables attestation-based issuances

SPIFFE is to workloads what SSO is to humans-but stronger and ephemeral.

Spire Identity Flow

1. Step-by-step SPIRE Identity Flow
2. Workload starts
3. SPIRE Agent attests platform/workload
4. SPIRE Server issues SVID
5. Workload uses SVID to connect via mTLS
6. Expired every N minutes; rotated automatically

Verifiable Trust

- Replacing institutional trust (banks, governments, corporations) with mathematically provable systems.
  - Instead of trusting a bank to approve a transaction, we use cryptographic proofs that verify it mathematically.

Verifiable Compute

- Verifiable compute provides **mathematically-guaranteed integrity** by proving that some end state is the result of a set of inputs into some program

Verifiability is NOT Trust

- Trust-based systems are fragile and often abused.
- What if we didn't need to trust but could verify everything?
- **Sovereignty, cryptography, and open-source governance** enable that shift.

Zero Knowledge Proofs

- Zero-Knowledge Proofs (ZKPs) provide cryptographic method that allows one party to prove something is true without revealing the actual data.
- Example: Proving you are over 18 without revealing your birthdate.

Secure Isolation

- Secure Isolation is the ability to keep data and workloads separate from untrusted entitites during computation
  - A securely isolated system ensures
    - Confidentiality - prevents unauthorized access
    - Integrity - proteects data from unauthoized modification
    - Availability - ensures isolated workloads run without disruption

Secure Isolation

- Types of Secure Isolation
  - **Process Isolation**: keeping applicatins and user processes separate
  - **Memory Isolation**: Preventing unabuthorized access to active data
  - **Workload Isolation**: Ensuring different workloads to not interfere

For a long time, I've been figuring out how we do security for systems with a human in the loop. Now I am interested in how we do security in a system to prevent **a human, or machine, or workload identity** in the loop

Understanding Trust Boundaries

- A **trust boundary** is a line where the level of trust changes between components, users, or systems
  - It defines which entities can be trusted to handle sensitive data or executive privileged operations

- **Where do trust boundaries exist?**
  - Between a user and a system (e.g. authentication mechanisms)
  - Between a client and a server (e.g. HTTPS encrypting traffic)
  - Between microservices in a distributed system (e.g. API gateways)
  - Between a cloud providers and its cusomters (e.g. multi-tenant data separation)

Zero Trust Principles in Practice

- Zero Trust: Every request is evaluated based on context and real-time attestation.

- Real-World Example Today: Uber’s adoption of SPIFFE to secure service-to-service communications across thousands of services.

- Tool Integration:
  - SPIRE + Open Policy Agent (OPA) → dynamic policy evaluation.

Zero Trust for Agentic Platforms

- Agentic Workflow: autonomous execution by AI or software agent.
  - Autonom ut cryptographic identit
  - Agentic + fine-grained policie

CHECK THIS SLIDE

Google Cloud

🟦 AWS Roles Anywhere – X.509 certs for IAM roles

🟦 Microsoft Entra – App and pod identity with RBAC

Highlight: All 3 major clouds now offer workload identity systems–with short-lived, verifiable credentials.

Workload Identity in Kubernetes

- Kubernetes-Specific Challenges
  - Dynamic pods
  - No long-lived secrets
  - Trust per namespace
- Tools:
  - SPIRE Workload Attestor
  - Azure Workload Identity for K8s
  - cert-manager + cert rotation

Scaling Workload Identity

- Key Challenges:
  - Rotation: Auto-expiring credentials = safer, but error-prone at scale.
  - Latency: Trust token exchange across clouds adds delay.
  - Revocation: Federated identity revocation isn't instantaneous.
- Solutions:
  - SPIFFE Federation
  - JWKS caching
  - Cross-cloud trust stores
  -

Credential Lifecycle

- Creation
- Validation
- Expiration
- Rotation
- Revocation

MAKE DIAGRAM

Scaling Workload Identity

- Key Concept: Confidential Computing
  - Hardware-backed execution isolation for code + data.

- Attestation Use Case:
  - SPIRE integrates with SGX/SEV
  - Only verified enclaves receive SVIDs

Trusted Execution Environemnt → Attestation Service → SPIRE Workload identity

Scaling Workload Identity

- Examples:
  - Fintech: encrypted queries across shared compute
  - Healthcare: secure analysis on private medical data
  - Multi-party ML: Encrypted model training across orgs

Confidential computing makes Zero Trust tangible—even when workloads share untrusted infrastructure.

AI Powered Threat Detection

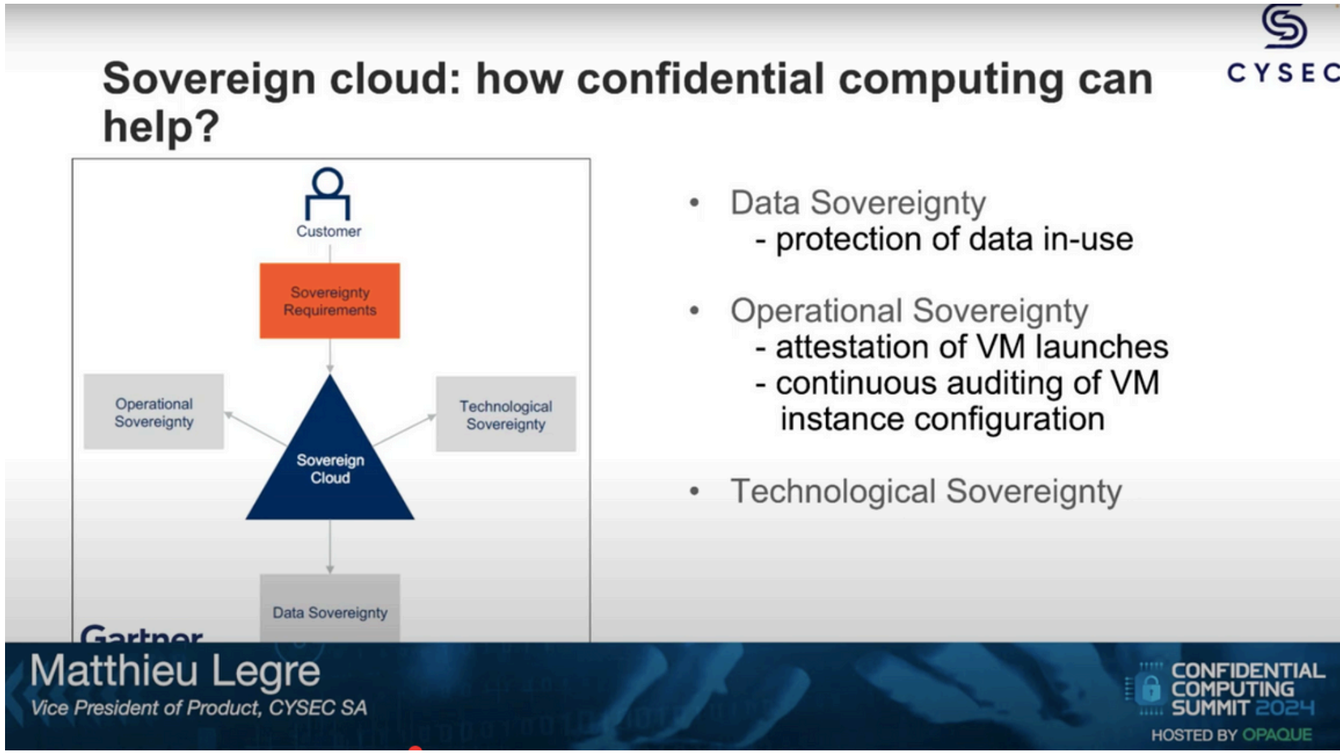
- Key Concept: Anomaly Detection for Workload Behavior
  - AI learns workload baselines
  - Flags credential misuse, lateral movement, or suspicious privilege escalation

- Challenges:
  - False positives
  - Need for robust audit trails + model retraining

- **Scenario: Workload accesses resource it never has before**
  - AI flags as anomaly
  - System revokes SPIFFE ID, logs incident, isolates pod
  - Outcome: Automated risk containment within seconds

Sovereign Cloud is Now

- Sovereign Cloud between Italy, Switzerland and France
- **Data Sovereignty** - protection of data in use
- **Operational Sovereignty** - user that wants transparency to host operations
- **Attestation** of VM launches
- **Continuous auditing** of VM instance configuration
- **Extend Supervision** on their VM to some operations of the host related to isolation and agreed to between two parties



Understanding the Secure Kernel

1961 - The Secure Kernel Concept emerges with Ferranti Atlas and IBM's Privelaged Mode, introducing the idea of executing trusted operations sepa[illegible]ly

1971 - the first Virtual Memory Implementations prevent one [illegible]from accessing another's memory, laying the foundation for process isolati[illegible]

1974 - MITRE's Security Kernel Prototype formally defi[illegible]Computing Base (TCB, establishing the idea that some parts of a [illegible]d while others must not.

1983 - The Bell-LaPadula Model is introudced, pi[illegible]ol (MAC) to enforce strict data confidentiality in [illegible]the first time we see layers of trust.

1985 - the Orange Book (TCEC) sets the US governme[illegible]ardware security standards, leading to the introduction of A1-Cla[illegible]s (highly trusted, formally verified security kernels).

1990s - Virtualizaton technology gains momentum, enablin[illegible]oad isolation through Virtual Machines (VMS), a precurser to modern cloud secu[illegible]y

2000s - The rise of Cloud and Multi-Tenant Computing shifts security research towards isolating workloads in shared infrastructure, addressing the challenges of cloud security

Cryptography before Linux

- **1970s: Development of the Data Encryption Standard (DES)**
  - The U.S. government adopted DES, marking the first widespread use of cryptographic security in computing.
  - DES would later be integrated into early Unix cryptography libraries.
- **1980s: Public-Key Cryptography and PGP**
  - RSA encryption was developed in 1977 and gained popularity in the 1980s.
  - Pretty Good Privacy (PGP), developed by Phil Zimmermann in 1991, introduced a widely accessible form of encryption for securing emails and files.
  - These advancements set the stage for later cryptographic applications in Linux.

Cryptography after Linux

- **1991: Linux Kernel 0.01 Released:** Linus Torvalds released the first version of the Linux kernel, which did not yet include built-in encryption features.
- **1994: Introduction of DES in Unix and Early Linux Systems:** DES encryption was incorporated into Unix-based systems for password hashing. Early Linux distributions also included support for DES-based encryption.
- **1997: Weaknesses in DES Lead to the Development of AES:** The Electronic Frontier Foundation cracked DES using brute force, demonstrating its insecurity. This led to a global push for a stronger encryption standard, resulting in the later adoption of AES (Advanced Encryption Standard).

Rise of Strong Encryption

- **2001: Linux Kernel 2.4 Introduces /dev/random and /dev/urandom:** providing a reliable entropy source for cryptographic functions.This significantly improved the security of key generation in Linux systems.
- **2003: Cryptoloop Introduced for Disk Encryption (Linux Kernel 2.6):** provided the first in-kernel implementation for encrypting disk partitions. It suffered from design flaws and was later deprecated.
- **2004: LUKS (Linux Unified Key Setup)** Introduced a standard for disk encryption, improving cross-distribution compatibility. This allowed users to securely encrypt entire disks with strong cryptographic algorithms.

Encryption Expands

- **2005: Introduction of dm-crypt in Linux Kernel 2.6.4**
  - **dm-crypt:** a superior alternative to Cryptoloop, became the standard for Linux disk encryption, allwoing seamless encryption of block devices with the device mapper framework.
- **2006: eCryptfs (Encrypted Filesystem) Integrated into the Linux Kernel (v2.6.19):**
  - eCryptfs enabled per-file encryption, making it ideal for protecting home directories. This provided a more flexible alternative to full-disk encryption.
- **2008: Full Disk Encryption Becomes a Default in Ubuntu (Home Directory Encryption):**
  - ne of the first major distributions to provide encrypt home directories by default, **making encryption more accessible to end users.**

Strengthening Kernel Security

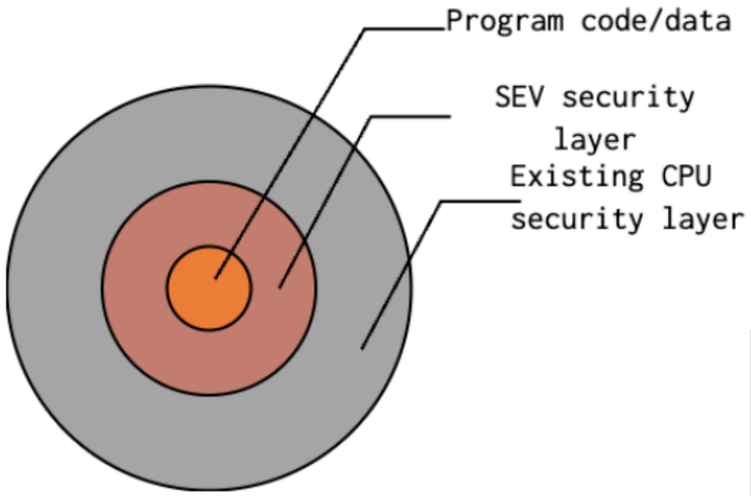
- **2014: Introduction of Kernel Lockdown and Secure Boot Enhancements:** Linux community pushed for stronger protections against unauthorized kernel modifications.
- Kernel lockdown mode prevented users from tampering with sensitive system components.
- **2017: Inclusion of the Adiantum Algorithm for Lightweight Encryption:** Google introduced Adiantum, an encryption method optimized for low-powered devices (e.g., Android phones and IoT devices), integrated into Linux to support lightweight encryption.
- **2018: NSA's SPECK Cipher Controversy:** The Linux kernel briefly included the NSA-developed SPECK cipher, optimized for performance on constrained hardware. Due to security concerns, community backlash led to its removal in 2019.

Modern Linux Encryption

- **2021: Linux Kernel 5.10 Enhances Secure Computation:** The kernel introduced Secure Encrypted Virtualization (SEV), enabling encrypted memory for virtual machines.
- **2022: Remote Attestation Integrated into Linux for Confidential Computing:** Support for remote attestation was introduced, allowing users to verify the integrity of their computing environment in cloud-native workloads.
- **2023: Introduction of Rust in the Linux Kernel for Safer Cryptographic Implementations:** The Linux kernel began supporting Rust, a memory-safe language, to improve cryptographic security implementations.

Modern Linux Encryption

- **2024: AI and Confidential Computing Converge**
  - Confidential computing solutions (e.g., Intel SGX, AMD SEV, and ARM CCA) integrate deeper with AI and cloud-native workloads.
  - Zero-Knowledge Proofs and Multi-Party Computation are now used alongside trusted execution environments.



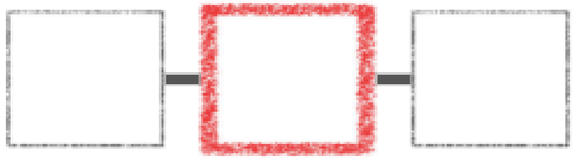
SEV security layers

Roberto Castellotti (rcastellotti.dev)

Performance Evaluation of AMD SEV

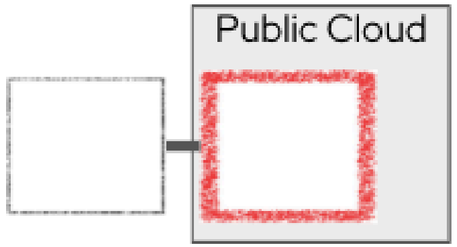
rcastellotti.dev

Partner Interaction



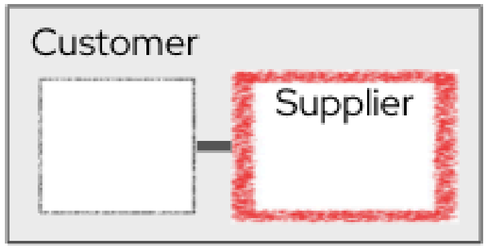
2 protected datasets interacting in confidential container

Secure Cloudburst



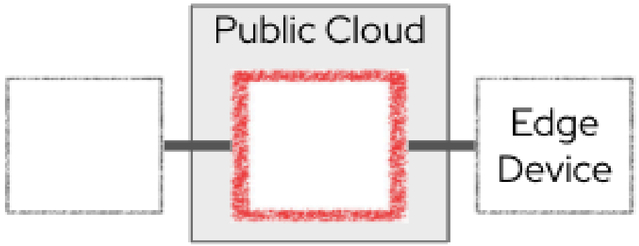
Using the public cloud to for peak workload or shared resources

IP Protection/Integrity



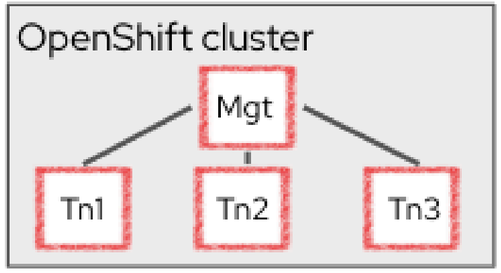
Protection of supplier data and business logic in customer environments

Edge use case



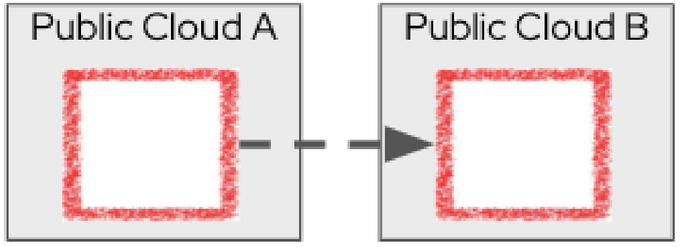
Protecting Edge device data in the public cloud for aggregation

Total Tenant Isolation



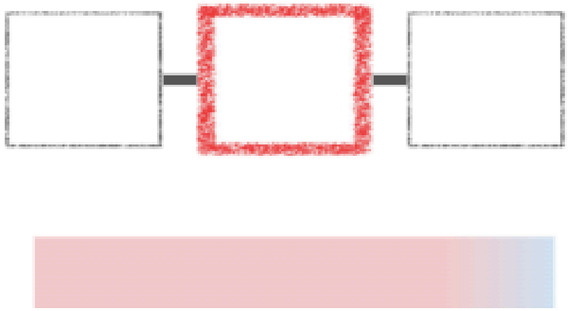
Isolating OpenShift Tenants

Digital Sovereignty

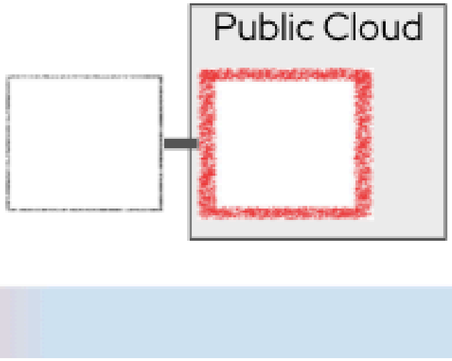


Encapsulating and moving workload from one provider to the next.

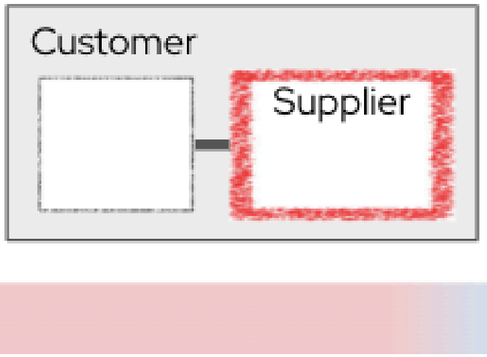
Partner Interaction



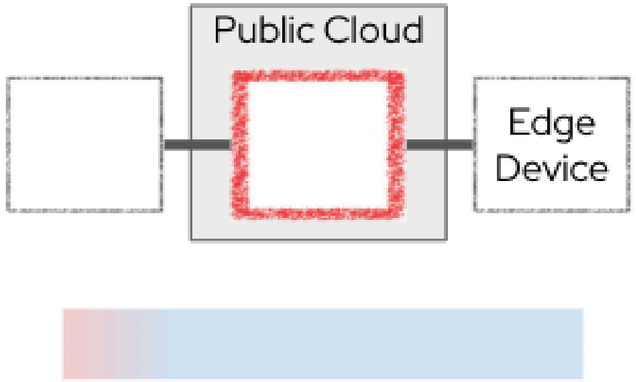
Secure Cloudburst



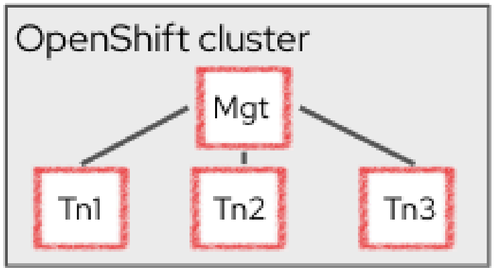
IP Protection/Integrity



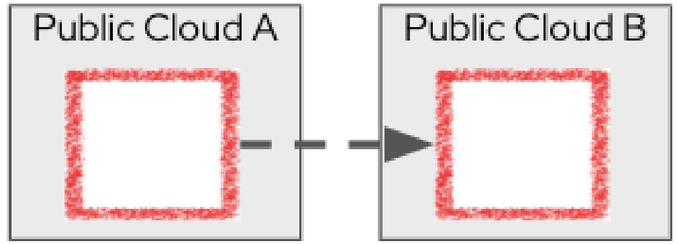
Edge use case



Total Tenant Isolation



Digital Sovereignty



Bare Metal

Public Cloud

Axel Saab, RedHat

GEOPOLITICAL CODE: HOW DIGITAL SOVEREIGNTY SHAPES NATIONS

SEEKING EXPERT PERSPECTIVES ON THE TOPIC OF HOW DIGITAL SOVEREIGNTY SHAPES NATIONS, DIGITAL POLICIES AND TECHNOLOGICAL POWER AND AUTONOMY.