EterGam Vault

Security Architecture & Cryptographic Design

Security White Paper — Version 1.0

---

1. Executive Summary

EterGam Vault is an offline encrypted storage application designed around a fundamental security principle: protected data should remain under the user's control, and access to that data should depend on cryptographic material that EterGam does not possess.

Vault does not depend on cloud-based encryption, remote authentication, user accounts, or server-side key storage for access to protected content.

The application uses **Argon2id** for password-based key derivation and **AES-256-GCM** for authenticated encryption.

Rather than placing all protected files inside a single monolithic encrypted archive, EterGam Vault encrypts stored files independently. Each encrypted file receives its own randomly generated 128-bit salt and independently generated 96-bit AES-GCM nonce.

The Vault's structural, cryptographic, identity, and file-association metadata is maintained separately inside an encrypted **Vault Core**.

EterGam Vault does not persistently store the user's master passwords, conventional master-password hashes, or derived encryption keys.

Required cryptographic keys are reconstructed at runtime.

EterGam does not possess the user's master passwords and cannot remotely recover or decrypt a Vault if the required credentials are permanently lost.

---

2. Security Design Principles

EterGam Vault is built around the following principles.

Local cryptographic control

Encryption, decryption, password processing, and key derivation are performed locally on the user's device.

The cryptographic architecture does not require EterGam infrastructure to be available.

No stored password verifier

Vault does not store the user's plaintext master password.

It also does not maintain a conventional master-password hash against which authentication attempts are compared.

The supplied master password instead participates directly in cryptographic key derivation and authenticated decryption.

No persistent derived encryption keys

Derived encryption keys are not stored as persistent Vault credentials.

Required key material is reconstructed when necessary from the supplied master password and applicable cryptographic parameters.

Per-object cryptographic isolation

Protected files are encrypted independently.

Each file has its own random salt, Argon2id derivation context, derived encryption key, and AES-GCM nonce.

Authenticated encryption

AES-256-GCM provides confidentiality together with ciphertext authentication and integrity verification.

Fail-closed behavior

When required authentication, integrity, structural, or device-consistency checks fail, Vault is designed to reject continued access rather than silently ignore the inconsistency.

---

3. Production Cryptographic Profile

The current EterGam Vault cryptographic profile is:

Component	Production configuration
Password-based KDF	**Argon2id**
KDF memory cost	**65,536 KiB / 64 MiB**
KDF time cost	**3 iterations**
KDF parallelism	**1**
KDF output	**256 bits / 32 bytes**
Salt	**128 bits / 16 bytes, random**

Component	Production configuration
Encryption	**AES-256-GCM**
AES key size	**256 bits**
AES-GCM nonce	**96 bits / 12 bytes, random**
GCM authentication tag	**128 bits**
Cryptographic RNG	Platform `SecureRandom`
Vault state fingerprint	**SHA-256**
Device consistency value	SHA-256-based device-derived value

Cryptographic salts and AES-GCM nonces are generated using the platform's cryptographically secure SecureRandom implementation.

The KDF currently produces exactly 32 bytes of raw derived key material.

These parameters describe the current Vault format and implementation. Future Vault versions may revise cryptographic parameters where security requirements or platform capabilities justify doing so.

---

4. Vault Initialization

Vault initialization is deliberately designed as a high-consequence operation.

The user creates two independent master passwords corresponding to two logically separate Vault compartments.

Each master password must contain at least **16 characters**.

Strong, unique passphrases are recommended.

The two master passwords cannot be identical.

Because EterGam does not provide conventional password recovery, each master password must be entered **three times** during initialization.

The application also requires the user to explicitly acknowledge three security notices covering the permanent nature of the credential, responsibility for retaining access credentials, and consequences of credential loss.

Initialization proceeds only after the required confirmations have been completed.

The resulting Vault Core and cryptographic state are created locally on the device.

---

5. Master Password Permanence

EterGam Vault does not provide an in-place master-password replacement mechanism for an existing populated Vault.

This is an intentional architectural boundary.

A master password can effectively be replaced through a controlled Vault lifecycle procedure:

Export protected data

↓

Delete the existing Vault

↓

Initialize a new Vault with new master passwords

↓

Import the data into the newly initialized Vault

The imported data is then protected under newly generated cryptographic contexts associated with the new Vault.

For an empty Vault, the existing Vault can simply be deleted and initialized again with new credentials.

This distinction is important: the master password associated with an existing cryptographic Vault state is not silently rewritten or replaced in place.

---

6. Dual-Vault Architecture

A single EterGam Vault installation contains two logically separate Vault compartments protected by independent master-password contexts.

The supplied master password is used within the cryptographic authentication process to access the corresponding protected compartment. Incorrect credentials return a generic authentication failure without disclosing internal failure details.

Compartment metadata and statistics are logically isolated. Information belonging to one protected compartment is not intentionally aggregated into or exposed through the other compartment.

7. Password Processing

EterGam Vault preserves the exact character sequence supplied by the user as the cryptographic password input.

Master passwords are encoded directly as **UTF-8** before Argon2id key derivation.

Vault does not apply Unicode normalization, automatic case conversion, trimming, locale-based transformation, or other intentional character modification before derivation.

Conceptually:

Exact User Input

↓

UTF-8 Encoding

↓

Argon2id

↓

256-bit Derived Key

As a consequence, character sequences that may appear visually similar but have different Unicode representations are intentionally treated as different passwords.

This behavior is part of the Vault cryptographic format and ensures that the password supplied to Argon2id corresponds deterministically to the exact character sequence entered by the user.

---

8. Argon2id Key Derivation

EterGam Vault uses **Argon2id** as its password-based Key Derivation Function.

A master password is never directly used as an AES encryption key.

For each applicable cryptographic context, the master password is combined with a randomly generated **128-bit salt** and the configured Argon2id parameters.

The current production configuration is:

Memory: 65,536 KiB / 64 MiB
Iterations: 3
Parallelism: 1
Salt: 128 bits
Output: 256 bits

Conceptually:

Master Password + Random Salt

↓

Argon2id
`m = 65536 KiB, t = 3, p = 1`

↓

256-bit derived key

↓

AES-256-GCM

Argon2id is a memory-hard password KDF intended to increase the computational and memory cost of password-guessing attacks.

Derived keys are not stored as persistent replacements for the user's master password.

---

9. Independent Per-File Encryption

EterGam Vault does not combine protected files into a single encrypted ZIP-style archive. Each imported file is encrypted independently and stored as an opaque UUIDv4-based .bin object rather than under its original plaintext filename.

Each protected file uses an independently generated 128-bit random salt, a separate Argon2id derivation context, a derived 256-bit encryption key, a 96-bit random AES-GCM nonce, and a 128-bit GCM authentication tag.

Because files use independent salts and derivation contexts, compromise of one derived per-file key does not automatically reveal the derived encryption keys used for other protected files.

This isolation does not protect against compromise of the master password itself. Possession of the correct master password together with the required Vault data and protected cryptographic metadata fundamentally changes the attack model.

10. AES-256-GCM Authenticated Encryption

EterGam Vault uses **AES with a 256-bit key in Galois/Counter Mode**.

Each applicable encryption operation uses a randomly generated:

96-bit / 12-byte nonce

and produces a:

128-bit authentication tag

The nonce is generated using `SecureRandom`.

AES-GCM provides two core security properties.

Confidentiality

Ciphertext does not disclose its plaintext content without the required key.

Authentication and integrity

Unauthorized modification of authenticated ciphertext is detected during authenticated decryption with overwhelming probability under the assumed security of AES-GCM and correct nonce handling.

Nonce uniqueness is essential to AES-GCM security. Vault generates independent random nonces rather than intentionally reusing a fixed nonce.

---

11. The Vault Core

The Vault Core is an encrypted structural component containing protected metadata required to reconstruct, associate, and validate the logical Vault state.

The Core is independently protected using the established cryptographic primitives described in this document and maintains a cryptographic context separate from individual encrypted file objects.

The encrypted Vault Core does not contain a plaintext master password, conventional master-password hash, or persistently stored derived encryption key.

Structural inconsistencies or authenticated-integrity failures are treated as invalid Vault state and cause access to fail closed.

12. Split Cryptographic Storage Model

EterGam Vault separates encrypted payload objects from the protected metadata required to reconstruct and validate their cryptographic context.

An isolated encrypted file object does not itself contain the complete context used by the application to reconstruct protected content. The Vault Core is likewise not itself an encryption key.

Successful access therefore depends on the required user credential together with the corresponding protected Vault data and cryptographic context.

13. Vault Identity

During initialization, EterGam Vault creates an encrypted internal Vault Identity.

Vault Identity maintains protected provenance, state, and consistency metadata used by the application to validate the expected Vault environment and logical state.

Vault Identity information is maintained within the encrypted Vault context and is not intentionally exposed as ordinary plaintext metadata.

14. Vault State Integrity

EterGam Vault maintains an encrypted SHA-256-based fingerprint representing security-relevant elements of the expected logical state of each Vault compartment.

This is a cryptographic state digest, not a public-key digital signature.

Vault therefore applies two conceptually separate integrity layers: AES-256-GCM authentication protects individual encrypted objects, while SHA-256 state fingerprinting supports verification of the expected logical Vault state.

Unexpected inconsistencies are treated as potential integrity violations.

15. Device Consistency Verification

Vault maintains an encrypted, SHA-256-based device consistency value derived from device-specific application context.

The value is used to detect whether protected Vault data is being accessed in a device context inconsistent with the one expected by the Vault. A detected mismatch causes normal Vault access to be denied.

This mechanism is designed as a device-consistency control. It must not be interpreted as hardware-backed device attestation, cryptographic hardware binding, or a substitute for the confidentiality provided by the user's master password and Vault encryption.

16. Local Time Consistency

Vault maintains local temporal consistency information as part of its protected state and can reject or flag states that are inconsistent with its recorded lifecycle.

These checks rely on the device's local time environment and are intended as consistency controls, not as trusted external time attestation.

17. Authentication Failure and Cooldown

Authentication failures intentionally return a generic result without exposing detailed information about the internal condition that failed.

Vault applies application-level rate limiting and escalating cooldown controls to repeated authentication attempts through the normal application interface.

These controls are supplementary and are not considered a cryptographic defense against direct offline password guessing. The primary cryptographic resistance to such guessing is provided by the Argon2id derivation cost together with the strength of the user's master password.

18. Runtime Key Handling

Master passwords and derived encryption keys are not persistently stored as Vault credentials.

During normal operation, required key material is derived when needed and retained only for the operations or active session that require it.

Where Vault controls mutable sensitive buffers, the application performs **best-effort memory zeroization** by explicitly overwriting those buffers after use.

Vault therefore does not intentionally rely exclusively on managed-runtime garbage collection for disposal of application-controlled sensitive key material.

Android nevertheless operates in a managed runtime and complex operating-system environment.

Garbage collection, JIT/runtime implementation, internal copies, memory management, paging, and platform internals can prevent an application from guaranteeing physical erasure of every transient historical representation.

EterGam Vault therefore does not claim perfect forensic RAM erasure.

---

19. Session Locking

Vault limits the lifetime of authenticated application state and automatically locks on security-relevant application lifecycle transitions, including when the device screen is turned off or the application enters the background.

When a Vault session is terminated, application-controlled runtime key material is discarded and best-effort zeroization is applied where applicable.

Returning to protected content requires authentication and the required cryptographic derivation process again.

20. Internal File Access and Preview

Supported protected content can be previewed within the authenticated Vault context without intentionally creating ordinary plaintext temporary files in general storage.

Content that cannot be handled by the internal protected preview path requires explicit export before it can be opened externally.

Decrypted filenames and related metadata are exposed only as required within the authenticated Vault context and are not represented by the opaque physical filenames used for encrypted storage.

21. Export as a Plaintext Release Boundary

Export is treated as a higher-trust operation than viewing protected content inside Vault.

Export requires **master-password re-authentication**.

The application then performs the required derivation and authenticated decryption before allowing the plaintext file to be saved to a user-selected destination.

This creates an explicit boundary between:

protected internal access

and

intentional plaintext release.

Once a plaintext file has been exported outside EterGam Vault, Vault's encryption-at-rest protections no longer apply to that external copy.

Its subsequent security depends on the destination storage, Android environment, installed applications, backup mechanisms, and user handling.

---

22. Clipboard Isolation

EterGam Vault deliberately restricts clipboard interaction.

Master passwords must be manually entered.

Pasting a master password from the Android clipboard is not permitted.

Clipboard-based copying into or out of protected Vault workflows is blocked as an enforced application security policy and cannot be disabled by the user.

This reduces exposure through the general Android clipboard environment.

---

23. Screen Capture Protection

Screen-capture protection is enabled by default throughout EterGam Vault.

Users may explicitly allow screenshots through application settings if they choose to relax this protection.

The control reduces exposure through normal Android screenshot mechanisms.

It is not presented as protection against privileged malware, compromised operating systems, modified devices, external cameras, or other capture mechanisms outside the application's security boundary.

---

24. Individual File Deletion

Deleting a protected file requires explicit user confirmation identifying the file to be removed.

Vault implements **best-effort secure deletion** rather than intentionally relying solely on logical deletion where stronger operations are available.

This includes overwrite operations where permitted by the operating system and underlying storage.

Associated Vault metadata and logical state are updated to reflect removal of the object.

Modern flash storage may employ wear-leveling, controller remapping, filesystem journaling, garbage collection, and other mechanisms outside application control.

EterGam Vault therefore does not claim guaranteed physical destruction of every historical representation of a deleted file.

---

25. Entire Vault Deletion

Complete Vault deletion is intentionally designed as a multi-stage destructive operation requiring explicit user confirmation and authentication before protected Vault data is removed.

An optional Secure Erase mode performs additional best-effort overwrite and sanitization operations within the capabilities exposed by Android and the underlying storage.

Secure Erase is not presented as guaranteed forensic destruction on flash-based storage because physical media behavior may remain outside application control.

26. No Remote Password Recovery

EterGam Vault does not maintain recovery copies of user master passwords.

It does not maintain a server-side master key capable of bypassing user credentials.

It does not maintain a conventional password verifier from which EterGam can reconstruct the user's credential.

If the required credential is permanently lost while protected data remains inside the existing Vault, EterGam cannot remotely reconstruct or decrypt that content.

The supported path for changing credentials therefore requires access to the existing Vault data first, followed by export, Vault recreation, and re-import under newly created credentials.

---

27. Threat Model

EterGam Vault is primarily designed to protect **data at rest** and reduce unnecessary plaintext exposure during normal application operation.

Its security architecture addresses scenarios including:

- unauthorized inspection of locked Vault storage;
- acquisition of isolated encrypted UUIDv4 file blobs;
- direct inspection of protected physical storage filenames;
- unauthorized modification of AES-GCM authenticated ciphertext;
- unexpected modification of the expected logical Vault state;
- straightforward copying of Vault data to another Android device;
- accidental plaintext export without re-authentication;
- ordinary Android screenshot exposure under default settings;
- clipboard-based leakage through normal application workflows;
- persistent storage of master passwords or derived encryption keys by Vault itself.

These protections operate within a defined Android platform and runtime security boundary.

---

28. Security Assumptions

The EterGam Vault security model assumes:

- the user's master password remains confidential;
- each master password has sufficient strength to resist practical guessing;
- the Android operating environment has not been fully compromised by a privileged attacker;
- the cryptographic primitives and platform implementations behave according to their intended security properties;
- `SecureRandom` provides cryptographically suitable randomness;

- the user does not deliberately export plaintext into an insecure environment and subsequently expect Vault protections to remain attached to that copy.

Compromise of these assumptions can change the effective security of the system.

---

29. Explicit Security Non-Claims

EterGam Vault does not describe itself as unbreakable.

It does not claim to make a compromised operating system trustworthy.

It does not claim protection against an attacker with full privileged control of the device while protected content is in use.

It does not claim that a rooted, instrumented, maliciously modified, or otherwise compromised Android runtime cannot observe application execution or memory.

It does not claim perfect physical RAM erasure.

It does not claim guaranteed forensic erasure from modern flash storage.

It does not claim that screenshot restrictions defeat privileged capture mechanisms or external photography.

It does not claim that its device-consistency mechanism is equivalent to hardware-backed device attestation.

It does not claim that SHA-256 Vault-state fingerprinting replaces AES-GCM authentication.

It does not protect plaintext copies after deliberate export outside the Vault.

It cannot compensate for disclosure of the user's master password.

It cannot transform a weak and predictable password into a high-entropy secret merely by applying Argon2id.

EterGam does not possess a hidden recovery mechanism capable of bypassing the user's cryptographic credentials.

---

30. Master Password Security

The security of password-derived encryption depends substantially on the secrecy and strength of the user's master password.

The current Argon2id profile is:

```
m = 65536 KiB

t = 3

p = 1

output = 32 bytes
```

with an independently generated 16-byte salt.

This makes each password-guessing derivation deliberately more memory- and computation-intensive than simple password hashing.

However, no password KDF can make a trivially predictable password equivalent to a high-entropy secret.

Vault therefore requires at least 16 characters and recommends strong, unique passphrases.

Credentials should not be reused from websites, email accounts, cloud services, or unrelated applications.

---

31. Cryptographic Compromise Boundaries

Different compromise scenarios have materially different consequences.

Isolated encrypted file

Possession of a single `UUIDv4.bin` object does not itself provide the complete derivation context used by Vault.

Individual derived file key

Because individual files use independent random salts and independent Argon2id derivations, compromise of one file's derived AES key does not automatically reveal the derived keys protecting other files.

Vault Core

Possession of Vault Core data does not itself reveal the user's master password.

Master password plus required Vault data

If an attacker obtains the correct master password together with the required Vault data and cryptographic metadata, the attacker may perform the necessary derivations and attempt authenticated decryption of protected content.

Per-file cryptographic isolation must therefore not be interpreted as protection against compromise of the master password itself.

---

32. Implementation and Platform Dependencies

EterGam Vault operates within the Android application and operating-system security environment.

Its implementation consequently depends on platform components including:

- Android application isolation;
- runtime memory management;
- cryptographic provider behavior;
- filesystem behavior;
- platform `SecureRandom`;
- underlying device storage;
- operating-system lifecycle behavior.

Vault applies additional controls above these platform facilities but cannot create an independent trusted computing environment after the underlying operating system has been fully compromised.

This platform boundary is part of the Vault threat model.

---

33. Security Testing and Audit Status

EterGam Vault has undergone **internal adversarial and security-focused testing** during development.

Testing included attempts to identify weaknesses in authentication behavior, protected storage, application state, attack paths, and security controls.

Findings from this testing resulted in multiple changes intended to strengthen security behavior, stability, and failure handling.

This testing should not be confused with an independent third-party cryptographic audit.

EterGam Vault has not, as of this White Paper version, undergone an independent external security or cryptographic audit.

EterGam does not represent internal testing as independent certification, formal verification, or third-party security assurance.

---

34. Why the Cryptographic Design Is Public

EterGam Vault does not rely on secrecy of its cryptographic algorithms or parameters as a security mechanism.

The use of Argon2id, AES-256-GCM, SHA-256, 128-bit salts, 96-bit GCM nonces, 128-bit authentication tags, and the configured KDF parameters can be publicly documented without disclosing the user's master password.

Security should not depend on an attacker being unaware of which established cryptographic primitive an application uses.

It should depend on correct cryptographic construction, adequate randomness, credential secrecy, authenticated encryption, cryptographic separation, controlled key lifetime, and explicit trust boundaries.

Knowing how EterGam Vault protects data should not be equivalent to possessing the information required to decrypt that data.

---

35. Security Architecture Summary

EterGam Vault currently combines:

Argon2id
64 MiB memory cost, three iterations, parallelism one, 256-bit output;

128-bit random salts
generated using `SecureRandom`;

AES-256-GCM authenticated encryption;

96-bit random AES-GCM nonces
generated using `SecureRandom`;

128-bit GCM authentication tags;

independent per-file key derivation and encryption
instead of a monolithic encrypted archive;

encrypted Vault Core storage
for protected structural and cryptographic context;

dual logically isolated Vault compartments
selected cryptographically through the supplied master password;

SHA-256 Vault-state fingerprinting
for logical-state consistency verification;

SHA-256 device-consistency verification;

runtime-only key handling with best-effort memory zeroization;

automatic locking on security-relevant application lifecycle transitions;

re-authentication before plaintext export;

clipboard isolation controls;

default screen-capture protection;

and

best-effort secure deletion within Android and flash-storage limitations.

---

36. Final Statement

EterGam Vault is designed to provide local encrypted storage without requiring users to entrust their master passwords or plaintext Vault contents to EterGam infrastructure.

Its architecture separates encrypted file objects, protects structural metadata, independently derives cryptographic keys for protected objects, uses authenticated encryption, limits persistent secret material, and creates explicit boundaries around plaintext release.

EterGam does not use terms such as **“unbreakable”** or **“military-grade encryption”** as substitutes for a documented security model.

Security is not an absolute property.

The purpose of the EterGam Vault architecture is to make its trust boundaries explicit, minimize unnecessary exposure, use established cryptographic primitives with documented parameters, and ensure that possession of encrypted Vault data is not, by itself, equivalent to possession of its plaintext.

Appendix A — Cryptographic Reference

Property	Value
KDF	Argon2id
Memory cost	65,536 KiB / 64 MiB
Time cost	3

Property	Value
Parallelism	1
KDF output	256 bits / 32 bytes
Salt	128 bits / 16 bytes
Salt source	`SecureRandom`
Encryption	AES-256-GCM
AES key	256 bits
GCM nonce	96 bits / 12 bytes
Nonce source	`SecureRandom`
Authentication tag	128 bits
Vault state fingerprint	SHA-256
Device consistency value	SHA-256-based device-derived value
Physical encrypted file identifier	UUIDv4-based `.bin` object
Password recovery	No remote recovery
Persistent password storage	None
Persistent derived-key storage	None

Appendix B — Security Assurance Status

Internal adversarial testing: Performed
Security-driven implementation changes: Performed throughout development
Independent external security audit: Not performed
Formal cryptographic verification: Not claimed
Hardware-backed device attestation: Not claimed
Guaranteed forensic RAM erasure: Not claimed
Guaranteed flash-storage secure erase: Not claimed

EterGam Vault
Security Architecture & Cryptographic Design
Security White Paper — Version 1.0