Network Working Group
Internet-Draft
Intended status: Informational
Expires: 28 June 2026

J. S. Wang
Independent Researcher

25 December 2025

# ACE-GF: A Generative Framework for Atomic Cryptographic Entities
## draft-wang-acegf-protocol-00

Abstract

This document specifies the Atomic Cryptographic Entity Generative Framework (ACE-GF), a cryptographic construction for deriving and reconstructing stable cryptographic identities from user-held credentials without requiring persistent storage of a master secret.

ACE-GF addresses a structural limitation of existing deterministic key-derivation and identity systems, which rely on long-lived root secrets and rigid derivation hierarchies. By separating identity reconstruction from long-term secret storage, ACE-GF enables stateless identity recovery, credential rekeying, and context-isolated derivation across multiple cryptographic algorithms.

The framework is designed to be application-agnostic and may be applied to diverse environments such as authentication systems, distributed identities, secure key management, and cryptographic wallets. This document defines the core construction, security properties, and interoperability considerations of ACE-GF, while application-specific profiles are defined separately.

Table of Contents

# 1. Introduction

## 1.1. Background and Motivation

Deterministic key management schemes, such as BIP-32 *[BIP32]* and BIP-39 *[BIP39]*, have simplified key recovery by deriving keys from a single master seed. However, these schemes rely on the long-term persistent storage of that seed, which constitutes a single point of failure (SPOF). If the seed is compromised, all derived keys are irreversibly exposed.

This storage-centric design is ill-suited for Autonomous Digital Entities (ADEs), such as AI agents and IoT deployments, which require identity continuity without centralized trust anchors *[DID-Core]*. ACE-GF addresses these challenges by decoupling deterministic identity from long-term secret storage, ensuring that the identity root exists only ephemerally during the reconstruction process.

### 1.2. Core Contributions

ACE-GF introduces a generative framework for Atomic Cryptographic Entities (ACE) with the following properties:

* *Seed-Storage-Free*: The identity root (REV) exists only ephemerally in memory during active operations.

* *Deterministic Reconstruction*: The REV is reconstructed from a sealed artifact and authorization credentials.

* *Context Isolation*: Cryptographic keys are isolated across algorithms (e.g., Ed25519, ML-DSA) using explicit context encoding *[RFC5869]*.

## 2. Conventions and Terminology

### 2.1. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 *[RFC2119]* *[RFC8174]* when, and only when, they appear in all capitals, as shown here.

### 2.2. Terminology

The following terms are used throughout this document:

*ACE-GF* Atomic Cryptographic Entity Generative Framework. A cryptographic framework that enables deterministic reconstruction of digital identities from user-held credentials without requiring persistent storage of a master secret.

*Credential* A user-provided secret input to ACE-GF, such as a passphrase, mnemonic, or other high-entropy material. Credentials are supplied at the time of identity reconstruction and are not persistently stored by the framework.

*Identity* A stable cryptographic identity derived via ACE-GF. An identity may correspond to one or more public/private key pairs, addresses, or identifiers, depending on the application context in which ACE-GF is applied. An Identity in ACE-GF is defined by the ability to reconstruct the underlying Root Entropy Value (REV); loss of this ability renders the identity cryptographically unreachable.

*Context* An explicit domain-separation parameter, as defined in the

context of Key Derivation Functions *[NIST-SP800-108]*, used by ACE-GF to ensure that derived material for different purposes, algorithms, or applications remains cryptographically isolated.

*Rekeying* The process of updating the authorization metadata (Sealed Artifact) to associate new credentials with an existing identity while preserving the underlying identity semantics. Unlike traditional key rotation, ACE-GF rekeying does not modify the underlying Root Entropy Value (REV).

*Profile* A specification that defines how ACE-GF is applied within a specific application domain. Profiles may impose additional constraints, parameters, or output formats, but do not modify the core ACE-GF construction defined in this document.

*Wallet Application Profile* An application profile that applies ACE-GF to cryptocurrency wallet systems, including multi-algorithm key derivation and migration from legacy wallet formats. This profile is illustrative and does not constrain other applications of ACE-GF.

## 3. Protocol Overview

This section provides a high-level, non-normative overview of the ACE-GF framework. It establishes the mental model for the decoupled architecture before the detailed protocol specification in Section 4.

### 3.1. The Decoupled Identity Model

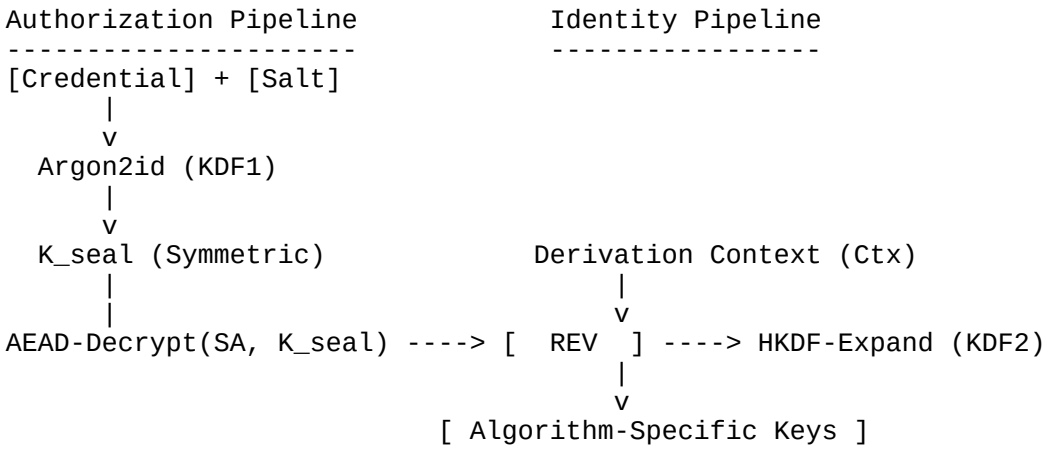
ACE-GF operates on the principle of *Functional Decoupling*. Unlike traditional deterministic systems that require the persistent storage of a master seed, ACE-GF separates the identity into two logically independent pipelines:

* *Authorization Pipeline (The "Lock")*: Manages the transformation between a user’s volatile *Credential* and a persistent *Sealed Artifact (SA)*. It controls _access_ to the identity.

* *Identity Pipeline (The "Key")*: Performs deterministic derivation from the reconstructed *Root Entropy Value (REV)* to various cryptographic algorithms. It defines the _substance_ of the identity.

### 3.2. Data Flow and Transformation

The following diagram illustrates how entropy flows from a raw state, through the authorization lock, and into algorithm-specific keys:




### 3.3. Atomic Entity Lifecycle and States

The lifecycle of an ACE is defined by the availability and reachability of the Root Entropy Value (REV).

#### 3.3.1. Operational States

| State | Description | Persistence |
|---|---|---|
| *Sealed* | The REV is encrypted within a Sealed Artifact (SA). | Persistent (Disk/ Storage) |
| *Reconstructed* | The REV is unsealed and resides in volatile memory. | Ephemeral (RAM only) |
| *Zeroized* | Sensitive material has been wiped. Identity is dormant. | N/A |
| *Unreachable* | The SA is destroyed or Credentials lost. | Permanent (Cryptographically Unreachable) |

Table 1

#### 3.3.2. Lifecycle Sequence

1. *Generate*: A 256-bit REV is sampled from a CSPRNG.

2. *Seal*: The REV is encrypted with a key derived from a Credential and Salt, producing a *Sealed Artifact (SA)*. The raw REV is then zeroized.

3. *Unseal*: When an operation is required, the user provides the Credential to decrypt the SA, restoring the REV to RAM.

4. *Derive*: The REV is used to generate keys for specific contexts (e.g., Ed25519, ML-DSA).

5. *Zeroize*: Once complete, the REV and all derived material are wiped from RAM.

### 3.4. Core Operations: Rekeying and Identity Unreachability

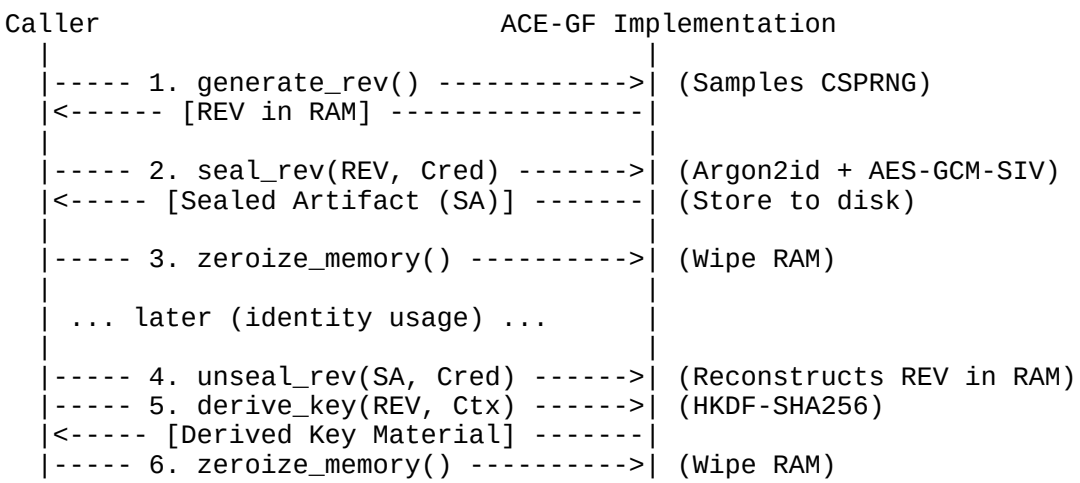
By decoupling authorization from identity, ACE-GF enables critical operations without changing underlying cryptographic identifiers (e.g., blockchain addresses):

* *Stateless Rekeying*: To change a password, the entity unseals the REV and re-seals it using a _new_ Credential and _new_ Salt. The REV remains invariant; thus, all derived public keys remain stable.

* *Identity Unreachability*: An identity becomes unreachable if the Sealed Artifact (SA) or required authorization inputs are lost or destroyed. Without these, the REV is mathematically lost, rendering the identity cryptographically unreachable without reliance on external revocation infrastructure.

### 3.5. Local Call Sequence

The following sequence describes the typical interaction between an application ("Caller") and an ACE-GF implementation:




## 4. Architecture and Design Principles

ACE-GF separates the identity lifecycle into two distinct pipelines:

### 4.1. Identity Pipeline

The identity pipeline manages the transformation from the REV to application-specific keys. It utilizes a deterministic, one-way derivation function (HKDF) combined with explicit context strings. This ensures that compromise of a specific child key (e.g., an Ed25519 signing key) does not leak information about the REV or other sibling keys.

### 4.2. Authorization Pipeline

The authorization pipeline manages the secure "sealing" and "unsealing" of the REV. By utilizing Argon2id for memory-hard key stretching and AES-GCM-SIV for nonce-misuse resistant encryption, the framework ensures that the REV is only accessible when the correct Credential is provided.

## 5. Cryptographic Primitives

The ACE-GF framework relies on industry-proven cryptographic primitives. Implementations MUST strictly adhere to the selection of these primitives to ensure deterministic consistency across different platforms and environments.

### 5.1. KDF1: Credential Hashing (Argon2id)

To resist offline brute-force attacks, ACE-GF utilizes the Argon2id algorithm *[RFC9106]* for stretching user-provided credentials (Cred) into a symmetric sealing key (K_seal).

The specific mechanism by which auth_index is bound into the sealing key derivation is an implementation detail, provided that distinct values of auth_index result in cryptographically independent sealing keys.

* *Algorithm Selection*: Implementations MUST use the Argon2id variant (as opposed to Argon2i or Argon2d) to provide optimized protection against both side-channel attacks and GPU-based cracking.

* *Parameter Negotiation*: Implementations SHOULD support pre-defined Sealing Profiles that specify memory cost (M), time iterations (T), and parallelism (P).

* *Salt*: Each Sealed Artifact (SA) MUST include a 128-bit random salt sampled from a CSPRNG. For the purpose of binding the authorization epoch, the salt input to the Argon2id function MUST be a 20-byte sequence, constructed by concatenating the 16-byte random salt and the 4-byte Big-Endian representation of the auth_index.

* *Credential Encoding*: To ensure deterministic consistency across different platforms and programming languages, the Authorization Credential (Cred) MUST be encoded as a UTF-8 string *[RFC3629]* before being provided as the password input to the Argon2id function.

### 5.2. AEAD: Sealing Encryption (AES-GCM-SIV)

ACE-GF employs AES-GCM-SIV *[RFC8452]* for the authenticated encryption of the Root Entropy Value (REV).

AES-256-GCM-SIV is used exclusively for authorization sealing of the Root Entropy Value (REV). It is not part of the Derive operation and does not correspond to any AlgID value in the Context Identifier Registry.

* *Misuse Resistance*: AES-GCM-SIV is selected for its Synthetic Initialization Vector (SIV) properties. In the specific context of ACE-GF, given that the REV is a 256-bit high-entropy random value, using a fixed 96-bit all-zero nonce (N_fixed) is safe under the assumption that the plaintext (REV) is a uniformly random,

high-entropy value and that the sealing key is not reused for arbitrary plaintexts. This usage constitutes a restricted, single-message sealing construction and MUST NOT be generalized beyond the specific case defined in this document. This assumption holds only for the sealing of a single, uniformly random REV per Sealed Artifact (SA) and MUST NOT be extended to sealing arbitrary data or multiple plaintexts under the same key.

* *Security Assurance*: The use of AES-GCM-SIV prevents confidentiality leaks even in cases of nonce reuse, eliminating the dependency on high-fidelity hardware Random Number Generators (RNG) during every sealing operation.

* *Data Structure*: The encryption process MUST produce a 256-bit ciphertext and a 128-bit authentication tag.

* *Additional Authenticated Data (AAD)*: To ensure the integrity of the authorization metadata, the AEAD encryption MUST supply the first 10 bytes of the Sealed Artifact (SA) as Additional Authenticated Data. This AAD string consists of the Magic Number (4 bytes), Version (1 byte), ProfileID (1 byte), and *the Authorization Index (4 bytes)*. All multi-byte fields MUST be encoded in Big-Endian (Network Byte Order).

* *Authenticated Metadata Binding*: Implementations MUST supply the Authorization Index (auth_index) as Additional Authenticated Data (AAD) to AES-256-GCM-SIV. This ensures that any modification of authorization metadata results in authentication failure during unsealing. Failure to bind auth_index via AAD constitutes a violation of this specification.

### 5.3. KDF2: Key Derivation (HKDF-SHA256)

For deriving algorithm-specific keys from the REV, ACE-GF utilizes HKDF *[RFC5869]*.

* *Core Logic*: The standard Extract-then-Expand workflow is followed.

* *Extract*: Map the REV into a cryptographically strong Pseudorandom Key (PRK).

* *Expand*: Input the structured Context tuple (Ctx) as the ’info’ parameter to generate target keys of specific lengths.

* *Hash Function*: SHA-256 MUST be used.

* *Isolation Guarantee*: By explicitly encoding AlgID, Domain, and Index into the 'info' string, HKDF ensures that the resulting child keys are computationally independent.

# 6. Protocol Specification

This section defines the operational procedures for the ACE-GF framework. All cryptographic operations MUST follow the sequences described herein to maintain cross-platform interoperability.

## 6.1. Root Entropy Value (REV) Generation

This operation generates the Root Entropy Value (REV), which serves as the atomic foundation of an ACE identity.

### 6.1.1. Inputs

* None.

### 6.1.2. Outputs

* *REV*: A freshly generated Root Entropy Value.

* *Error*: On failure.

### 6.1.3. Processing Requirements

1. The REV MUST be a 256-bit (32-byte) value.

2. The REV MUST be sampled from a cryptographically secure random number generator (CSPRNG) with uniform distribution.

3. The REV MUST NOT be stored in persistent storage in plaintext form. It MUST exist only in volatile memory during active use.

### 6.1.4. Error Conditions

* *EntropyUnavailable*: A suitable CSPRNG is not available or fails to produce sufficient entropy.

### 6.1.5. Error Handling Requirements

If REV generation fails, implementations MUST return an error and MUST NOT proceed with any sealing or derivation operations.

### 6.2. Sealing Process (Seal)

The Sealing process transforms a Root Entropy Value (REV) into a persistent Sealed Artifact (SA) using an Authorization Credential.

#### 6.2.1. Inputs

* *REV*: The Root Entropy Value to be protected.

* *Authorization Credential (Cred)*: Secret material used to derive the sealing key.

* *Sealing Profile*: Identifies the Argon2id parameters to be used.

* *Authorization Index (auth_index)*: A non-negative integer identifying the logical authorization epoch associated with this Sealed Artifact.

The Authorization Index (auth_index) is not a user-supplied secret and is intended to be managed transparently by implementations. Its value is carried within the Sealed Artifact and SHOULD NOT require manual tracking by users.

#### 6.2.2. Outputs

* *Sealed Artifact (SA)*: A serialized artifact containing the encrypted REV, authorization metadata, and associated parameters.

* *Error*: On failure.

#### 6.2.3. Processing Steps

1. Generate or select a 128-bit random Salt.

2. Derive the sealing key K_seal using Argon2id. *The Cred input MUST be encoded as a UTF-8 string*. The salt parameter input to Argon2id MUST be the 20-byte concatenation: Salt (16 bytes) || auth_index (4 bytes, Big-Endian).

3. Encrypt the REV using AES-256-GCM-SIV with K_seal and the fixed nonce N_fixed. The AAD input MUST be the 10-byte sequence: Magic || Version || ProfileID || auth_index, where auth_index *is serialized as a 32-bit unsigned integer in Big-Endian (Network Byte Order)*.

4. Construct the Sealed Artifact (SA) by concatenating the Magic Number, Version, ProfileID, auth_index, Salt, Ciphertext, and Authentication Tag, as specified in Section 6.1.

5. Zeroize the REV from volatile memory after successful sealing.

#### 6.2.4. Error Conditions

* *InvalidREV*: The supplied REV does not meet length or format requirements.
* *ProfileUnsupported*: The specified Sealing Profile is not supported.
* *InvalidAuthIndex*: The provided auth_index is malformed or unsupported.
* *SealingFailure*: Encryption or key derivation fails.

#### 6.2.5. Error Handling Requirements

If sealing fails, implementations MUST return an error and MUST ensure that the REV and any intermediate key material are not retained in memory.

### 6.3. Unsealing Process (Unseal)

The Unsealing process reconstructs the Root Entropy Value (REV) from a provided Sealed Artifact (SA) and Authorization Credential.

#### 6.3.1. Inputs

* *Sealed Artifact (SA)*: A serialized artifact containing the encrypted REV and associated metadata.
* *Authorization Credential (Cred)*: User- or system-provided secret material required to derive the sealing key.

#### 6.3.2. Outputs

* *REV*: The reconstructed Root Entropy Value, on success.
* *Error*: On failure.

#### 6.3.3. Processing Steps

1. *Parse*: Extract the Magic Number, Version, ProfileID, auth_index, Salt, Ciphertext, and Authentication Tag from the SA. The Magic Number and Version MUST be validated. Failure MUST result in InvalidFormat.

2. *Derive*: Recompute the sealing key K_seal using the *UTF-8 encoded* Credential, extracted Salt, auth_index, and the parameters associated with the ProfileID.

3. *Decrypt*: Execute AES-256-GCM-SIV decryption on the Ciphertext using K_seal, the fixed nonce N_fixed, and the Authentication Tag. The same Additional Authenticated Data (AAD) used during sealing (*constructed by concatenating Magic, Version, ProfileID, and the Big-Endian encoded* auth_index) MUST be supplied.

#### 6.3.4. Error Conditions

* *AuthenticationFailure*: Authentication tag verification fails.

* *InvalidFormat*: The SA cannot be parsed or contains unsupported version or profile identifiers.

* *ProfileUnsupported*: The referenced ProfileID is not implemented.

* *AuthorizationMismatch*: The derived sealing key does not match the authorization metadata embedded in the SA.

#### 6.3.5. Error Handling Requirements

If any error occurs, implementations MUST return an error and MUST NOT release any portion of the decrypted REV. Implementations SHOULD ensure that error handling does not introduce observable timing differences that could leak information about the Credential or K_seal.

### 6.4. Key Derivation Function (Derive)

The Derive operation deterministically produces algorithm-specific cryptographic key material from a reconstructed Root Entropy Value (REV).

#### 6.4.1. Inputs

* *REV*: The reconstructed Root Entropy Value.

* *Context Tuple (Ctx)*: A structured context specifying algorithm, usage domain, and key index.

* *Output Length*: The required length of derived key material.

#### 6.4.2. Outputs

* *DerivedKey*: Algorithm-specific key material.

* *Error*: On failure.

#### 6.4.3. Processing Steps

1. Perform HKDF-Extract using the REV as the input key material (IKM) to produce a pseudorandom key (PRK). The salt parameter for HKDF-Extract MUST be a 32-byte string of all zeros.

2. Perform HKDF-Expand using the PRK and the serialized Context Tuple (Ctx) as the info parameter to generate key material of the requested length.

3. Map the derived output to algorithm-specific key material according to the target algorithm’s requirements.

#### 6.4.4. Error Conditions

* *InvalidContext*: The Context Tuple is malformed or unsupported.

* *DerivationFailure*: HKDF expansion fails or produces invalid output for the target algorithm.

#### 6.4.5. Error Handling Requirements

On failure, implementations MUST return an error and MUST NOT release partial or malformed key material. The REV MUST remain protected in volatile memory and SHOULD be zeroized once derivation operations are complete.

## 7. Data Structures and Encodings

### 7.1. Sealed Artifact (SA) Binary Format (Version 0x01)

This section defines the binary encoding of the Sealed Artifact (SA) for protocol version 0x01. The SA encapsulates the encrypted Root Entropy Value (REV) together with all metadata required for deterministic reconstruction.

All multi-byte integer fields MUST be encoded in Big-Endian (Network Byte Order). Implementations MUST reject any SA whose total length does not exactly match the expected length for the indicated Version.

### 7.1.1. SA Binary Layout (Version 0x01)

| Offset | Length | Field | Description |
|---|---|---|---|
| 0 | 4 | Magic Number | Fixed bytes: 0x41 0x43 0x45 0x00 ("ACE\0") |
| 4 | 1 | Version | Protocol version (0x01) |
| 5 | 1 | ProfileID | Identifier for Sealing Profile |
| 6 | 4 | auth_index | Authorization Index (Unsigned 32-bit) |
| 10 | 16 | Salt | Random salt for Argon2id |
| 26 | 32 | Ciphertext | AES-256-GCM-SIV encrypted REV |
| 58 | 16 | Tag | Authentication Tag (MAC) |

Table 2

*Total Length: 74 bytes*

### 7.1.2. Field Semantics

* *auth_index*: The Authorization Index represents an authorization epoch associated with the Sealed Artifact. Different values of auth_index under the same REV represent distinct authorization states. Implementations MUST ensure that Sealed Artifacts generated with different auth_index values are cryptographically independent and mutually non-decryptable.

## 7.2. Sealing Profiles

Sealing Profiles define the cost parameters for the Argon2id function. This allows ACE-GF to scale from resource-constrained mobile devices to high-security server environments.

| Profile ID | Label | Memory (MiB) | Iterations (T) | Parallelism (P) |
|---|---|---|---|---|
| 0x01 | Mobile | 64 | 3 | 1 |
| 0x02 | Standard | 256 | 4 | 2 |
| 0x03 | Paranoid | 1024 | 8 | 4 |

Table 3

Implementations MUST support at least the 'Standard' (0x02) profile. The 'Paranoid' profile is RECOMMENDED for high-value administrative identities.

### 7.3. Context Labels

To ensure computational independence between different cryptographic algorithms, the AlgID field in the Context Tuple MUST use the following initial registry. This ensures that a key derived for Ed25519 cannot be mistakenly used or mathematically linked to a Secp256k1 key.

Usage Domains describe the functional intent of derived key material and do not override algorithm-specific restrictions defined elsewhere in this document.

#### 7.3.1. Algorithm Identifiers (AlgID)

| AlgID | Algorithm Name | Reference |
|---|---|---|
| 0x0001 | Ed25519 | *[RFC8032]* |
| 0x0002 | Secp256k1 (ECDSA) | *[SEC2]* |
| 0x0003 | X25519 (Diffie-Hellman) | *[RFC7748]* |
| 0x0004 | ML-DSA (Dilithium-PQC) | *[FIPS204]* |
| 0x0005 | ML-KEM (Kyber-PQC) | *[FIPS203]* |

Table 4

#### 7.3.2. Usage Domains (Domain)

The 8-bit Domain field separates keys by their functional intent:

* *0x01 (Signing)*: Primary identity signatures.
* *0x02 (Encryption)*: Data at rest or in transit.
* *0x03 (Authentication)*: Challenge-response protocols.
* *0x04 (Key Wrapping)*: Protecting other sub-keys.

### 7.4. Context Tuple Encoding

The Context Tuple (Ctx) is a compact, fixed-length binary structure used as the info parameter for HKDF expansion. All fields are encoded in Big-Endian (Network Byte Order).

#### 7.4.1. Context Tuple Binary Layout

| Offset | Length | Field | Description |
|---|---|---|---|
| 0 | 2 | AlgID | Algorithm Identifier |
| 2 | 1 | Domain | Usage Domain |
| 3 | 4 | Index | Key Index (unsigned integer) |

Table 5

*Total Length: 7 bytes.*

#### 7.4.2. Encoding Rules

* The AlgID field MUST correspond to a registered value in the ACE-GF Context Identifier Registry.
* The Domain field MUST correspond to a registered Usage Domain.
* The Index field is an unsigned 32-bit integer and MAY be incremented to derive multiple independent keys within the same algorithm and domain.

# 8. Operational Considerations and Identity Lifecycle

This section describes operational aspects of deploying and managing Atomic Cryptographic Entities (ACEs) using the ACE-GF framework. It focuses on lifecycle management, authorization updates, identity unreachability semantics, and deployment considerations. This section is informational and does not introduce new protocol requirements beyond those specified elsewhere in this document.

One of the primary advantages of the ACE-GF framework is its ability to manage the full identity lifecycle without requiring changes to the underlying Root Entropy Value (REV). Authorization material may be updated or rendered unavailable independently, while derived cryptographic identifiers remain stable as long as the REV remains reachable.

## 8.1. Sealed Artifact Persistence and Mobility

The Sealed Artifact (SA) is a persistent, encrypted authorization artifact, not a cryptographic root. Possession of an SA alone does not enable identity reconstruction without the corresponding Authorization Credential.

Because the SA contains no plaintext secret material and reveals no information about the Root Entropy Value (REV), it MAY be stored, replicated, and transported using untrusted storage mechanisms, including public cloud storage or distributed content-addressable systems.

Loss of all copies of the SA renders the identity cryptographically unreachable. Applications SHOULD treat the SA as durable authorization metadata and apply backup and redundancy strategies appropriate to their threat model.

## 8.2. Stateless Credential Rotation

ACE-GF supports stateless credential rotation, allowing an entity to update its Authorization Credential (e.g., changing a password or authorization factor) without altering the underlying Root Entropy Value (REV).

Because the REV is the sole source of all derived cryptographic keys, this process preserves identity continuity. No derived public keys, addresses, or identifiers need to be regenerated or re-announced.

The rotation process is performed as follows:

1. *Unseal*: Use the current Authorization Credential and the existing Sealed Artifact (SA) to reconstruct the REV in volatile memory.

2. *Generate New Salt*: Sample a new 128-bit random salt value.

3. *Re-Seal*: Perform the Sealing Process (as defined in Section 4.2) using the new Authorization Credential and the new salt, while maintaining the same REV.

4. *Commit*: Replace the previous SA with the newly generated SA.

This operation is stateless with respect to identity semantics. No persistent state other than the updated SA is required, and no protocol- visible identity attributes are modified.

### 8.3. Logical Revocation via Authorization Index

ACE-GF defines _logical revocation_ as the ability to render a previously valid Sealed Artifact (SA) unusable without modifying the underlying Root Entropy Value (REV).

An ACE-GF Identity is defined by the ability to reconstruct the REV. Logical revocation invalidates specific authorization states while preserving the identity itself and all derived cryptographic identifiers.

#### 8.3.1. Mechanism

Logical revocation is achieved by updating the Authorization Index (auth_index) and re-sealing the same REV:

1. The current SA is unsealed to reconstruct the REV in volatile memory.

2. The auth_index value is incremented or otherwise updated according to application policy.

3. The REV is re-sealed under the new auth_index, producing a new SA.

4. The previous SA is discarded or rendered inaccessible.

Because auth_index is cryptographically bound into the sealing key derivation, any SA created under a prior authorization index becomes undecryptable once the active index changes.

#### 8.3.2. Properties

* Logical revocation does not require regeneration of the REV.
* Logical revocation does not require identifier rotation (e.g., blockchain address changes).
* Logical revocation does not rely on external revocation infrastructure such as CRLs or OCSP.
* The effect of logical revocation is immediate and purely cryptographic.

#### 8.3.3. Scope and Policy

This specification defines the cryptographic mechanism required to support logical revocation. Policies governing authorization index management, distribution of updated Sealed Artifacts, recovery procedures, and governance models are application-specific and out of scope.

#### 8.3.4. Authorization Index Synchronization

In multi-device or multi-instance deployments, applications are responsible for ensuring that the most recent Sealed Artifact is distributed to all authorized endpoints. Implementations SHOULD treat the Sealed Artifact as versioned authorization state and ensure that stale artifacts are replaced when authorization updates occur.

The ACE-GF protocol does not require global state synchronization and does not define mechanisms for resolving concurrent authorization updates.

### 8.4. Failure Modes and Recovery Considerations

Failure to reconstruct the REV may occur due to incorrect credentials, corrupted Sealed Artifacts, unsupported sealing profiles, or intentional revocation actions.

Implementations SHOULD treat all unsealing failures as indistinguishable from an external observability perspective and MUST NOT leak partial information about the REV or derived keys in failure scenarios.

Recovery is possible only if sufficient authorization material remains available. Loss of all authorization inputs results in permanent unreachability of the REV.

### 8.5. Storage and Ephemeral State Considerations

The Root Entropy Value (REV) is intended to exist only in volatile memory during active operations and MUST NOT be persistently stored.

The Sealed Artifact (SA) is the sole persistent representation required to reconstruct an ACE identity. It MAY be stored, transmitted, or backed up using conventional storage mechanisms, subject to application-specific security policies.

Implementations SHOULD ensure timely zeroization of all sensitive material, including the REV, sealing keys, and intermediate buffers, after use. Where available, hardware-backed isolation mechanisms such as Trusted Execution Environments (TEEs) are RECOMMENDED to reduce exposure risk during reconstruction and derivation operations.

### 8.6. Migration Considerations

Existing cryptographic identities MAY be migrated into the ACE-GF framework by treating a legacy high-entropy private key as a Root Entropy Value (REV) and sealing it using standard ACE-GF procedures.

Such migrations are OPTIONAL and profile-dependent. Implementations SHOULD document any deviations from native ACE-GF derivation semantics and SHOULD clearly distinguish between natively generated ACE-GF identities and migrated identities when relevant.

## 9. Security Considerations

The security of the ACE-GF framework depends on the strength of the underlying cryptographic primitives and the rigor of the implementation environment.

### 9.1. Entropy Requirements for Credentials

The security of the Sealed Artifact (SA) is directly proportional to the entropy of the Authorization Credential (Cred).

1. *Minimum Entropy*: Human-provided credentials SHOULD be long, randomly generated passphrases managed by password managers. Machine-generated credentials MUST provide at least 128 bits of entropy.

2. *Entropy Stretching*: While Argon2id provides significant resistance against brute-force, it cannot compensate for extremely weak secrets (e.g., short, common passwords).

3. *Machine-Generated Credentials*: For Autonomous Digital Entities (ADEs), credentials MUST be generated using a CSPRNG.

### 9.2. Protection of Volatile Memory and Memory Management

Since the Root Entropy Value (REV) serves as the atomic foundation for all derived identities, its exposure in volatile memory constitutes a single point of failure. An attacker capable of observing the memory space of an active ACE-GF implementation could compromise the entire identity hierarchy derived from that REV.

#### 9.2.1. Isolated Execution Environments (IEE)

It is STRONGLY RECOMMENDED that implementations perform the Unsealing (Section 4.3) and Derivation (Section 4.4) operations within an Isolated Execution Environment (IEE). An IEE is a platform-provided security boundary that ensures:

* *Memory Confidentiality*: Sensitive material, including the REV, Authorization Credentials, and the Sealing Key (K_seal), MUST NOT be observable by the host operating system, hypervisor, or other unauthorized processes.

* *Execution Integrity*: The cryptographic logic of ACE-GF MUST be protected from unauthorized modification during runtime.

* *Isolation Guarantee*: Even in the event of a total compromise of the Host OS, the secrets within the IEE remain protected.

Implementations SHOULD leverage hardware-backed technologies to satisfy these requirements. Examples of such environments include, but are not limited to: * *Hardware Enclaves*: Intel SGX, RISC-V Keystone. * *Trust Zones*: ARM TrustZone. * *Virtualization-based Security*: AWS Nitro Enclaves, Azure Confidential Computing (SNP/ TDX).

#### 9.2.2. Memory Hardening and Zeroization

When a hardware-isolated IEE is unavailable, or as a defense-in-depth measure within an IEE, implementations MUST adhere to the following memory management principles to mitigate "cold boot" attacks or memory forensics:

1. *Zeroization*: All volatile memory buffers containing the REV, K_seal, or raw Authorization Credentials MUST be overwritten with zeros (Zeroized) immediately following the conclusion of an operation or upon process termination. Implementations MUST use platform-specific mechanisms (e.g., memset_s in C, Zeroize trait in Rust) to ensure that compilers do not optimize away these routines.

2. *Anti-Swapping*: Memory regions used for sensitive material SHOULD be marked as non-swappable to prevent them from being written to persistent storage (e.g., using mlock() on POSIX or VirtualLock() on Windows). This prevents secrets from being leaked via swap files, core dumps, or hibernation images.

3. *Side-Channel Mitigation*: Implementations MUST ensure that the transformation logic, especially the handling of K_seal and REV, is resistant to timing attacks and other micro-architectural side-channels. The use of constant-time cryptographic primitives is REQUIRED.

4. *Process Isolation*: ACE-GF operations SHOULD be executed in a dedicated process with minimal privileges (least privilege principle) to reduce the attack surface from other resident applications.

### 9.3. Resistance to Brute-Force Attacks

The use of Argon2id allows ACE-GF to scale its defense according to the threat model. The following table provides a theoretical analysis of attack costs across different Profiles:

| Profile | Target Device | Attacker Cost Assumption (Memory-Hardness) |
|---|---|---|
| Mobile | Low-Power IoT | Balanced for battery life; vulnerable to high-end GPU clusters. |
| Standard | Desktop/ Web | Resistant to mid-scale commodity GPU attacks. |
| Paranoid | Server/ HSM | Prohibitively expensive for all but state-actor level adversaries due to high memory bandwidth requirements (1GB+ per attempt). |

Table 6

## 9.4. Post-Quantum Transition and Hybrid Security

ACE-GF is designed with "Algorithm Agility" to survive the transition to Post-Quantum Cryptography (PQC).

1. *Context-Isolated PQC*: As defined in Section 5.3, the framework allows the derivation of PQC keys (e.g., ML-DSA) alongside classical keys (e.g., Ed25519) from the same REV.

2. *Hybrid Derivation*: For maximum security during the transition period, implementations MAY use a hybrid approach where a classical signature and a PQC signature are required to authorize a single action.

3. *Future-Proofing REV*: Because the REV is a high-entropy (256-bit) random value, it provides an effective 128-bit preimage security margin against generic quantum search attacks (e.g., Grover-style algorithms [Grover1996]), under standard complexity assumptions.

## 9.5. Note on AES-GCM-SIV and Fixed Nonce

The use of a fixed all-zero nonce (N_fixed) in Section 4.2.2 is safe specifically because the plaintext being encrypted (the REV) is guaranteed to be a unique, high-entropy 256-bit value for every Sealed Artifact. Therefore, the SIV (Synthetic Initialization Vector) derivation will inherently produce unique sub-keys for the underlying CTR mode, preventing the catastrophic key-stream reuse associated with standard AES-GCM.

## 10. IANA Considerations

This section describes a proposed registry structure. Creation of IANA registries is contingent on the publication status of this document.

This document requests IANA to create three new registries for the Atomic Cryptographic Entity Generative Framework (ACE-GF).

The following registries are defined for future extensibility of the ACE-GF framework.

### 10.1. ACE-GF Sealing Profile Registry

IANA is requested to create a new registry entitled "ACE-GF Sealing Profiles". This registry manages the parameter sets for the credential hashing function (Argon2id).

The registration policy for this registry is "Specification Required" as defined in *[RFC8126]*.

Initial entries for this registry are as follows:

| Profile ID | Label | Memory (MiB) | Iterations (T) | Parallelism (P) | Reference |
|---|---|---|---|---|---|
| 0x01 | Mobile | 64 | 3 | 1 | [This RFC] |
| 0x02 | Standard | 256 | 4 | 2 | [This RFC] |
| 0x03 | Paranoid | 1024 | 8 | 4 | [This RFC] |
| 0x04-0xFF | Unassigned | | | | |

Table 7

### 10.2. ACE-GF Context Identifier Registry

IANA is requested to create a new registry entitled "ACE-GF Context Identifiers". This registry manages the Algorithm Identifiers (AlgID) used in the key derivation context string.

The registration policy for this registry is "Expert Review" as defined in *[RFC8126]*.

Initial entries for this registry are as follows:

| AlgID | Algorithm Name | Reference |
|---|---|---|
| 0x0001 | Ed25519 | [RFC8032] |
| 0x0002 | Secp256k1 | [SEC2] |
| 0x0003 | X25519 | [RFC7748] |
| 0x0004 | ML-DSA (Dilithium) | [FIPS204] |
| 0x0005 | ML-KEM (Kyber) | [FIPS203] |
| 0x0006-0xFFFF | Unassigned | |

Table 8

### 10.3. ACE-GF Usage Domain Registry

IANA is requested to create a new registry entitled "ACE-GF Usage Domains". This registry manages the 8-bit Domain field.

| Domain ID | Description | Reference |
|---|---|---|
| 0x01 | Signing | [This RFC] |
| 0x02 | Encryption | [This RFC] |
| 0x03 | Authentication | [This RFC] |
| 0x04 | Key Wrapping | [This RFC] |
| 0x05-0xFF | Unassigned | |

Table 9

## 11. Test Vectors

This section defines test vectors intended to verify the correctness and interoperability of ACE-GF implementations.

The test vectors in Sections 10.1 through 10.3 define normative behavior with respect to input handling, parameter binding, and deterministic reconstruction. Ciphertext values are illustrative and not required to match across implementations unless explicitly specified.

Certain encrypted outputs are omitted in this version and marked as TBD. These vectors validate structure, parameter selection, and deterministic REV reconstruction rather than ciphertext equality.

### 11.1. Basic Protocol Test Vectors

#### 11.1.1. Standard Sealing Profile

This test case uses the "Standard" Sealing Profile (0x02).

* *Credential*: "password123"

* *Salt*: 0102030405060708090a0b0c0d0e0f10

* *Argon2id Parameters*: M=256MiB, T=4, P=2

* *REV*: f0e1d2c3b4a5968778695a4b3c2d1e0f00112233445566778899aabbccddeeff

*Output - Sealed Artifact (SA)*: 41434500 (Magic) 01 (Version) 02 (ProfileID) 00000001 (auth_index) 0102030405060708090a0b0c0d0e0f10 (Salt) [Insert 32-byte Hex Ciphertext here] [Insert 16-byte Hex Tag here]

### 11.2. Cross-Platform and Encoding Consistency

#### 11.2.1. UTF-8 Credential Handling

This test case ensures that non-ASCII credentials are handled consistently using UTF-8 encoding before the Argon2id process.

* *Credential*: "密123" (UTF-8: e5af86e7a081313233)

* *Salt*: f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff

* *ProfileID*: 0x01 (Mobile)

*Output - Reconstructed REV*: [Insert 32-byte Hex REV here]

### 11.3. Multi-Algorithm and Post-Quantum Derivation

#### 11.3.1. Classical + PQC Key Derivation from a Single REV

This test case demonstrates the generative nature of the framework by deriving keys for both classical and post-quantum algorithms from the same REV.

* *REV*: 603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4

#### 11.3.2. Case A: Ed25519 Signing Key

* *AlgID*: 0x0001
* *Domain*: 0x01
* *Index*: 0
* *Context Info (Hex)*: 00010100000000
* *Derived Key (32 bytes)*: [Insert Hex Key here]

#### 11.3.3. Case B: ML-KEM (Kyber) Key

* *AlgID*: 0x0005
* *Domain*: 0x02
* *Index*: 0
* *Context Info (Hex)*: 00050200000000
* *Derived Key (64 bytes)*: [Insert Hex Key here]

### 11.4. Application Profile Test Vectors (Informational)

#### 11.4.1. Cryptocurrency Wallet Application Profile

This section defines test vectors for the *ACE-GF Cryptocurrency Wallet Application Profile*.

These test vectors are intended to demonstrate the deterministic behavior of ACE-GF when applied to a multi-chain cryptocurrency wallet context, including mnemonic generation, passphrase-based rekeying, identity reconstruction ("view"), and migration from legacy wallet mnemonics.

These vectors *DO NOT* validate the ACE-GF protocol encoding, wire format, or generic file representations defined elsewhere in this document. Instead, they apply exclusively to implementations that claim conformance to the wallet application profile.

Implementations that do not support this application profile are not required to produce identical results.

The following test vectors are expressed in JSON format for readability and ease of cross-platform testing.


```
{
"description": "ACE-GF Deterministic Identity Test Vectors V1.0",
"project_version": "2025.12.23",
"test_cases": [
{
"case_id": "TEST_CASE_001",
"name": "Initial_Generation_ASCII_Passphrase",
"description": "Generate a new ACE-GF identity using ASCII passphrase ’abc’. Establish
es Identity #1.",
"command": "./acegf generate abc",
"input": {
"passphrase": "abc"
},
"expected_output": {
"mnemonic": "vintage crash medal recycle item pigeon error real join december image in
to toe bag coffee pyramid gorilla tank scrub drift pencil clap vacant unlock",
"solana": "5yFK8sUGK6Ng5TqENvnLhwHRPDzyTMzLLp5ViUyx2ppk",
"evm": "0x1aCDA9f78D27e433317de8b14F50bb333df5055e",
"bitcoin": "bc1pnjptwsfjpukkj4xcw20fm5ddjrfqqquahj34q4lkd6jemxnvs84s5wgs2u",
"cosmos": "cosmos1kzuju6s08kacthf45vd3raexjfuggrx2n9y6aq",
"polkadot": "16VnAyYpbPgnVzS2TEHeZK7LMegJ8dTDez68KUbrsS3D2thU",
"xidentity": "ZgjpfHPPDtpT0vHezh5inQW/ruBmDQTDuNMyrCWLzhw="
}
},
{
"case_id": "TEST_CASE_002",
"name": "View_Deterministic_Restore_ASCII",
"description": "Restore Identity #1 using the original mnemonic and passphrase ’abc’.
Must deterministically match TEST_CASE_001.",
"command": "./acegf view abc [mnemonic]",
"input": {
"passphrase": "abc",
"mnemonic": "vintage crash medal recycle item pigeon error real join december image in
to toe bag coffee pyramid gorilla tank scrub drift pencil clap vacant unlock"
},
"expected_output_ref": "TEST_CASE_001"
},
{
"case_id": "TEST_CASE_003",
"name": "Rekey_ASCII_to_ASCII",
"description": "Rekey Identity #1 by changing passphrase from ’abc’ to ’cde’.",
```

```
   "command": "./acegf rekey abc cde [mnemonic]",
   "input": {
   "old_passphrase": "abc",
   "new_passphrase": "cde",
   "old_mnemonic": "vintage crash medal recycle item pigeon error real join december imag
e into toe bag coffee pyramid gorilla tank scrub drift pencil clap vacant unlock"
   },
   "expected_output": {
   "new_mnemonic": "stuff roast enable clown casual crazy swing sun spoil home derive oce
an screen frog shoot devote memory fall chuckle float canvas blue sudden quick"
   }
   },
   {
   "case_id": "TEST_CASE_004",
   "name": "View_After_Rekey_ASCII",
   "description": "Restore Identity #1 using new mnemonic and new passphrase ’cde’. Addre
sses must match TEST_CASE_001.",
   "command": "./acegf view cde [new_mnemonic]",
   "input": {
   "passphrase": "cde",
   "mnemonic": "stuff roast enable clown casual crazy swing sun spoil home derive ocean s
creen frog shoot devote memory fall chuckle float canvas blue sudden quick"
   },
   "expected_output_ref": "TEST_CASE_001"
   },
   {
   "case_id": "TEST_CASE_005",
   "name": "Legacy_12_Word_Aceize",
   "description": "Convert a legacy 12-word mnemonic into an ACE-GF identity using passph
rase ’abc’.",
   "command": "./acegf aceize abc [12-word mnemonic]",
   "input": {
   "passphrase": "abc",
   "legacy_mnemonic": "figure pony hour transfer toilet blush easy sorry taste write swin
g neck"
   },
   "expected_output": {
   "mnemonic": "language issue crash subject warm when step shadow two live fantasy cake
crush key calm cabbage ready error sure idea dish rotate drama balance",
   "solana": "p9vabuSUkrntQwDNiA7X2xaiaBkwFn2SxJdmAFdHgKd",
   "evm": "0xe21AA588aEFA0AE391798A122f0994E9eD9406E5",
   "bitcoin": "bc1pet90lyvvca05c4zhme6925zr2kgdwjj564nm8kqhf758gvgp9fvs6px5g7",
   "cosmos": "cosmos1zvtxtz5vh500rh0k0n3laeh6z0t263yp8krl80",
   "polkadot": "132SUNr7yrVqTMYYACtHQSz9RCR1Pkw126BkF5PGdW892LoV",
   "xidentity": "DUZt+nGZE1ZpGJhW0MYdtibPQcEBKXMHYkZW5phnoTo="
   }
   },
   {
   "case_id": "TEST_CASE_006",
   "name": "Unicode_Passphrase_Emoji",
   "description": "Rekey identity using emoji passphrase. Tests UTF-8 and non-ASCII passp
hrase handling.",
   "command": "./acegf rekey cde  [mnemonic]",
   "input": {
   "old_passphrase": "cde",
   "new_passphrase": "",
```

"old_mnemonic": "mushroom unusual extend belt torch sketch limb symbol invest fury din
ner fly capital you love school logic banana mention wrist buffalo spike finger elite"
},
"expected_output": {
"new_mnemonic": "rabbit thing indoor flower brass blush distance sauce swear ramp guar
d decorate news girl belt kid embody spoil skate alone suit record park wisdom"
}
},
{
"case_id": "TEST_CASE_007",
"name": "Unicode_Passphrase_Chinese",
"description": "Rekey identity using Chinese UTF-8 passphrase ’’.",
"command": "./acegf rekey [mnemonic]",
"input": {
"old_passphrase": "",
"new_passphrase": "",
"old_mnemonic": "rabbit thing indoor flower brass blush distance sauce swear ramp guar
d decorate news girl belt kid embody spoil skate alone suit record park wisdom"
},
"expected_output": {
"new_mnemonic": "deliver truck spread mask fetch connect cute credit cigar garlic stud
ent forest decade exchange swap issue wage gauge eagle soup squirrel cash dog whale"
}
},
{
"case_id": "TEST_CASE_008",
"name": "View_After_Multi_Rekey_Unicode",
"description": "Restore identity after multiple rekey operations with Unicode passphra
ses. Addresses must remain invariant.",
"command": "./acegf view [mnemonic]",
"input": {
"passphrase": "",
"mnemonic": "deliver truck spread mask fetch connect cute credit cigar garlic student
forest decade exchange swap issue wage gauge eagle soup squirrel cash dog whale"
},
"expected_output_ref": "TEST_CASE_005"
}
]
}

### 11.5. 10.5. Interoperability Verification Guidance

An implementation claiming conformance to this specification SHOULD verify interoperability as follows:

1. Using the test vectors in Sections 10.1 and 10.2, confirm that: - The reconstructed REV exactly matches the expected value. - The serialized Sealed Artifact (SA) fields match the specified binary layout and contents.

2. Using the vectors in Section 10.3, confirm that: - Derived key material for different algorithms and domains is deterministic and context-isolated. - Classical and post-quantum derivations can coexist under the same REV.

3. Implementations supporting the Cryptocurrency Wallet Application Profile SHOULD additionally verify conformance using the vectors defined in Section 10.4.

Successful verification across independent implementations indicates interoperability of the ACE-GF construction.

# 12. References

## 12.1. Normative References

*[RFC2119]* Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

*[RFC3629]* Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003.

*[RFC5869]* Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010.

*[RFC8126]* Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017.

*[RFC8174]* Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

*[RFC8452]* Gueron, S., Langley, A., and Y. Lindell, "AES-GCM-SIV: Nonce-Misuse-Resistant Authenticated Encryption", RFC 8452, DOI 10.17487/RFC8452, April 2018.

*[RFC9106]* Biryukov, A., Dinu, D., and D. Khovratovich, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications", RFC 9106, DOI 10.17487/RFC9106, September 2021.

*[FIPS203]* NIST, "Module-Lattice-Based Key-Encapsulation Mechanism Standard", FIPS PUB 203, August 2024.

*[FIPS204]* NIST, "Module-Lattice-Based Digital Signature Standard", FIPS PUB 204, August 2024.

*[NIST-SP800-108]* Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009.

### 12.2. Informative References

*[BIP32]* Wuille, P., "Hierarchical Deterministic Wallets", February 2012.

*[BIP39]* Palatinus, M., et al., "Mnemonic code for generating deterministic keys", 2013.

*[RFC8032]* Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017.

*[RFC7748]* Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016.

*[SEC2]* Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", Version 2.0, January 2010.

*[DID-Core]* Drummond, R., et al., "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation, July 2022.

*[Grover1996]* Grover, L., "A Fast Quantum Mechanical Algorithm for Database Search", 1996.

## Appendix A. Implementation Considerations for Resource-Constrained Devices

For IoT devices with limited RAM that cannot support the ’Standard’ Argon2id profile (256 MiB), the following optimizations are RECOMMENDED:

1. *Offloading Sealing*: The Sealing process can be performed on a provisioning terminal. The device only needs to store the resulting Sealed Artifact (SA).

2. *Unsealing via TEE*: If the device features a Secure Element (SE) or TEE, the K_seal derivation SHOULD be pinned to the hardware UID to provide an additional layer of security even if the Credential is weak.

## Appendix B. Illustrative Python Pseudocode

An illustrative pseudocode representation of the ACE-GF derivation logic:

### B.1. Note

*Note:* The following pseudocode is provided for explanatory purposes only. It omits error handling, memory zeroization, and side-channel protections, and MUST NOT be used as a reference implementation. Conforming implementations MUST adhere to the normative requirements specified elsewhere in this document.

```
import struct
from cryptography.hazmat.primitives.ciphers.aead import AESGCMSIV
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes

# ---------------------------------------------------------------------
# Pseudocode placeholder for Argon2id
# ---------------------------------------------------------------------
def argon2id_kdf(password_bytes, salt_bytes, memory_mib, iterations, parallelism, out_len
):
    """
    Pseudocode representation of Argon2id.
    Implementations MUST use RFC 9106compliant Argon2id.
    """
    # This function is a placeholder.
    # A real implementation must call an Argon2id library.
    raise NotImplementedError("Argon2id KDF placeholder")


# ---------------------------------------------------------------------
# ACE-GF: Seal REV
# ---------------------------------------------------------------------
def ace_seal_rev(
    rev: bytes,
    credential: str,
    salt_16: bytes,
    auth_index: int,
    profile_id: int = 0x02,
):
    """
    Seal a 32-byte Root Entropy Value (REV) into a Sealed Artifact (SA).
    """

    assert len(rev) == 32
    assert len(salt_16) == 16

    # --- Extend salt with auth_index (Big-Endian) ---
    extended_salt = salt_16 + struct.pack(">I", auth_index)

# --- Select Argon2id parameters by profile ---
    if profile_id == 0x01:      # Mobile
```

```
        m, t, p = 64, 3, 1
    elif profile_id == 0x02:    # Standard
        m, t, p = 256, 4, 2
    elif profile_id == 0x03:    # Paranoid
        m, t, p = 1024, 8, 4
    else:
        raise ValueError("Unsupported Sealing Profile")

    # --- Derive sealing key using Argon2id ---
    k_seal = argon2id_kdf(
        password_bytes=credential.encode("utf-8"),
        salt_bytes=extended_salt,
        memory_mib=m,
        iterations=t,
        parallelism=p,
        out_len=32,  # 256-bit AES key
    )

    # --- Construct Additional Authenticated Data (AAD) ---
    # AAD = Magic || Version || ProfileID || auth_index
    aad = b"ACE\x00" + struct.pack(">BBI", 0x01, profile_id, auth_index)

    # --- Encrypt REV using AES-256-GCM-SIV ---
    nonce = b"\x00" * 12  # Fixed nonce (restricted single-message usage)
    aead = AESGCMSIV(k_seal)

    ciphertext = aead.encrypt(nonce, rev, aad)

    return ciphertext  # Ciphertext || Tag (library-defined layout)


# ---------------------------------------------------------------------
# ACE-GF: Derive Algorithm-Specific Key Material
# ---------------------------------------------------------------------
def ace_derive_key(
    rev: bytes,
    alg_id: int,
    domain: int,
    index: int,
    length: int,
):
    """
    Derive context-isolated key material from REV using HKDF-SHA256.
    """

    assert len(rev) == 32

    # --- Context Tuple encoding ---
```

```
    info = struct.pack(">HBI", alg_id, domain, index)

    # --- HKDF Extract + Expand ---
    hkdf = HKDF(
        algorithm=hashes.SHA256(),
        length=length,
        salt=b"\x00" * 32,
        info=info,
    )

    derived_key = hkdf.derive(rev)
    return derived_key
```

## Appendix C. Example Use Cases (Informational)

This appendix provides illustrative, non-normative examples of how the ACE-GF framework may be applied in different deployment contexts. These examples are intended to aid understanding and do not impose additional protocol requirements.

### C.1. Autonomous Digital Entities (ADEs)

Autonomous Digital Entities such as AI agents may require a stable cryptographic identity that can be deterministically reconstructed across restarts without persistent secret storage.

In such deployments: - The Sealed Artifact (SA) may be stored in local or remote storage. - Authorization Credentials may be supplied at runtime by a controller or policy engine. - Derived keys may be used for signing, authentication, or secure communication between agents.

### C.2. IoT and Embedded Devices

IoT devices often operate under memory and storage constraints and may lack secure persistent storage for long-lived secrets.

ACE-GF allows: - Off-device provisioning of Sealed Artifacts. - On-device reconstruction of identity only when required. - Integration with Secure Elements or TEEs where available.

### C.3. Blockchain and Cryptographic Wallet Identities

Cryptographic wallets may use ACE-GF to derive multiple algorithm-specific keys (e.g., Ed25519, secp256k1, PQC) from a single REV.

Stateless rekeying allows credential updates without changing public addresses, while identity unreachability provides a recovery and lockout mechanism without reliance on external revocation infrastructure such as CRLs.

## Appendix D. Security Boundaries and Trust Assumptions (Informational)

This appendix summarizes the security boundaries and trust assumptions implicit in the ACE-GF framework. It is informational and complements the Security Considerations section.

### D.1. Trust Boundaries

ACE-GF assumes a clear separation between: - Authorization inputs (Credentials, salts, policy-controlled inputs) - Identity material (REV and derived keys) - Persistent storage (Sealed Artifacts)

The compromise of persistent storage alone does not reveal the REV or derived keys.

### D.2. Authorization Boundary

Authorization Credentials are assumed to be protected by the application or deployment environment. Weak credentials reduce the effective security of the Sealing Process but do not compromise the cryptographic soundness of the framework.

### D.3. Execution Environment Assumptions

During unsealing and derivation, the execution environment is assumed to provide basic process isolation. Where stronger guarantees are required, hardware-backed isolation (e.g., TEEs) is RECOMMENDED.

### D.4. Non-Goals

ACE-GF does not attempt to: - Protect against fully compromised execution environments. - Provide anonymity or unlinkability guarantees. - Replace application-level access control or policy enforcement.

## Acknowledgments

The author would like to thank the members of the IETF Security Area for their initial feedback and review of this generative framework.

## Author’s Address

Jian Sheng Wang
Independent Researcher
Email: jason@mscikdf.com