Qwen

Architectural Evolution in Cryptographic Design: A Comparative Analysis of BLAKE vs. SHA-2/3 and Argon2id for Secure, Auditable, and Energy-Efficient AI Chat Systems

Comparative Security Implications: From Monolithic Risk to Defense-in-Depth Architecture

The transition from a cryptographic design centered on the BLAKE2b-256 primitive to one utilizing a specialized stack of Argon2id/scrypt and SHA-2/SHA-3 represents a fundamental shift from a monolithic risk model to a defense-in-depth architecture. The original design's reliance on a single algorithm for disparate security objectives—password hashing and contribution fingerprinting—created a consolidated risk surface that could be exploited if any weakness were discovered . This approach violates the principle of least privilege applied to cryptographic primitives, where each function should be used for its intended purpose without overlap. The corrected design directly addresses this by segregating concerns, assigning memory-hard Key Derivation Functions (KDFs) exclusively to the task of securing user credentials and employing fast, collision-resistant hashes for data integrity verification. This separation is not merely an optimization but a critical security enhancement that materially reduces the potential blast radius in the event of a partial system compromise .

The most significant security upgrade lies in the adoption of memory-hard KDFs for password storage. Passwords require protection against offline brute-force and dictionary attacks, which are often executed at massive scale using specialized hardware like GPUs and ASICs [46]. Simple, fast hash functions are ill-suited for this task because they allow attackers to compute billions of guesses per second with minimal resource overhead. In contrast, memory-hard functions like Argon2id and scrypt are engineered to be computationally expensive and, more importantly, to consume a large amount of memory during their computation [11] [12]. Argon2id, the winner of the Password Hashing Competition in 2015, achieves this by filling a configurable block of RAM with

pseudorandom data derived from the password and salt, forcing adversaries to allocate substantial physical memory for each guess they wish to test [11] [45]. This dramatically increases the cost of parallel attacks; studies have shown that applying Scrypt with sufficient memory can limit an attacker's parallel processing power to as low as 0.37 H/s [14]. This property makes GPU and ASIC-based cracking operations prohibitively expensive in terms of both capital expenditure (for memory modules) and operational costs (for electricity and cooling), thereby providing a robust defense against modern password cracking techniques [10] [13]. The use of Argon2id or scrypt in the corrected design ensures that even if an attacker gains access to the database of hashed passwords, deriving the original plaintext passwords would be a highly resource-intensive endeavor, effectively raising the barrier to entry [46].

Conversely, the requirement for contribution fingerprinting is fundamentally different. For this purpose, the primary goals are determinism, speed, and strong collision resistance. A contribution ID must be a unique, verifiable identifier for a piece of data, and the process of generating it must be efficient enough to handle high-volume workloads. Memory-hardness is an unnecessary and counterproductive property in this context. Here, the corrected design appropriately substitutes BLAKE2b-256 with standardized hash functions like SHA-256 or SHA3-256 . These algorithms are the product of extensive public review and standardization efforts by bodies like NIST [2] [26]. They are optimized for high throughput and exhibit excellent diffusion properties, making it computationally infeasible to find two different inputs that produce the same output (a collision). Empirical evaluations show that SHA-256 performance is competitive with BLAKE2b, often achieving around 11.8 cycles per byte on modern processors [17]. In some complex scenarios, such as certain Proof-of-Work applications, SHA-256 has demonstrated superior performance due to its deep integration and optimization in both hardware and software ecosystems [16]. By using these well-vetted primitives for fingerprints, the system ensures data integrity without introducing the computational and energy overhead associated with memory-hard functions. This clear separation of duties—using Argon2id/scrypt for credential security and SHA-2/SHA-3 for data integrity—is a cornerstone of sound cryptographic engineering, as it prevents a vulnerability in one domain from contaminating the other .

The following table provides a comparative analysis of the two cryptographic designs based on their security implications.

Feature	Original Design (BLAKE2b-256)	Corrected Design (Argon2id/SHA-2+SHA-3)
Primary Use Case	Single primitive for both password hashing and contribution fingerprinting.	Dedicated primitives for distinct purposes: memory-hard KDF for passwords, standard hash for fingerprints.
Password Security	Vulnerable to GPU/ASIC-based offline brute-force attacks due to lack of memory hardness.	Highly resistant to GPU/ASIC-based attacks via memory-hard KDFs (Argon2id/scrypt) that increase cost per guess [14] [45].
Fingerprint Integrity	Provides deterministic hashing but relies on a single algorithm for both security roles.	Provides strong collision resistance and deterministic hashing for fingerprints using standardized, widely reviewed functions (SHA-256/SHA3-256) [15].
Risk Surface	Monolithic; a weakness in BLAKE2b could impact both password security and data integrity simultaneously.	Segregated; a vulnerability in one primitive (e.g., SHA-2) would not compromise the security of the other (e.g., Argon2id) .
Standards Compliance	Not explicitly aligned with the "no BLAKE" doctrine; BLAKE family may have limited FIPS compliance [43].	Fully compliant with the "no BLAKE" doctrine and uses primitives recommended by NIST [56].

This segregation of concerns aligns with modern cryptographic best practices, including those outlined in NIST Special Publication 800-63B, which advocates for the use of strong, adaptive hashing algorithms for password storage [27] [56]. The corrected design moves away from a minimalist philosophy of using one versatile tool for everything towards a more robust, specialized toolkit where each component is chosen for its specific strengths. This architectural change provides a more resilient security posture, directly addressing the latent risks present in the original monolithic design.

Energy Efficiency and Sustainable Computing in High-Volume Hashing

While security is the paramount concern, the operational sustainability of a high-volume AI-chat platform necessitates a rigorous evaluation of energy consumption. The corrected cryptographic design, far from being a burden in terms of energy efficiency, is architected to be both powerful and sustainable through deliberate optimizations. The analysis reveals that the choice of primitives is balanced, and the overall system design incorporates patterns that actively reduce energy usage, a concept central to sustainable computing [1]. Globally, data centers already account for about 1% of the world's electricity usage, making energy-aware design a critical responsibility for large-scale systems [1].

Empirically, the performance of the selected hash functions is highly competitive. Standard hash functions like SHA-256 and SHA3-256 are designed for speed and are

heavily optimized in modern CPU microarchitectures. Performance benchmarks indicate that SHA-256 can achieve approximately 11.8 cycles per byte, a measure of its computational efficiency [17]. While some comparative studies suggest no significant performance difference between SHA-2, SHA-3, and BLAKE2 in general-purpose scenarios [48], others highlight that SHA-256's maturity in hardware implementations gives it an edge in complex, real-world workloads [16]. For the purpose of generating contribution fingerprints—a task performed frequently and in bulk—the speed of SHA-256 or SHA3-256 is ideal, allowing for rapid processing of user contributions without creating a performance bottleneck. The energy cost per byte for these algorithms is well-understood and forms the baseline for the system's efficiency metrics [65].

The more nuanced aspect of energy efficiency relates to the memory-hard KDFs used for password hashing. Argon2id and scrypt are inherently slower than simple hashes because their security model is predicated on consuming a significant amount of memory, which is a more expensive resource than CPU cycles [12]. However, this slowness is not an inefficiency but a direct investment in security. The energy consumed in executing Argon2id is a tangible cost imposed on an attacker attempting a brute-force attack. The key insight is that this energy expenditure should be tuned to provide an acceptable level of security against the expected threat model. The corrected design facilitates this tuning through several mechanisms. First, the inclusion of `avg_energy_nj_per_hash` in the eco-metrics provides a concrete, quantifiable measure of the system's energy consumption . This allows developers to move beyond theoretical models and make decisions based on empirical data collected from the target infrastructure.

Second, the architectural pattern of asynchronous batching is a powerful technique for improving energy efficiency at scale . Instead of performing cryptographic operations individually and incurring the full overhead of system calls and cache misses for each one, the system groups multiple operations into a single batch. Processing workloads in larger chunks improves CPU cache behavior and reduces lock contention, leading to better overall throughput and a lower average energy cost per operation [18] [65]. The proposed `hash_batch_size: 256` parameter in the ALN file is a practical example of this tunable optimization . By measuring the energy required for different batch sizes, operators can find the sweet spot that maximizes throughput while minimizing idle power draw. This data-driven tuning is reflected in the `energy_reduction_vs_naive_pct: 31.0` metric, which quantifies the savings achieved through these optimizations .

Furthermore, the specified infrastructure context—a "CPU-only, green-scheduled" environment—is strategically advantageous for this design . On CPUs, the parallel

processing capabilities that make GPUs so effective for attacking simple hashes are less pronounced when dealing with memory-bound algorithms like Argon2id and scrypt. This means that the chosen KDFs remain effective deterrents against attacks launched from commodity hardware. The focus on CPU efficiency, combined with the ability to measure and tune energy consumption using tools like RAPL (Running Average Power Limit) via Linux `perf` events, creates a closed-loop system for managing the environmental impact of the platform [59] [60] [71]. This approach aligns with the principles of sustainable computing, which emphasize optimizing software to reduce energy consumption as a core part of responsible system design [1] [21]. The corrected design thus successfully balances the need for high-security cryptography with the imperative of operational sustainability.

Policy-Driven Cryptography: Enhancing Auditability and Governance

The corrected cryptographic design introduces a paradigm shift from hard-coded cryptographic logic to a flexible, auditable, and governable policy-driven model. This approach, where algorithms and their parameters are defined as structured data rather than embedded in code, provides profound benefits for long-term security management, regulatory compliance, and system maintainability . The core of this innovation is the `HashPolicy` registry, which serves as a formal declaration of the cryptographic rules governing a set of accounts . By storing this information in a versioned schema within the ALN file, the system becomes transparent and adaptable, enabling reproducible audits and smooth transitions as cryptographic standards evolve .

The primary advantage of this data-driven approach is enhanced auditability. When a security auditor inspects an account record, they can immediately see not only the resulting hash values but also the precise `HashPolicy` that was used to generate them. The policy includes details such as the chosen password KDF (`argon2id`), its specific parameters (`password_memory_kib`, `password_time_cost`, `password_parallelism`), the key derivation function (`HKDF-SHA256`), and the contribution hash algorithm (`SHA3-256`) . This transparency allows auditors to verify that the cryptographic choices made at the time of account creation were appropriate and aligned with contemporary best practices. It eliminates guesswork and ambiguity, providing a clear, immutable record of the security posture of every account. This is a stark contrast to systems with hardcoded parameters, where the rationale behind specific

settings may be lost over time, and verifying compliance requires manual inspection of source code.

Beyond auditability, the policy-driven model enables a smooth and non-disruptive migration path for strengthening security. Cryptographic best practices and computational power are constantly evolving. An algorithm or parameter set considered secure today may become vulnerable tomorrow. The versioned `HashPolicy` registry provides a mechanism to manage this evolution gracefully . New policies can be introduced with stronger parameters, and old accounts can be migrated to the new policy over time. The proposed design facilitates this through the `rehash_on_login: true` flag . When an older account logs in, the system can detect its outdated policy, transparently recompute the password hash using the current best-practice algorithm and parameters, and update the stored value without requiring any action from the user. This asynchronous, background-upgrade process ensures that the entire user base can be brought up to the highest level of security without service interruptions or the need for a mass, disruptive database migration . This forward-looking architecture demonstrates a commitment to continuous security improvement.

In terms of compliance, this design is exceptionally robust. It directly satisfies the explicit "no BLAKE" doctrinal constraint by completely removing the BLAKE-family hash from the cryptographic stack . Furthermore, the selected primitives—Argon2id, scrypt, SHA-2, SHA-3, and HKDF—are all widely recognized and standardized by reputable organizations like NIST [2] [26]. Their use ensures alignment with industry standards and reduces the risk associated with proprietary or less-reviewed algorithms. Many of these primitives are also available in FIPS 140-3 validated modules, which is essential for enterprise-grade systems handling sensitive data [39] [54]. Major platforms like Red Hat Enterprise Linux 9.6 include packages and modules for these algorithms, further cementing their status as foundational, interoperable components of a secure system [7] [8]. By codifying these choices within a clear, auditable policy, the system not only meets its immediate doctrinal requirements but also builds a foundation for enduring compliance and adaptability in a changing security landscape.

Rust-Specific Implementation Patterns for Secure and Efficient Execution

The Rust programming language offers a uniquely powerful combination of performance, safety, and control, making it an ideal choice for implementing the corrected cryptographic design. Its core philosophy of preventing common classes of bugs at compile time aligns perfectly with the stringent requirements of cryptographic software, where implementation flaws can lead to catastrophic security failures [44]. Leveraging Rust's specific features and its rich ecosystem of cryptographic crates allows for the construction of a system that is not only secure but also highly efficient and maintainable.

One of Rust's most significant advantages is its guarantee of memory safety without a garbage collector. Through its strict ownership and borrowing rules, Rust's compiler prevents entire categories of vulnerabilities that plague languages like C and C++, such as buffer overflows, use-after-free errors, and double frees [44]. These types of flaws are particularly dangerous in cryptographic contexts, as they can leak secret keys or allow an attacker to manipulate internal state. By building cryptographic wrappers and core logic in Rust, developers can enforce these safety guarantees across the entire codebase. Furthermore, Rust's emphasis on zero-cost abstractions and its ability to generate highly optimized machine code ensure that there is no performance penalty for these safety features, a critical requirement for a high-volume hashing system. This safety-first approach is complemented by Rust's suitability for writing constant-time code, which is essential for mitigating timing-channel side-channel attacks that can leak information from cryptographic operations .

The Rust ecosystem provides mature, well-maintained, and secure crates for all the required cryptographic primitives. For instance, the `sha2` and `sha3` crates offer performant implementations of the SHA-2 and SHA-3 hash functions, while `hkdf` provides a safe wrapper for the HMAC-based Key Derivation Function [4] [72]. For the memory-hard KDFs, the `ring` crate is a comprehensive, battle-tested library that includes a highly optimized and secure implementation of Argon2 [38]. Using these established crates is preferable to implementing algorithms from scratch, as it leverages the expertise of the wider security community and reduces the risk of introducing subtle implementation errors. The `argon2-cffi-bindings` crate provides bindings for Argon2, demonstrating its integration into various language ecosystems [37]. This wealth of high-quality, auditable code allows developers to build upon a solid foundation rather than reinventing the wheel.

The proposed architectural pattern of an asynchronous worker for off-chain tasks like rehashing and batch processing is a natural fit for Rust's concurrency model. The language's async/await syntax, combined with popular runtimes like `tokio` or `async-std`, provides a clean and efficient way to manage thousands of concurrent I/O-bound and CPU-bound operations without the overhead of threads [35]. This is crucial for implementing the `rehash_on_login` feature and the bulk contribution hashing described in the design, as these tasks can be scheduled in the background without blocking the main application thread or degrading the user experience . The `HashPolicy` structure defined in the ALN file can be easily represented as a Rust struct, which can be deserialized from the configuration file. This struct can then be used to parameterize the hashing routines, embodying the policy-driven cryptography concept in a type-safe manner. The combination of Rust's safety guarantees, performance characteristics, and a rich ecosystem of cryptographic libraries makes it an exceptionally well-suited language for securely and efficiently realizing the corrected cryptographic design.

Cross-Platform Compatibility and Long-Term Maintainability

For a distributed system like an AI-chat platform, the ability for components written in different programming languages to interoperate seamlessly is not an optional feature but a fundamental requirement for scalability and long-term maintainability. The corrected cryptographic design, by relying on a set of universally adopted and standardized primitives, ensures robust cross-platform compatibility. The selection of Argon2id, scrypt, SHA-256, SHA3-256, and HKDF is a strategic choice that prioritizes interoperability, guaranteeing that future non-Rust components can generate and verify cryptographic outputs that are consistent with the core system .

These cryptographic primitives are not tied to a specific language or vendor; they are foundational elements of modern cryptography, standardized by bodies such as ISO/IEC and NIST [2] [41] [42]. Their ubiquity means that robust, well-tested implementations exist across virtually every major programming language, including Python, Go, Java, JavaScript, and C++. For example, Argon2 itself is noted for its wide portability, supporting languages like C, C++, Go, Rust, Python, Java, JavaScript, and Swift [53]. Similarly, the Node.js `crypto` module provides built-in functionality for SHA-2, and the Python `hashlib` library supports SHA-256 and SHA3-256 [50]. This widespread

availability ensures that a microservice written in Go for analytics, a data processing pipeline in Python, or a client-side application in JavaScript can all generate valid contribution fingerprints or derive keys using HKDF-SHA256 in a manner that is cryptographically compatible with the Rust-based backend.

The backing of major operating systems and enterprise-grade platforms further solidifies this compatibility. For instance, Red Hat Enterprise Linux 9.6 provides new packages for HKDF, PBKDF2, and SHA-3, indicating strong, integrated support for these standards within a production-ready environment [7] [8] [51]. The Go Cryptographic Module is under review for FIPS 140-3 certification, a key requirement for government and regulated industries, which validates its security and reliability [52]. This institutional support ensures that the primitives are not just academic curiosities but are battle-tested and trusted components of secure software development. This broad support network significantly reduces the maintenance burden, as updates and security patches are handled by a diverse community of contributors and vendors, rather than being the sole responsibility of a single project team. It also lowers the barrier to entry for new developers who may be more familiar with a language other than Rust, as they can rely on stable, well-documented libraries to interact with the core cryptographic logic. This design choice fosters a healthy, multi-language ecosystem around the platform, promoting modularity and specialization while maintaining a consistent and secure cryptographic backbone.

Reference

1. https://cdn.qwenlm.ai/qwen_url_parse_to_markdown/system00-0000-0000-0000-webUrlParser?key=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyZXNvdXJjZV91c2VyX2lkIjoicXdlbl91cmxfcGFyc2VfdG9fbWFya2Rvd24iLCJyZXNvdXJjZV9pZCI6InN5c3RlbTAwLTAwMDAtMDAwMC0wMDAwLXdlYlVybFBhcnNlciIsInJlc291cmNlX2NoYXRfaWQiOm51bGx9.cz1eeZEZdaQH5CgUaxwUmfEJfqTOZMoh3PbosHslSPA
2. Algorithms, Key Size and Protocols Report (2018) https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ba203b9b&appId=PPGMS
3. 9.6 Release Notes | Red Hat Enterprise Linux | 9 https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html-single/9.6_release_notes/index

4. Software Packages in "trixie", Subsection rust https://packages.debian.org/stable/rust/
5. Red Hat Enterprise Linux 9 9.6 发行注记 https://docs.redhat.com/zh-cn/documentation/red_hat_enterprise_linux/9/pdf/9.6_release_notes/Red_Hat_Enterprise_Linux-9-9.6_Release_Notes-zh-CN.pdf
6. 9.6 リリースノート | Red Hat Enterprise Linux | 9 https://docs.redhat.com/ja/documentation/red_hat_enterprise_linux/9/html-single/9.6_release_notes/index
7. Red Hat Enterprise Linux 9 9.6 リリースノート https://docs.redhat.com/ja/documentation/red_hat_enterprise_linux/9/pdf/9.6_release_notes/Red_Hat_Enterprise_Linux-9-9.6_Release_Notes-ja-JP.pdf
8. Red Hat Enterprise Linux 9 9.6 릴리스 노트 https://docs.redhat.com/ko/documentation/red_hat_enterprise_linux/9/pdf/9.6_release_notes/Red_Hat_Enterprise_Linux-9-9.6_Release_Notes-ko-KR.pdf
9. Does FIPS 140-2 allow using MD5 for checksum https://stackoverflow.com/questions/73771473/does-fips-140-2-allow-using-md5-for-checksum
10. Optimized Implementation of Argon2 Utilizing the Graphics ... https://www.mdpi.com/2076-3417/13/16/9295
11. Argon2, Memory-hard Hash Function https://zhuanlan.zhihu.com/p/80490980
12. Argon2: New Generation of Memory-Hard Functions for ... https://www.researchgate.net/publication/303032281_Argon2_New_Generation_of_Memory-Hard_Functions_for_Password_Hashing_and_Other_Applications
13. 2504.17121v2 | PDF | Password | Computing https://www.scribd.com/document/946834665/2504-17121v2
14. Password-Hashing Status https://pdfs.semanticscholar.org/99f2/a662fcfa500f654763d8d74640fd98f2eefd.pdf
15. Performance results per byte of the SHA256 and SHA3- ... https://www.researchgate.net/figure/Performance-results-per-byte-of-the-SHA256-and-SHA3-256-hash-functions-on-the-three_fig3_356509839
16. Performance Evaluation of SHA-256 and BLAKE2b in ... https://www.academia.edu/91352263/Performance_Evaluation_of_SHA_256_and_BLAKE2b_in_Proof_of_Work_Architecture
17. Performance Evaluation of SHA-256 and BLAKE2b in ... https://www.researchgate.net/publication/360869188_Performance_Evaluation_of_SHA-256_and_BLAKE2b_in_Proof_of_Work_Architecture
18. Security modeling and efficient computation offloading for ... https://arxiv.org/pdf/1907.02506

19. SHIELD: Security-Aware Scheduling for Real-Time DAGs ... https://dl.acm.org/doi/full/10.1145/3702236
20. Secure IoT Communications with Optimized AES and ... https://ieeexplore.ieee.org/iel8/30/8306365/11301744.pdf
21. CLKSCREW: Exposing the Perils of Security-Oblivious Energy ... https://www.cs.columbia.edu/~simha/preprint_USENIX17_clkscrew.pdf
22. Clock Frequency Impact on the Performance of High- ... https://www.mdpi.com/1424-8220/19/1/15
23. (PDF) Dynamic Voltage and Frequency Scaling as a ... https://www.researchgate.net/publication/378354921_Dynamic_Voltage_and_Frequency_Scaling_as_a_Method_for_Reducing_Energy_Consumption_in_Ultra-Low-Power_Embedded_Systems
24. Best Practices: Salting & peppering passwords? [closed] https://stackoverflow.com/questions/16891729/best-practices-salting-peppering-passwords
25. https://ftp.sjtu.edu.cn/sites/download.opensuse.org/ ... https://ftp.sjtu.edu.cn/sites/download.opensuse.org/distribution/leap-micro/5.3/product/repo/Leap-Micro-5.3-x86_64-Media/repodata/7cd2073b13974dd4dab4c85fac9f08fe7c29a06001b51514060cd3d5cb8b6b0e-other.xml.gz
26. Dokumen - Pub Practical Cryptography For Developers ... https://www.scribd.com/document/820484096/Dokumen-pub-Practical-Cryptography-for-Developers-9786190008705
27. System Password Security: Attack and Defense Mechanisms https://arxiv.org/html/2510.10246v1
28. Advanced Information Networking and Applications https://link.springer.com/content/pdf/10.1007/978-3-031-87763-6.pdf
29. PhD-Syllabus -CSE | PDF | Routing | Operating System https://www.scribd.com/document/975123457/PhD-Syllabus-CSE
30. Computational Science – ICCS 2024 https://link.springer.com/content/pdf/10.1007/978-3-031-63751-3.pdf
31. PhD Thesis on Memory Disaggregation | PDF https://www.scribd.com/document/712610242/2023-Practical-Memory-Disaggregation-thesis
32. Computational Science – ICCS 2025 https://link.springer.com/content/pdf/10.1007/978-3-031-97635-3.pdf
33. B.tech ECE Syllabus | PDF | Academic Degree https://www.scribd.com/document/578539028/B-tech-ECE-Syllabus-18-2
34. JNTUK MCA Course Structure 2019 | PDF https://www.scribd.com/document/461223268/JNTUK-MCA-R19-Syllabus

35. PDF Rust Web Programming Third Edition Early Access ... https://www.scribd.com/document/847585559/PDF-Rust-Web-Programming-Third-Edition-Early-Access-Maxwell-Flitton
36. Ali Hassan - Threat Detection • Digital Forensics https://pk.linkedin.com/in/ali-hassan-engineer
37. Simple Index https://mirrors.sustech.edu.cn/pypi/simple/
38. Software Packages in "forky" https://packages.debian.org/testing/allpackages
39. Microsoft PKI Services Public TLS Certification Practice ... https://www.microsoft.com/pkiops/Docs/Content/policy/Microsoft_PKI_Services_public_tls_CPS_v3.3.2.pdf
40. Effective Cybersecurity A Guide to Using Best Practices ... https://pdfcoffee.com/effective-cybersecurity-a-guide-to-using-best-practices-and-standardspdf-pdf-free.html
41. ISO/IEC 18033-4:2011(en), Information technology ... https://www.iso.org/obp/ui/en/#!iso:std:54532:en
42. ISO/IEC 18033-4:2005, Information technology https://www.amazon.com/ISO-IEC-18033-4-Information-technology/dp/B000XYT4AK
43. Python and OpenSSL FIPS mode - Ideas https://discuss.python.org/t/python-and-openssl-fips-mode/51389
44. Cryptographic Implementation Flaws: Modern Encryption ... https://dev.to/rafalw3bcraft/cryptographic-implementation-flaws-modern-encryption-analysis-2jjj
45. Qatsi: Stateless Secret Generation via Hierarchical Memory ... https://arxiv.org/pdf/2510.18614
46. Argon2 vs bcrypt vs. scrypt: which hashing algorithm is ... - Stytch https://stytch.com/blog/argon2-vs-bcrypt-vs-scrypt/
47. Evaluating the Energy Costs of SHA-256 and SHA-3 ... https://www.mdpi.com/2624-831X/6/3/40
48. Evaluation of Hash Algorithm Performance for ... https://arxiv.org/pdf/2408.11950
49. Extending Composable Data Services to the Realm of ... https://search.proquest.com/openview/d39cfcf73a236d808a0ae75a6f4f3529/1?pq-origsite=gscholar&cbl=18750&diss=y
50. Crypto | Node.js v25.3.0 Documentation https://nodejs.org/api/crypto.html
51. 9.6 发行注记| Red Hat Enterprise Linux | 9 https://docs.redhat.com/zh-cn/documentation/red_hat_enterprise_linux/9/html-single/9.6_release_notes/index
52. 9.6 Release Notes | Red Hat Enterprise Linux | 9 https://docs.redhat.com/fr/documentation/red_hat_enterprise_linux/9/html-single/9.6_release_notes/index
53. ASecuritySite Index https://asecuritysite.com/index

54. Federal Information Processing Standard 140-3 ... https://www.ibm.com/docs/en/quantum-safe/quantum-safe-explorer/2.x?topic=findings-fips140-3-compliance-reporting
55. Application Security Verification Standard | PDF https://www.scribd.com/document/969467306/Application-Security-Verification-Standard
56. NIST 800-63B: Modern Password Guidelines Explained https://synivate.com/blog/nist-800-63b-modern-password-guidelines-explained
57. 333333 23135851162 the 13151942776 of 12997637966 https://www.cs.princeton.edu/courses/archive/spring25/cos226/assignments/autocomplete/files/words-333333.txt
58. https://www.researchgate.net/file.PostFileLoader.h... https://www.researchgate.net/file.PostFileLoader.html?id=563869346143256c208b45ba&assetKey=AS:291613667545089@1446537524905
59. how to access RAPL via perf with Rocket Lake? https://stackoverflow.com/questions/66989354/how-to-access-rapl-via-perf-with-rocket-lake
60. [linux] RAPL perf event scale should be determined by sysfs files https://bugzilla.mozilla.org/show_bug.cgi?id=1794941
61. ChangeLog-6.1.113 https://www.kernel.org/pub/linux/kernel/v6.x/ChangeLog-6.1.113
62. stress-ng(1) - testing https://manpages.debian.org/testing/stress-ng/stress-ng.1.en.html
63. ISO/IEC 18033-2:2006(en), Information technology https://www.iso.org/obp/ui/en/#!iso:std:37971:en
64. Key derivation function: key-hash based computational ... https://pmc.ncbi.nlm.nih.gov/articles/PMC11419645/
65. Energy cost per byte of the SHA256 and SHA3-256 hash ... https://www.researchgate.net/figure/Energy-cost-per-byte-of-the-SHA256-and-SHA3-256-hash-functions-on-the-three-considered_fig5_356509839
66. ChangeLog-5.10.227 https://www.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.10.227
67. ChangeLog-5.15.168 https://mirrors.aliyun.com/linux-kernel/v5.x/ChangeLog-5.15.168
68. ChangeLog-6.10.3 https://www.kernel.org/pub/linux/kernel/v6.x/ChangeLog-6.10.3
69. ISO/IEC 18033-2:2006/FDAmd 2 - Information technology https://www.iso.org/standard/86890.html

70. ISO/IEC 18033-1:2021(en), Information security https://www.iso.org/obp/ui/#iso:std:iso-iec:18033:-1:ed-3:v1:en
71. Rust性能分析库criterion-perf-events的使用 - 鸿蒙- IT营 https://bbs.itying.com/topic/688a72542cb460013cc0ea0a
72. Software Packages in "sid", Subsection rust https://packages.debian.org/sid/rust/
73. Software Packages in "forky", Subsection rust https://packages.debian.org/testing/rust/