

The Ouroboros Hardened Framework: Argon2id + XChaCha20-Poly1305 for Offline-Resistant Terminal Authentication

Kevin Thomas

School of Computing and Information
University of Pittsburgh, Pittsburgh, PA, USA
ket189@pitt.edu

Abstract — We present a hardened cryptographic framework for terminal-entered passphrases under an offline attack model. This framework is based on the original Ouroboros design lineage while focusing strictly on the modern hardened construction: Argon2id key derivation, per-ciphertext random salt, XChaCha20-Poly1305 authenticated encryption, strict 12-word lowercase policy, and reproducible JSON artifact generation/consumption. We provide explicit notation, equation-level construction, quantitative brute-force cost analysis, and an optimistic Grover-style interpretation. The paper additionally maps formulas to implementation parameters used in this repository and demonstrates consistency across generator, artifact schema, and runtime decryption path.

Index Terms — Argon2id, XChaCha20-Poly1305, memory-hard KDF, offline guessing resistance, password entropy, Grover scaling, authenticated encryption.

I. Introduction

The hardened Ouroboros framework targets a practical threat boundary: an attacker who captures encrypted artifacts and performs unrestricted offline verification attempts. In this regime, security is determined by two coupled quantities:

- effective passphrase entropy,
- per-guess computational and memory cost.

The design objective is to maximize expected attacker work without introducing ambiguous runtime behavior. The implementation therefore enforces deterministic validation boundaries (passphrase policy, artifact schema) while preserving cryptographic randomness where required (salt and nonce).

I.I. Contributions

This paper provides the following concrete contributions:

- A fully specified hardened construction using Argon2id and XChaCha20-Poly1305.
- A repository-aligned artifact contract (`demo_artifact.json`) with fixed field semantics.
- A mathematical derivation of expected classical crack time and optimistic Grover-style scaling.
- A parameterized interpretation framework that operators can recalculate for hardware-specific calibration.

II. Related Work

Password-hardening systems derive security from both secret entropy and asymmetric verification cost. Historical approaches include bcrypt and scrypt, with Argon2id now commonly adopted where memory hardness is required. Authenticated encryption consolidates confidentiality and integrity into a single primitive and avoids ad hoc MAC composition mistakes.

The hardened Ouroboros framework is implementation-centric rather than proof-system-centric: it prioritizes explicit parameter surfaces and operational reproducibility over abstract interface minimalism.

III. Formal Model

III.I. Notation

Let:

- P be the user passphrase (policy-constrained),
- $S \in \{0, 1\}^{128}$ be a salt,
- $N \in \{0, 1\}^{192}$ be a nonce,
- $M \in \{0, 1\}^{384}$ be the 48-byte payload,
- (m, t, p) be Argon2id parameters,
- C be ciphertext plus authentication tag.

The policy accepts only passphrases composed of exactly 12 lowercase ASCII words separated by single spaces under input normalization used by the implementation.

III.II. Hardened Construction

Key derivation:

$$K_h = \text{Argon2id}(P, S, m, t, p, 32)$$

Authenticated encryption:

$$C = \text{XChaCha20-Poly1305.Encrypt}(K_h, N, , M)$$

Authenticated decryption:

$$M = \text{XChaCha20-Poly1305.Decrypt}(K_h, N, , C)$$

where decrypt returns failure on authentication mismatch.

III.III. Repository Default Parameters

The default desktop profile used for the hardened demo artifact is:

- $m = 1048576$ KiB,
- $t = 3$,
- $p = 1$.

These values are intentionally expensive and should be recalibrated for deployment hardware if latency budgets differ.

IV. Artifact Contract

The runtime demo consumes a JSON blob with the following fields:

```
{
  "format": "ouroboros-hardened-demo-v1",
  "memory_kib": 1048576,
  "iterations": 3,
  "parallelism": 1,
  "salt_hex": "...32 hex chars...",
  "nonce_hex": "...48 hex chars...",
  "ciphertext_and_tag_hex": "...128 hex chars..."
}
```

Field lengths are strict:

- salt_hex is 16 bytes,
- nonce_hex is 24 bytes,
- ciphertext_and_tag_hex is 64 bytes,

with parse-time rejection on malformed or wrong-size hex.

IV.I. Deterministic Regeneration Check

Given fixed (P, S, N, m, t, p) , encryption output is deterministic. Therefore artifact correctness can be verified by regenerating and byte-comparing ciphertext_and_tag_hex.

This property is operationally useful:

- documentation examples remain auditable,
- CI checks can validate example artifacts,
- accidental config drift is detectable.

V. Payload and Dispatch Semantics

The plaintext payload is fixed at 48 bytes.

Region	Meaning
Byte 0	LED state
Bytes 1..7	UART text bytes
Remaining bytes	Fixed dispatch/MAC region

Fig. 1: 48-byte hardened payload layout.

By default, generation appends CRLF to the user text before packing into bytes 1..7; therefore application text length is constrained by:

$$|\text{text}| + 2 \leq 7$$

under default CRLF mode.

VI. Threat Model and Security Objective

VI.I. Adversary Model

Assume adversary access to:

- full artifact JSON content,
- complete source and binary behavior,
- unlimited offline trial capability.

Assume no side-channel leakage from trusted runtime outside modeled interfaces. Assume no active firmware patching, instruction-level execution control, or injected control-flow faults in the trusted runtime.

VI.II. Security Objective

Delay successful recovery by maximizing expected offline cost:

$$E[T] = \frac{E[G]}{r}$$

with throughput reduced by memory-hard KDF cost and guesses driven by entropy.

VII. Classical Cost Derivation

Let H be effective entropy in bits for the passphrase distribution. Assuming uniform random ordering over 2^H candidates:

$$E[G] = \frac{1 + 2^H}{2} \approx 2^{H-1}$$

where G is guess index of the first success.

Let r be guesses/second. Then:

$$E[T_c] = \frac{E[G]}{r} \approx \frac{2^{H-1}}{r}$$

Converting seconds to years ($Y = 31557600$):

$$T_c = \frac{2^{H-1}}{r * Y}$$

This is the baseline formula used throughout the quantitative tables.

VII.I. Entropy Sensitivity

A one-bit entropy increase doubles expected crack time:

$$T_{c(H+1)} = 2T_{c(H)}$$

A ten-bit increase multiplies by $2^{10} = 1024$.

Thus policy quality is exponentially more important than micro-optimizing any single arithmetic constant in the estimator.

VIII. Optimistic Grover-Style Interpretation

A common coarse model for quantum search treats effective search exponent as halved. Under this optimistic attacker model:

$$E[G_q] \approx 2^{\frac{H}{2}-1}$$

With oracle rate r_q evaluations/second:

$$T_q = \frac{2^{\frac{H}{2}-1}}{r_q * Y}$$

This model is intentionally favorable to the attacker and does not account for engineering constraints of large-scale fault-tolerant quantum hardware running real password oracles.

VIII.I. Ratio to Classical Model

The ratio is:

$$\frac{T_c}{T_q} = 2^{\frac{H}{2}} \frac{r_q}{r}$$

When $r_q = r$, the gap scales as $2^{\frac{H}{2}}$.

IX. Worked Scenario and Expanded Math

Set:

- $H = 155.1$,
- $r = 0.1$ guesses/s,
- $r_q = 0.1$ oracle eval/s,
- $Y = 31557600$ s/year.

Classical expected time:

$$T_c = \frac{2^{154.1}}{0.1 * 31557600} \approx 7.76 \times 10^{39} \text{ years}$$

Optimistic quantum expected time:

$$T_q = \frac{2^{76.55}}{0.1 * 31557600} \approx 3.51 \times 10^{16} \text{ years}$$

Relative to universe age $U = 1.38 \times 10^{10}$ years:

$$R_c = \frac{T_c}{U} \approx 5.6 \times 10^{29}$$

$$R_q = \frac{T_q}{U} \approx 2.5 \times 10^6$$

Metric	Classical	Optimistic Grover
Expected years	7.76×10^{39}	3.51×10^{16}
Universe-age ratio	5.6×10^{29}	2.5×10^6

Fig. 2: Reference attack-cost magnitudes for hardened policy target.

This worked row is a paper-standard reference scenario, not a measured desktop benchmark. Actual deployment claims must use measured passphrase entropy quality and deployment-specific per-guess throughput.

IX.I. Why the Numbers Stay Large

Even with exponent-halving assumptions, the combined effect of high entropy and slow per-guess verification preserves extreme expected timescales.

In log-domain terms:

$$\log_2(T_c) = H - 1 - \log_2(rY)$$

$$\log_2(T_q) = \frac{H}{2} - 1 - \log_2(r_q Y)$$

So increasing H continues to dominate over moderate throughput gains.

X. Implementation Compliance Mapping

The repository runtime and tooling implement the same hardened boundary:

- Generator: `scripts/dec.py` writes hardened JSON artifacts.
- Runtime demo: `src/bin/demo.rs` loads and validates artifact fields.

- Engine: hardened decrypt path takes (m, t, p, S, N, C) explicitly.

A successful demo decrypt requires both:

- policy-compliant passphrase syntax,
- exact passphrase match with artifact-generation passphrase.

Therefore, policy validity is necessary but not sufficient.

X.I. Failure Modes

Authentication failure occurs when any of the following hold:

- wrong passphrase for current artifact,
- artifact corruption or malformed hex,
- KDF parameter drift relative to generated ciphertext,
- tag verification failure.

These are desirable explicit failures, not undefined behavior.

XI. Limitations and Scope

- This framework is not a NIST PQC KEM/signature system.
- Entropy estimates depend on operational passphrase generation quality.
- Cost formulas are expectation models; they are not formal proofs of irrecoverability.
- Optimistic Grover estimates are directional and may overstate practical attacker capability in near-term systems.

XII. Conclusion

The hardened Ouroboros framework provides an implementation-aligned, mathematically explicit path for passphrase-authenticated decryption under offline capture assumptions. Its security posture follows directly from measurable factors: entropy and per-guess cost. By coupling strict policy enforcement, memory-hard KDF calibration, and AEAD verification into a single deterministic artifact workflow, the framework avoids ambiguous success criteria and supports reproducible security analysis tied to real deployment parameters.

XIII. References

- [1] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: The memory-hard function for password hashing and other applications,” in *IEEE EuroS&P Workshops*, 2016.
- [2] IETF CFRG, “draft-irtf-cfrg-argon2,” parameterization and interoperability guidance, latest revision.
- [3] F. Denis, “XChaCha: eXtended-nonce ChaCha and AEAD constructions,” IRTF CFRG discussions and implementations.
- [4] D. J. Bernstein, “ChaCha, a variant of Salsa20,” Workshop Record of SASC, 2008.
- [5] N. Provos and D. Mazières, “A future-adaptable password scheme,” *USENIX ATC*, 1999.
- [6] C. Percival and S. Josefsson, “The scrypt Password-Based Key Derivation Function,” RFC 7914, 2016.
- [7] L. Grover, “A fast quantum mechanical algorithm for database search,” *STOC*, 1996.
- [8] A. Y. Kiayias et al., “Post-quantum considerations for password-authenticated systems,” survey literature on practical constraints in oracle realization.
- [9] A. Reinhold, “Diceware Passphrase Home Page,” wordlist-based entropy estimation references.
- [10] RustCrypto, “argon2” and “chacha20poly1305” crate documentation, implementation behavior and API constraints.