Qatsi: Stateless Secret Generation via Hierarchical Memory-Hard Key Derivation

René Coignard*
Independent Researcher
Dessau-Roßlau, Germany

Anton Rygin†
Technische Universität Dresden
Dresden, Germany

https://github.com/coignard/qatsi

October 2025

IACR Cryptology ePrint Archive
Preliminary version

Abstract

We present Qatsi, a hierarchical key derivation scheme using Argon2id that generates reproducible cryptographic secrets without persistent storage. The system eliminates vault-based attack surfaces by deriving all secrets deterministically from a single high-entropy master secret and contextual layers. Outputs achieve 103–312 bits of entropy through memory-hard derivation (64–128 MiB, 16–32 iterations) and provably uniform rejection sampling over 7776-word mnemonics or 90-character passwords. We formalize the hierarchical construction, prove output uniformity, and quantify GPU attack costs: 2.4×10^{16} years for 80-bit master secrets on single-GPU adversaries under Paranoid parameters (128 MiB memory). The implementation in Rust provides automatic memory zeroization, compile-time wordlist integrity verification, and comprehensive test coverage. Reference benchmarks on Apple M1 Pro (2021) demonstrate practical usability with 544 ms Standard mode and 2273 ms Paranoid mode single-layer derivations. Qatsi targets air-gapped systems and master credential generation where stateless reproducibility outweighs rotation flexibility.

Keywords: Key derivation, Argon2id, memory-hard functions, hierarchical derivation, stateless password management.

1 Introduction

Password management architectures face a fundamental trade-off between security and usability. Centralized encrypted vaults (KeePassXC, Bitwarden) enable arbitrary credential storage and cross-device synchronization but concentrate risk in a single encrypted database. Vault compromise, cloud interception, or file exfiltration exposes all credentials. Deterministic generation eliminates persistent attack surfaces but sacrifices per-credential rotation and key isolation.

Argon2id, winner of the 2015 Password Hashing Competition [8] and standardized in RFC 9106 [1], remains underutilized in consumer password management despite its proven resistance to GPU/ASIC parallelization through memory-hardness. Mainstream alternatives

*Email: contact@renecoignard.com. ORCID: 0009-0004-0484-3336
†Email: anton.rygin@tu-dresden.de. ORCID: 0009-0009-3640-9872

like PBKDF2 (RFC 2898 [2]) lack memory-hard properties and remain vulnerable to hardware acceleration.

Existing deterministic schemes include Diceware [6] (physical dice, manual), BIP39 [7] (PBKDF2-based, cryptocurrency wallets), and LessPass (stateless web passwords). None provide hierarchical context-aware derivation with memory-hard KDF and provably uniform output sampling.

Problem. Generate cryptographically strong, reproducible secrets for master credentials (password manager master passwords, full-disk encryption passphrases, PGP/SSH key passphrases) without storing any persistent state, while resisting GPU attacks through memory-hardness.

Contributions.

1. Hierarchical Argon2id chaining enabling context-specific derivation from a single master secret without storage overhead.
2. Formal proof of output uniformity via rejection sampling for both word-based (7776-word) and character-based (90-char) alphabets.
3. Quantitative GPU attack cost analysis with realistic hardware parameters showing $\approx 10^{16}$-year resistance for 80-bit entropy under 128 MiB memory constraints.
4. Production implementation in Rust with automatic zeroization, Unicode NFC normalization, and comprehensive testing including regression vectors verified on Apple M1 Pro hardware.

Non-goals. Qatsi does not replace traditional password managers for everyday website credentials with varying policies, existing passwords, or frequent rotation requirements. It targets high-stakes reproducible secrets in constrained environments (air-gapped systems, master credentials).

2 Related Work

2.1 Memory-Hard Key Derivation

Argon2 [1] achieves memory-hardness by filling memory with pseudorandom data dependent on password and salt, forcing adversaries to allocate comparable memory per guess. Argon2id combines data-independent (Argon2i) and data-dependent (Argon2d) phases, resisting both side-channel attacks and GPU optimization. Standard configurations use 64 MiB memory and 16 iterations; cryptographic libraries recommend ≥ 2048 iterations for PBKDF2 to achieve comparable security [10].

scrypt [9] pioneered memory-hard password hashing but achieved less widespread standardization than Argon2. PBKDF2 [2], despite decades of deployment, remains vulnerable to GPU farms processing $> 10^9$ hashes/second on consumer hardware [11].

Distinction from prior work. While Argon2 provides single-level key derivation, Qatsi introduces hierarchical chaining with context-aware layers, enabling stateless multi-domain secret generation from a single root secret without storage overhead.

2.2 Deterministic Password Schemes

Diceware [6] generates passphrases by mapping physical dice rolls to wordlists. The EFF Large Wordlist [5] contains exactly $6^5 = 7776$ words, yielding $\log_2(7776) \approx 12.925$ bits per word. Diceware requires manual dice rolling; Qatsi automates generation via ChaCha20 keystream with provably uniform rejection sampling.

BIP39 [7] encodes 128–256 bit seeds as 12–24 word mnemonics using a 2048-word list and PBKDF2-HMAC-SHA512 (2048 iterations). The scheme appends 4–8 checksum bits to the

original entropy, requiring 12–24 words total for 128–256 bits of entropy. BIP39 optimizes for cryptocurrency wallet recovery; Qatsi generalizes to arbitrary hierarchical secrets with stronger memory-hard KDF.

LessPass implements stateless password generation via PBKDF2 but lacks hierarchical derivation and memory-hard protection.

Our contribution. Qatsi is the first system combining hierarchical context-aware derivation, memory-hard KDF (Argon2id), and provably uniform output sampling for general-purpose stateless secret management.

2.3 Unbiased Sampling

Lemire [12] formalized fast unbiased random integer generation via rejection sampling. Modulo reduction introduces bias when range size does not divide sample space size. For uniform sampling from $[0, n)$ using b-bit integers: sample $r \in [0, 2^b)$; accept if $r < 2^b - (2^b \bmod n)$; otherwise reject. Expected samples per output: $(1 - \frac{2^b \bmod n}{2^b})^{-1}$.

Our implementation (Algorithms 1, 2) applies this technique to word and character selection with formal uniformity proof (Theorem 2).

3 Hierarchical Derivation

3.1 Definitions

Definition 1 (Hierarchical Key Derivation Function)**.** *Let $\mathcal{M} = \{0,1\}^*$ be the master secret space and $\mathcal{L} = (\{0,1\}^*)^n$ be an ordered sequence of n context layers. Define the hierarchical key derivation function $\mathcal{H} : \mathcal{M} \times \mathcal{L} \to \{0,1\}^{256}$ as:*

$$\mathcal{H}(M, (L_1, \ldots, L_n)) = K_n \tag{1}$$

where $K_0 = M$ and for $i \in [1, n]$:

$$K_i = \textsc{Argon2id}(K_{i-1}, \textsc{Salt}(L_i), m, t, p, \ell) \tag{2}$$

Parameters: m (memory cost in KiB), t (iterations), p (parallelism), $\ell = 32$ bytes output length.

Definition 2 (Salt Preprocessing)**.** *For input $L \in \{0,1\}^*$:*

$$\textsc{Salt}(L) = \begin{cases} L & \text{if } |L| \geq 16 \text{ bytes} \\ \textsc{Blake2b-512}(L) & \text{if } |L| < 16 \text{ bytes} \end{cases} \tag{3}$$

Argon2 requires salts ≥ 8 bytes [1]; we enforce 16 bytes minimum for additional security margin. Inputs shorter than 16 bytes are expanded to 64 bytes via BLAKE2b-512, ensuring sufficient entropy and preventing short-salt attacks while maintaining determinism.

Definition 3 (Input Normalization)**.** *All text inputs undergo:*

$$\textsc{Normalize}(s) = NFC(\textsc{Trim}(s)) \tag{4}$$

where NFC applies Unicode Normalization Form C and Trim removes leading/trailing whitespace.

3.2 Configuration Profiles

Profile	m (MiB)	t	p	M1 Pro Time (ms)
Standard	64	16	6	544
Paranoid	128	32	6	2273

Table 1: Argon2id parameter configurations with measured single-layer derivation times on Apple M1 Pro (2021). High-end GPUs achieve estimated 0.25–1.0 s (Standard) and 0.6–1.25 s (Paranoid).

Output length fixed at $\ell = 32$ bytes for compatibility with ChaCha20 [4].

3.3 Output Generation

Algorithm 1 Mnemonic generation via rejection sampling

Require: Derived key $K \in \{0,1\}^{256}$, word count w, wordlist W of size $n = 7776$
Ensure: Mnemonic phrase M of w words

cipher $\leftarrow$ ChaCha20(K, nonce $= 0$)
$T \leftarrow \lfloor 2^{16}/n \rfloor \times n$ ▷ $T = 62208$ for $n = 7776$
words $\leftarrow$ []
while |words| $< w$ **do**
    $r \leftarrow$ cipher.next_u16() ▷ Sample 16-bit value
    if $r < T$ **then**
        words.append($W[r \bmod n]$)
    end if
end while
return words.join(" − ")

Algorithm 2 Password generation via rejection sampling

Require: Derived key $K \in \{0,1\}^{256}$, length ℓ, alphabet A of size $|A| = 90$
Ensure: Password P of length ℓ

cipher $\leftarrow$ ChaCha20(K, nonce $= 0$)
$T \leftarrow 256 - (256 \bmod |A|)$ ▷ $T = 180$ for $|A| = 90$
chars $\leftarrow$ []
while |chars| $< \ell$ **do**
    $b \leftarrow$ cipher.next_u8() ▷ Sample 8-bit value
    if $b < T$ **then**
        chars.append($A[b \bmod |A|]$)
    end if
end while
return chars.join("")

The alphabet A contains 90 characters: A–Z (26), a–z (26), 0–9 (10), and 28 special characters: `!@#$%^&*()_+-=[]{};:,.<>?/ |`

3.4 Uniformity Analysis

Lemma 1 (Rejection Sampling Uniformity)**.** *Let $n \in \mathbb{N}$ and $b \in \mathbb{N}$ such that $n \leq 2^b$. Define $T = \lfloor 2^b/n \rfloor \times n$. Sampling r uniformly from $[0, 2^b)$ and accepting if $r < T$ with output $r \bmod n$*

produces uniform distribution over $[0, n)$.

Proof. For accepted samples $r \in [0, T)$, define $k = \lfloor 2^b/n \rfloor$. Then $T = kn$ and $r \bmod n$ maps exactly k values to each element in $[0, n)$. Since r is uniform over $[0, 2^b)$ and we condition on $r < T$, the conditional distribution of r given acceptance is uniform over $[0, T)$. Thus each output value has probability $k/(kn) = 1/n$. □

Theorem 2 (Output Entropy)**.** *Let $\mathcal{W}$ be a wordlist of size $n = 7776$ and $\mathcal{A}$ be an alphabet of size $|A| = 90$. For uniform random key $K \xleftarrow{\$} \{0,1\}^{256}$ and output generation via Algorithms 1, 2:*

Mnemonic output of w words has entropy:

$$H_{mnemonic} = w \log_2 n \tag{5}$$

Password output of length ℓ has entropy:

$$H_{password} = \ell \log_2 |A| \tag{6}$$

Proof. By Lemma 1, each word/character is uniform and independent. For w independent uniform choices from n elements: $H = \log_2 n^w = w \log_2 n$. Similarly for passwords with ℓ choices from $|A|$ elements. □

For $n = 7776$ and $|A| = 90$:

Mode	Count	Entropy (bits)
Mnemonic (Standard)	8 words	103.4
Mnemonic (Paranoid)	24 words	310.2
Password (Standard)	20 chars	129.8
Password (Paranoid)	48 chars	311.6

Table 2: Output entropy for standard configurations

3.5 Rejection Rate

Proposition 3 (Expected Samples)**.** *For n-element output with b-bit sampling and rejection threshold $T = \lfloor 2^b/n \rfloor \times n$, the expected number of samples per accepted output is:*

$$E[samples] = \frac{2^b}{\lfloor 2^b/n \rfloor \times n} \tag{7}$$

Proof. The acceptance probability is $p = T/2^b = (\lfloor 2^b/n \rfloor \times n)/2^b$. Expected samples follow geometric distribution: $E = 1/p = 2^b/T$. □

For mnemonic ($b = 16$, $n = 7776$): $T = 62208$, rejection rate $= 3328/65536 \approx 5.08\%$, $E[\text{samples}] = 65536/62208 \approx 1.053$.

For password ($b = 8$, $n = 90$): $T = 180$, rejection rate $= 76/256 \approx 29.69\%$, $E[\text{samples}] = 256/180 \approx 1.422$.

4 Security Analysis

4.1 Threat Model

Adversarial capabilities. We consider an adversary $\mathcal{A}$ with:

1. **Offline brute-force:** $\mathcal{A}$ can compute Argon2id hashes on custom hardware (GPUs, ASICs) with full parameter knowledge.

2. **Memory access:** $\mathcal{A}$ observes all derived outputs but not intermediate keys.

3. **Layer knowledge:** $\mathcal{A}$ knows all layer strings (Kerckhoffs's principle).

4. **Computational bound:** $\mathcal{A}$ has access to p parallel processors with memory M GB each.

Out of scope: Keyloggers during input, side-channel timing attacks, quantum adversaries (Grover's algorithm [13] provides only $\sqrt{2}$ speedup), physical coercion, social engineering.

4.2 Attack Cost Estimation

Theorem 4 (Brute-Force Resistance)**.** *For master secret entropy e bits and Argon2id configuration (m, t, p) with processing time τ seconds per hash, average-case brute-force time to recover secret:*

$$T_{attack} = 2^{e-1} \cdot \tau \quad \textit{(half the search space)} \tag{8}$$

For $e = 80$ bits (recommended minimum):

$$T_{\text{attack}} = 2^{79} \cdot \tau \approx 6.04 \times 10^{23} \cdot \tau \text{ seconds} \tag{9}$$

Conservative GPU estimate. High-end GPUs (NVIDIA A100, H100) optimized for Argon2 achieve estimated $\tau \approx 1.25$ s for Paranoid configuration (128 MiB, $t = 32$, $p = 6$):[1]

$$T_{\text{single}} = 2^{79} \times 1.25 \text{ s} \approx 7.56 \times 10^{23} \text{ s} \tag{10}$$

$$\approx 2.39 \times 10^{16} \text{ years} \tag{11}$$

For 500-GPU cluster with 10% coordination overhead:

$$T_{500} = \frac{2.39 \times 10^{16}}{500} \times 1.1 \approx 5.27 \times 10^{13} \text{ years} \tag{12}$$

Memory-bound parallelism. The critical security property is not computational speed but memory requirements. A GPU with 40 GB RAM can maintain at most $\lfloor 40000/128 \rfloor = 312$ parallel Argon2 instances. Even with 500 such GPUs (156,000 parallel attempts):

$$T_{\text{parallel}} = \frac{2^{79}}{156000} \times 1.25 \text{ s} \tag{13}$$

$$\approx 4.84 \times 10^{18} \text{ s} \approx 1.53 \times 10^{11} \text{ years} \tag{14}$$

This remains computationally infeasible (153 billion years) even with massive parallelization, validating memory-hardness as the primary defense mechanism.

4.3 Hardware Attack Benchmarks

To validate theoretical attack costs, we measured actual Argon2id performance on representative hardware:

Platform	Standard (ms)	Paranoid (ms)
Apple M1 Pro (2021)†	544	2273
NVIDIA A100 (est.)‡	250	1000
NVIDIA H100 (est.)‡	150	625

Table 3: Single-layer Argon2id derivation times. †Measured (median of 5 runs). ‡Estimated based on memory bandwidth and published Argon2 benchmarks [11].

[1] Reference implementation on Apple M1 Pro (2021) measures $\tau = 2.273$ s, but dedicated GPU attacks with optimized memory controllers are expected to be faster. The primary defense is memory-hardness: the 128 MiB memory requirement fundamentally limits parallelization regardless of computational throughput.

M1 Pro measurements ($\tau = 2.273$ s for Paranoid) are 1.8× slower than our conservative GPU estimate ($\tau = 1.25$ s), confirming the paper's attack cost estimates are realistic lower bounds. The memory-hard property ensures even specialized hardware cannot achieve orders-of-magnitude speedups without proportional memory increases.

4.4 Master Secret Entropy Requirements

Source	Entropy (bits)	Status
16-byte `urandom`	128	Secure
11 EFF words	≈ 142	Secure
80-bit CSPRNG	80	Minimum*
Human password	< 40	Insufficient

Table 4: Master secret entropy guidelines. *80-bit minimum provides $\approx 2.4 \times 10^{16}$ year resistance against single-GPU attacks under Paranoid configuration.

4.5 Cryptographic Primitives

All components use standard, well-analyzed primitives:

- Argon2id (RFC 9106 [1]): memory-hard KDF, 256-bit output
- BLAKE2b-512 (RFC 7693 [3]): salt preprocessing for short inputs
- ChaCha20 (RFC 8439 [4]): stream cipher for keystream generation
- EFF Large Wordlist [5]: 7776 words, SHA-256 verified

4.6 Limitations

No key isolation. Master secret compromise exposes all derived secrets. Unlike vault-based managers where individual credential leaks remain isolated, deterministic derivation creates dependency on a single root secret. This is inherent to stateless deterministic systems.

No credential rotation isolation. Changing a single derived secret requires either (1) modifying layer inputs (user must remember modification) or (2) changing master secret (affects all derivations).

Layer enumeration. Predictable layer patterns (e.g., `service/YYYY`) enable targeted enumeration. Mitigation: use high-entropy layer strings documented separately from master secret.

5 Implementation

5.1 Memory Safety

Implementation in Rust uses `Zeroizing<T>` wrapper (crate `zeroize`) for automatic secure erasure:

1: **function** DeriveHierarchical($M, \{L_i\}$, config)
2: $K \leftarrow$ Zeroizing([0; 32]) ▷ Auto-zeroed on drop
3: Argon2id(M, Salt(L_1), config, K)
4: **for** $i \in [2, n]$ **do**
5: $K' \leftarrow$ Zeroizing([0; 32])
6: Argon2id(K, Salt(L_i), config, K')

7: $K \leftarrow K'$ ▷ Old K auto-zeroed
8: **end for**
9: **return** K
10: **end function**

`Zeroizing` provides panic-safe volatile writes preventing compiler optimization removal.

5.2 Wordlist Integrity

Compile-time verification prevents supply-chain attacks:

Require: Embedded wordlist W, expected SHA-256 h_{exp}
1: $h \leftarrow \text{SHA-256}(W)$
2: **assert** $h = h_{\text{exp}}$ at build time
3: ▷ Known indices: $W[0]$ = "abacus", $W[469]$ = "balance", $W[3695]$ = "life", $W[7775]$ = "zoom"

Expected hash: `addd3553...96b903e` (64 hex digits). Build script verifies integrity before compilation succeeds.

5.3 Test Coverage

Comprehensive test suite validates:

- **Determinism:** Identical inputs produce identical outputs (100 runs, zero variance)
- **Regression:** Known input/output pairs for Standard/Paranoid modes
- **Unicode:** NFC/NFD normalization equivalence, multi-byte character preservation
- **Wordlist:** Exactly 7776 words, no duplicates, SHA-256 integrity
- **Sampling:** Rejection thresholds (62208 for words, 180 for chars)
- **Layer independence:** Different layers produce uncorrelated keys

Regression test vectors:

Input	Config	Output (first 40 chars)
M = "life" L = [out, of, balance]	Standard (8 words)	eagle-huskiness-septum-defection...
M = "life" L = [out, of, balance]	Paranoid (24 words)	vigorous-purebred-exclusion-defa...
M = "life" L = [out, of, balance]	Standard (20 chars)	6n=rX.k:Qs+)6e5oa-Z:

Table 5: Regression test vectors.

6 Experimental Evaluation

6.1 Performance

Platform: Apple M1 Pro (2021), 16 GB RAM, Rust 1.90 release build. Median of 5 runs:

Operation	Time (ms)	Memory (MB)
Standard (64 MiB, $t = 16$, $p = 6$)		
Single layer	544	64
3 layers	1613	64
Paranoid (128 MiB, $t = 32$, $p = 6$)		
Single layer	2273	128
3 layers	6697	128
Output generation	< 1	< 1

Table 6: Performance benchmarks. Output generation measured over 1000 iterations: mnemonic generation 2 µs, password generation 3 µs.

Time complexity: $O(n)$ in layer count. Space complexity: $O(1)$ in output size, $O(m)$ in KDF memory. Paranoid mode $\approx 4\times$ Standard due to doubled memory and iterations.

6.2 Determinism Verification

100 independent runs with identical inputs produce byte-identical outputs (SHA-256 hash variance: 0). Chi-squared test on 10,000 generated mnemonics shows uniform word distribution ($p > 0.95$).

7 Discussion

7.1 Use Cases

Appropriate: Password manager master passwords (KeePassXC, Bitwarden), full-disk encryption passphrases (LUKS, BitLocker, FileVault), PGP/SSH key passphrases, cryptocurrency wallet seeds, critical service credentials on air-gapped systems.

Inappropriate: General website passwords (varying policies, frequent rotation), existing credentials (API keys, legacy passwords), multi-device sync with conflict resolution, scenarios requiring key isolation after breach.

7.2 Comparison

Property	KeePassXC	BIP39	Diceware	Qatsi
Storage	Vault	None	None	None
KDF	Argon2/AES	PBKDF2	N/A	Argon2id
Hierarchical	No	No	No	Yes
Memory-hard	Yes	No	N/A	Yes
Rotation	Easy	Hard	Hard	Hard
Key isolation	Yes	No	No	No

Table 7: Comparison with existing password management approaches

7.3 Operational Security

Master secret generation:

```
# 96-bit random (12 bytes)
od -An -tx1 -N12 /dev/urandom | tr -d ' '

# 11 EFF words (~142 bits)
shuf -n 11 eff_large_wordlist.txt | tr '\n' '-'
```

Layer design: Use high-entropy, non-obvious strings. Avoid sequential numbers or dictionary words. Document layers separately from master secret in offline storage.

Backup: Maintain physical copy of master secret in secure location (safe, bank deposit box). Loss is unrecoverable. Consider Shamir's Secret Sharing for distributed backup.

8 Conclusion

Qatsi demonstrates hierarchical deterministic key derivation using Argon2id achieving 103–312 bits entropy without persistent storage. GPU attack resistance ($\approx 10^{16}$ years for 80-bit master secrets under Paranoid parameters) derives from memory-hardness: 128 MiB allocation per hash invocation limits parallelization to 312 concurrent attempts per 40 GB GPU. Provably uniform rejection sampling eliminates modulo bias. Production implementation in Rust provides automatic zeroization and compile-time integrity verification.

The design intentionally sacrifices key isolation and rotation flexibility to eliminate vault-based attack surfaces. This trade-off suits high-stakes reproducible secrets in air-gapped environments and master credentials where deterministic regeneration is acceptable.

Open-source implementation: https://github.com/coignard/qatsi

Acknowledgments

We thank the Argon2 team for the Password Hashing Competition, the Electronic Frontier Foundation for the Large Wordlist, and the Rust cryptography community for production-grade primitives. Performance benchmarks conducted on Apple M1 Pro (2021) hardware.

References

[1] A. Biryukov, D. Dinu, D. Khovratovich, S. Josefsson. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications.* RFC 9106, September 2021.

[2] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0.* RFC 2898, September 2000.

[3] M-J. Saarinen, J-P. Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC).* RFC 7693, November 2015.

[4] Y. Nir, A. Langley. *ChaCha20 and Poly1305 for IETF Protocols.* RFC 8439, June 2018.

[5] Electronic Frontier Foundation. *EFF's New Wordlists for Random Passphrases.* https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases, July 2016.

[6] A. G. Reinhold. *The Diceware Passphrase Home Page.* https://theworld.com/~reinhold/diceware.html, 1995.

[7] M. Palatinus, P. Rusnak, A. Voisine, S. Bowe. *BIP-39: Mnemonic code for generating deterministic keys.* Bitcoin Improvement Proposal, 2013. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[8] Password Hashing Competition. https://www.password-hashing.net/, 2015.

[9] C. Percival. *Stronger Key Derivation via Sequential Memory-Hard Functions.* BSDCan 2009.

[10] OWASP Foundation. *Password Storage Cheat Sheet.* https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html, 2024.

[11] Hashcat. *Hashcat Benchmarks.* https://hashcat.net/hashcat/, 2024.

[12] D. Lemire. *Fast Random Integer Generation in an Interval.* ACM Transactions on Modeling and Computer Simulation, 29(1), 2019.

[13] L. K. Grover. *A fast quantum mechanical algorithm for database search.* In Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), pages 212–219, 1996.