

Breaking Trust Without Breaking Math

AI-Driven Systemic Cryptographic Compromise Scenarios

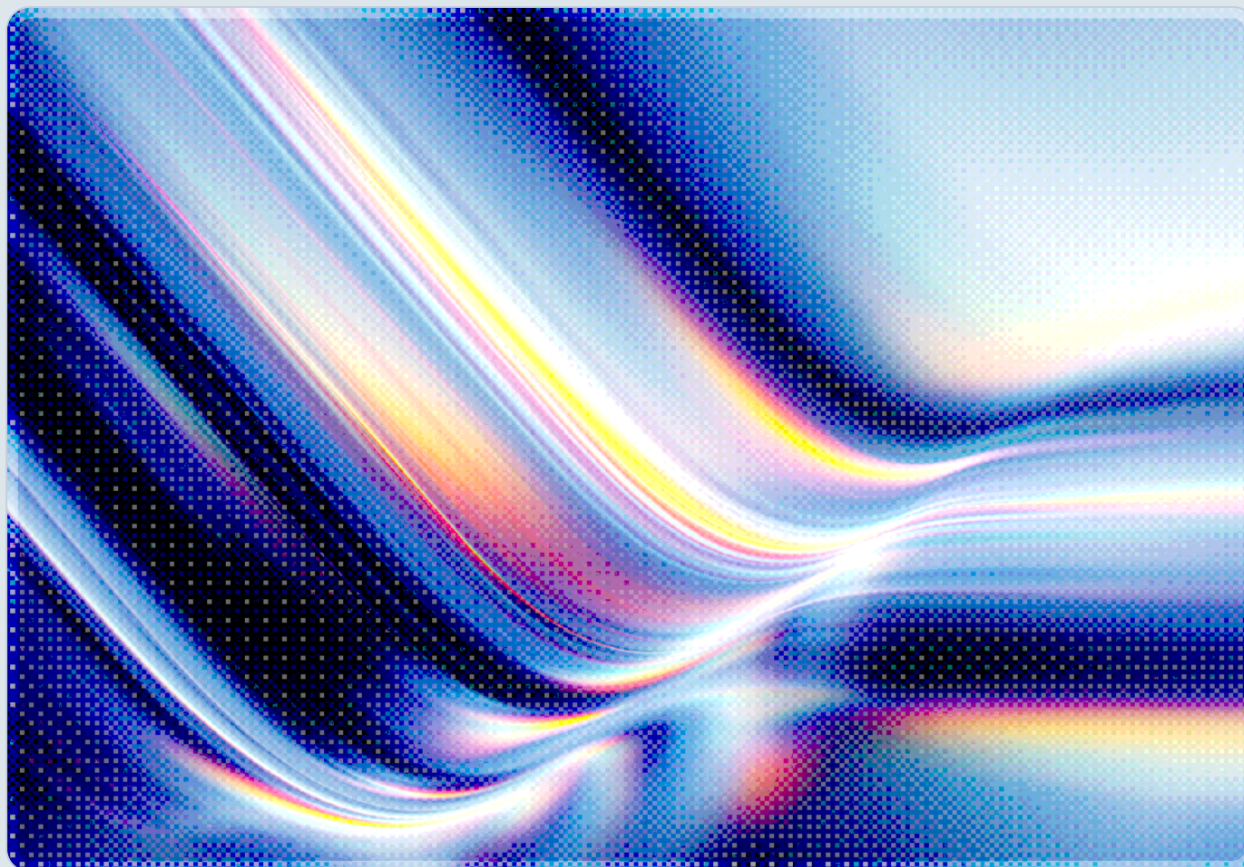
OPEN-WEIGHT MODEL RESEARCH

Produced with z-ai/glm-5.3

2026-10-30



AI-assisted Rapid Research



CSAI

cloud
CSA security
alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

The research in this document was produced with the open-weight model(s) z-ai/glm-5.3 rather than CSA's standard Claude-based pipeline.

Table of Contents

AI-Driven Systemic Cryptographic Compromise Scenarios	3
Executive Summary	4
1. Introduction: Purpose, Method, and Boundaries	6
2. The Source Analysis in Brief	7
2.1 Precision about what can fail	
2.2 The 2026 evidence, concentrated in the software record	
2.3 The implementation pathway: local defect, shared consequence	
2.4 The systemic-trust pathway and what AI changes	
3. Scenario Design and the Common Template	10
4. The Scenarios	11
4.1 Scenario 1 – The Clone in the Chain: certificate-verification bypass in a widely embedded TLS library	
4.2 Scenario 2 – The Certified Weak Generator: key-generation flaws in certified HSM, TPM, and secure-element firmware	
4.3 Scenario 3 – The Migration Window: post-quantum implementation defects found during migration	
4.4 Scenario 4 – The Authorized Signer: abuse of the authorization plane of a software-update signing service	
4.5 Scenario 5 – The Borrowed Key: cloud KMS and HSM management-plane flaws granting cross-tenant use of non-exportable keys	
4.6 Scenario 6 – The Minted Identity: token forgery in an identity-federation provider	
4.7 Scenario 7 – The Tainted Well: entropy and RNG defects in common libraries or operating systems	
4.8 Scenario 8 – The Root of the Root: certificate authorities, transparency logs, and recovery paths, including the Merkle Tree Certificate transition	
5. The Scenarios Compared with the Quantum Threat	20
6. Defenses and Recovery Capabilities by Scenario	23
7. Tabletop Exercise Injects	25
8. Conclusions and Recommendations	27
CSA Resource Alignment	27
References	28

AI-Driven Systemic Cryptographic Compromise Scenarios

Executive Summary

Most public discussion of cryptography failing at scale starts from the mathematics: a quantum computer, or an artificial intelligence system, finds a flaw in the algorithm itself and every key using that algorithm becomes unsafe. This paper takes a different starting point. It develops eight hypothetical scenarios in which cryptography keeps working exactly as designed – the algorithms are sound, the keys are strong, the signatures verify – and yet organizations everywhere lose cryptographic protection at the same time. In each scenario, the failure is in the software around the mathematics: a widely embedded library that accepts forged certificates, a hardware key generator that produces predictable keys, a signing service whose authorization is abused, post-quantum migration code whose implementation turns out to hold the weakest link, a cloud key service whose management plane lets one tenant use another's keys, an identity provider that mints tokens for the wrong people, a random-number defect shared by millions of devices, or a certificate authority whose compromise propagates to everyone who trusts it.

The common thread is shared dependency. Modern organizations do not each run their own cryptography; they rely on the same handful of libraries, hardware vendors, signing services, key managers, identity providers, and certificate authorities. When one of those shared components fails in a way that defeats authentication or exposes keys, the blast radius is set by the dependency graph, not by the attacker. Artificial intelligence matters to these scenarios for one specific, well-evidenced reason: it has made finding and combining software defects dramatically cheaper and faster. In 2026, wolfSSL credited fourteen CVEs in a single widely used TLS library to AI-assisted or AI-laboratory sources, including one that let forged certificate-authority "clones" pass verification in common build configurations [1]. AI-assisted teams found pre-authentication remote code execution flaws in the PHP interpreter dating back to the PHP 5.4 era (2012), and those flaws were recorded in CISA's exploited-vulnerability catalog within days of disclosure [2][3][4]. A single researcher's AI workflow produced 110 verified findings in one week, contributing to 89 of 119 first-round candidate algorithms in China's next-generation algorithm program being found faulty within seven days [6]. That is the tempo these scenarios are built around: not a smarter attack on the math, but machine-speed discovery of the weak points in the machinery around the math.

Why can this happen sooner than the quantum threat? Because nothing new has to be built first. A cryptographically relevant quantum computer does not exist, and no peer-reviewed or government estimate turns today's hardware into one on a predictable calendar [20]. The AI-driven scenarios in this paper require only software that already exists, attackers who already exist, and a discovery capability that was demonstrated repeatedly in 2026 [1][2][6][14]. Governments have set post-quantum migration deadlines of 2030 and 2035 [20][21] – a schedule that itself reflects the quantum threat's visible, multi-year arrival. The scenarios here have no such schedule. They could begin with the next advisory against the next shared library. The demonstrated capability is discovery, not reliable deployment: Google's threat-intelligence measurement found confirmed exploitation for only a handful of the 2,076 AI- and LLM-stack vulnerabilities disclosed in its study window, with exploitation growth driven by weaponization of known bugs rather than new discoveries [14]. The scenarios below are preparation instruments, not predictions.

For a security leader, the most useful way to compare these two threats is not "which is bigger" but "which properties do I lose, how fast, and what does recovery cost." The table below gives the comparison at a glance, in plain terms.

Question	Quantum threat	AI-driven systemic scenarios
What must exist first?	A large fault-tolerant quantum computer – none exists, and no peer-reviewed or government estimate dates one [20]	Nothing new; today's software and today's attackers, with 2026-demonstrated discovery speed [1][2][6]
What actually fails?	The mathematics of RSA and elliptic curves	The systems and processes around the mathematics – verification logic, key generation, signing authorization, trust infrastructure
How much warning?	Years of public hardware milestones and government deadlines (2030/2035) [20][21]	Little to none; disclosure-to-exploitation measured in days in 2026 [3][4]
What is lost first?	Confidentiality of recorded traffic; later, authenticity as keys fall	Usually authenticity and integrity first (forged certificates, signed malware, minted identities); confidentiality when keys are recoverable

Question	Quantum threat	AI-driven systemic scenarios
Can it reach the past?	Yes – recorded key exchanges become decryptable	Yes, in the key-recovery and weak-key scenarios, for every key generated or recorded in the exposure window
Does patching fix it?	Migration on a planned schedule closes future exposure	Patching closes future exposure only; keys stolen and artifacts forged during the window persist and must be recovered separately
What does recovery require?	Replace algorithms across the estate, on a schedule	Revoke, rotate, re-root, redistribute, and revalidate trust under pressure, often through channels that are themselves compromised

The eight scenarios, developed for PKI, cryptographic-engineering, and product security teams in Section 5, are: a certificate-verification bypass in a widely embedded TLS library; a ROCA-style key-generation flaw in certified hardware security modules and trusted platform modules; post-quantum implementation defects surfacing during migration; abuse of the authorization plane of a software-update signing service; a cloud key-management management-plane flaw enabling cross-tenant use of non-exportable keys; token forgery in an identity-federation provider; an entropy defect in a common library or operating system; and compromise of certificate authorities, transparency logs, or recovery paths, including the transition to Merkle Tree Certificates. Every scenario is labeled hypothetical, none describes a known incident, and all stay at the architectural and defensive level: they state conditions, propagation, detection difficulty, and recovery burden, never methods of attack.

The preparedness and recovery decisions that matter most, developed in Sections 6 through 8, reduce to eight, and the first three are about knowing the estate. The cryptographic inventory should be extended beyond algorithms and libraries to authorization and dependency: which services and identities can authorize a signing or decryption operation, and which certificate authorities, update signers, key managers, identity providers, and firmware trust anchors the organization depends on. In the assessment of this initiative, most inventories stop at the key and never map the perimeter that controls its use. Key provenance belongs in the same inventory, because where and when each key was generated, and on what hardware or library, decides which keys are caught by a key-generation or entropy defect – a question many organizations cannot answer today. Cryptographic software, finally, should be treated as a critical asset class with the shortest patch service levels in the organization, because 2026 showed such flaws arriving and being weaponized at machine speed [2][14].

The remaining five decisions concern assurance and recovery. Implementation assurance should be demanded during post-quantum migration rather than used as a reason to pause it: the replacement algorithms are young code on young mathematics, and hybrid constructions and differential testing against independent implementations are the controls that fit [27][28]. Revocation and rotation at scale should be built as a measured capability, so that the organization knows in advance how long it takes to revoke and reissue every certificate class it operates. Recovery paths – escrowed keys, break-glass credentials, and out-of-band update channels – must be protected independently of the services they are meant to recover, and that independence verified rather than assumed. The ability to validate history should be built as well, since trustworthy timestamps, revocation records, and transparency logs are what separate legitimate past signatures from forged ones after a compromise window. And the whole sequence should be rehearsed with the clock running, because a recovery capability that has never been exercised is a plan, not a capability; Section 8 provides tabletop injects for exactly this purpose.

Post-quantum migration remains necessary and should continue on its existing schedule; the AI risk is a reason to strengthen the assurance and recovery wrapped around that migration, not to stop it. The organizations that fare best under both threats are likely to be those for whom replacing an algorithm is routine and recovering trust is rehearsed.

1. Introduction: Purpose, Method, and Boundaries

This whitepaper is a companion to the CSA AI Safety Initiative analysis *Before Quantum: AI and the Risk of Systemic Cryptographic Failure* (version 2.1, October 2026), which asked whether AI could cause a systemic loss of cryptographic protection before a cryptographically relevant quantum computer arrives, and answered that the question warrants preparation but not prediction. That report organized the evidence into three pathways – practical cryptanalysis, implementation and protocol failure, and systemic trust failure – and constructed a single detailed scenario for the third pathway: an attacker gaining authorized use of a widely trusted signing service whose algorithms remain sound. This document extends that work in the direction security teams most need: a set of concrete, comparable scenarios covering the distinct failure classes by which cryptographic protection can be lost systemically without any mathematical break, each worked through the same technical template, each ranked against the quantum threat, and each mapped to the defenses and recovery capabilities that would and would not hold.

Three boundaries govern everything that follows. First, every scenario is **hypothetical**. Each is labeled as such, describes no known incident, and identifies no product or provider; where a scenario is modeled on a real event or advisory, the precedent is named and cited separately from the hypothetical construction. Second, the scenarios stay at the **architectural and defensive level**: they specify the shared dependency, the component and protocol layer, the security property lost, the propagation path, the detection difficulty, and the recovery burden – never exploitation methods or attack steps, following the discipline of the source report. Third, the scenarios are **sector-neutral**. They are framed around shared dependencies – libraries, hardware, signing services, key managers, identity providers, certificate authorities – rather than around industries, because the concentration that makes these failures systemic cuts across every sector that depends on the same small set of trust anchors.

The 2026 evidence base was verified against primary sources during the preparation of this document, and the citations throughout point to publicly accessible materials, with one exception that should be stated plainly: the companion analysis itself, *Before Quantum*, has not yet been publicly released by the CSA AI Safety Initiative, so it is identified by name and section number rather than by a reference entry, and every claim this paper borrows from it is re-grounded in the public sources cited below. Section 2 summarizes the source report's definitions and evidence for readers who need the foundation; readers who know it can proceed directly to Section 3.

2. The Source Analysis in Brief

2.1 Precision about what can fail

The source report's first contribution is a set of layered definitions, and they carry through this paper because each scenario attacks a different layer.

A **cryptographic primitive** is a mathematical object with a defined security property – a block cipher such as AES, a hash function such as SHA-256, a hard problem such as factoring or the module learning-with-errors problem underlying ML-KEM [28]. A **construction** combines primitives into a scheme – a key-encapsulation mechanism, a signature scheme, an authenticated-encryption mode – and can fail through proof errors, parameter choices, or exploitable structure even when its primitives hold. An **implementation** is code realizing a construction in a particular library, product, or device, and fails through memory-safety defects, logic errors, missing checks, side channels, and flawed randomness, with the primitive and construction remaining intact. A **protocol** governs how parties use constructions across a network – TLS, SSH, a messaging handshake – and fails through state-machine defects, downgrade paths, and unanticipated interactions. **Key management** is the set of processes and systems that generate, store, distribute, use, rotate, and revoke keys; it fails when a key is exposed, used for an unauthorized purpose, or its authorization bypassed. **Trust infrastructure** is the shared layer many organizations rely on simultaneously – certificate authorities, software-update signing systems, identity federation providers, cloud key managers, firmware trust anchors, transparency services – and its defining failure property is that the cryptography keeps working correctly while serving an attacker.

Three outcomes organize the analysis, and the table below restates them.

Outcome	What fails	Evidence and interpretation required
Practical cryptanalytic break	The security of a primitive or construction at specified parameters	The exact target, attack model, resource requirements, verification status, and effect on real deployments
Implementation or protocol failure	Particular libraries, products, key-generation processes, or protocol behaviors	Affected versions and configurations, prerequisites, the security property lost, and remediation – without implying the primitive is broken
Systemic trust failure	Shared signing, identity, certificate, update, or key-management infrastructure	How a local compromise propagates across organizations, why defenses fail to contain it, and what makes recovery difficult

This paper's scenarios live almost entirely in the second and third rows: failures that leave the mathematics intact. The distinction between mechanism and consequence is load-bearing – an implementation flaw becomes systemic through widespread dependency, and a mathematical result can have no operational effect at all.

2.2 The 2026 evidence, concentrated in the software record

The source report's evidence review covers four kinds of results, kept deliberately separate. On **original cryptanalysis**, in July 2026 a frontier model produced a structural attack on HAWK, a lattice-based signature scheme then in the third round of NIST's additional-signature process: a Galois-involution reduction from HAWK key recovery to a shortest-vector problem in dimension $n/2+1$, demonstrated at the small HAWK-256 challenge parameters in hours on a single server, and estimated at 2^{108} gates at HAWK-512 – faster than the design assumed, not practical at deployment sizes [7][8]. The scheme's designers confirmed the result and withdrew HAWK the following day, after it had advanced through two rounds of NIST review without the flaw being found [7][45]. The same announcement extended a published attack on seven-round AES-128, a reduced-round variant that no one deploys, claiming no impact on production deployments [7]. On **engineering leverage rather than mathematics**, AI-built GPU implementations set two RSA factoring records in sixteen days in September 2026, with no new algorithm; RSA-1024 was placed within reach of well-funded actors at an estimated \$30 million per key, while RSA-2048 remained roughly a billion times harder [9][10]. On **competition-scale review**, 89 of 119 first-round candidates in China's next-generation algorithm program were found faulty within seven days, 110 of the findings from one researcher's agentic workflow [6].

The evidence closest to this paper's subject is the **software record**. AI-assisted discovery produced two of the most widely deployed critical vulnerabilities of 2026: PHP2Shell, two pre-authentication remote code execution flaws in the PHP interpreter present in every release back to PHP 5.4 (2012), found independently by three AI-assisted teams and recorded in CISA's Known Exploited Vulnerabilities catalog within five days of disclosure [2][3][4]; and Node2Shell, an out-of-bounds write in the Brotli decoder whose discovery is credited to members of an AI security team, which triggered coordinated patching across Microsoft, Node.js, and Python packages within days, with no exploitation in the wild reported as of late October [5]. Google's threat intelligence group measured monthly CVE disclosures doubling between January and August 2026 and found AI-discovered vulnerabilities enabling remote code execution twice as often as the general population, while bug bounty programs at major vendors paused under automated submission volume [14][17]. According to one security firm's analysis, an attributed campaign chained PHP2Shell with a kernel privilege escalation to gain root on 25 servers – 24 firewalls and one firewall management server – with an AI coding agent deployed on the actors' own infrastructure [18].

The wolfSSL record is the most precise available for cryptographic libraries specifically. Across its 2026 releases, wolfSSL credits fourteen CVEs to AI-assisted or AI-laboratory sources in three distinct attribution categories [1]. Five, fixed in version 5.9.4, came via an AI laboratory's open-source program, whose operators reported that of 74 vulnerability reports submitted to wolfSSL, all but two were valid and five became CVEs [11]. The most serious, CVE-2026-93302, rated High, is a trust-matching defect in which the library ignores the public key used, allowing forged CA clones to pass verification: a peer who knows which certificate authorities a client trusts can present a self-made certificate carrying a trusted CA's name and bypass authentication, in versions 5.3.0 through 5.9.2, widening to all CA loading under the OpenSSL-compatible defaults enabled in wolfSSL's nginx, haproxy, Apache, stunnel, bind, ffmpeg, and distribution build profiles [1]. A second, CVE-2026-93304, lets an out-of-order handshake message cause a TLS 1.2 client to install keys derived from a known value, so that for pre-shared-key connections any fake server can complete the handshake without knowing the pre-shared key [1]. Eight further CVEs, fixed in version 5.9.1 and credited to a security firm working with an AI laboratory, include an ECCSI signature verifier that accepted forged signatures for any message for any identity, an authentication bypass in AES-EAX and CMAC on oversized messages, a ChaCha20-Poly1305 decryption path that returned plaintext without checking the authentication tag, a PKCS#7 defect reducing a tag check from 2^{-128} to 2^{-8} , and a certificate-chain verification bypass in the OpenSSL compatibility layer in which an attacker holding any leaf certificate from a trusted CA could forge a certificate for any name [1]. A fourteenth, CVE-2026-5194, rated Critical, allowed undersized digests to pass signature verification across ECDSA, DSA, ML-DSA, and EdDSA, and is credited to a named researcher at an AI laboratory; the advisory does not state that a model was involved, and neither the source report nor this paper assumes it [1].

Two further wolfSSL advisories matter for the post-quantum pathway and carry no AI attribution. CVE-2026-10097, rated High, is a deterministic logic error in the AVX2-optimized ML-KEM-1024 decapsulation that compared only 1,536 of 1,568 ciphertext bytes in the Fujisaki-Okamoto consistency check; a proof of concept recovered a full ML-KEM-1024 private key with approximately 98 percent success from roughly 350 chosen ciphertexts submitted to a decapsulation oracle on a static key [1]. A similar defect in the ARM64 path compared only half the input [1]. These are the KyberSlash class of failure – implementation defects in a standardized post-quantum scheme that recover keys without touching the mathematics – and the 2024 KyberSlash timing flaws in official reference code, which recovered keys within minutes, remain the canonical example [12]. The bounded, specific conclusion the source report draws, and this paper adopts, is that AI-assisted review found real authentication, forgery, and authenticated-encryption defects in a production TLS library, each tied to particular versions and configurations, each fixed within days of report, none involving any weakness in a primitive, and none with confirmed exploitation in the wild – but several of them exactly the class of failure that becomes systemic when the affected library sits inside widely deployed products.

2.3 The implementation pathway: local defect, shared consequence

The implementation pathway requires no mathematics, and it has the most 2026 evidence behind it. Its most important historical precedent is ROCA: in 2017, researchers at Masaryk University found that an RSA key-generation library used in Infineon security chips produced primes with a specific structure that made the resulting keys factorable from their public halves alone, with no physical access and no weak random-number generator; the affected chips were NIST FIPS 140-2 and CC EAL 5+ certified devices shipping since at least 2012, embedded in authentication tokens, smartcards, and trusted-boot mechanisms across many vendors' products, and the researchers' scans confirmed on the order of 760,000 vulnerable keys in the wild, with bulk pre-selection possible without time-consuming factorization attempts [13]. RSA was not broken. Particular keys, generated by particular certified code, became recoverable, and because that code sat in the trust infrastructure of many organizations at once, a library defect was a trust-infrastructure event.

Why can such a flaw become systemic? Because widely used libraries are shared dependencies, and the breadth of an implementation flaw is set by the dependency graph rather than by the attacker. The tempo evidence is the concern: PHP2Shell went from leaked CVE identifiers to exploitation attempts before the patch was released, to confirmed exploitation recorded in a national exploited-vulnerability catalog within five days [3][4]. And

patching does not equal recovery: of roughly 1.1 million internet-facing PHP hosts that disclosed a version after the fix, about 758,000 ran end-of-life releases for which no official patch exists [19]. For a cryptographic defect specifically, the post-patch question – what was signed, what was accepted, and what keys were generated or exposed while the flaw was live – is a trust-recovery question rather than a patching one.

2.4 The systemic-trust pathway and what AI changes

The third pathway has the least direct 2026 evidence and the greatest potential consequence, and the source report treats it as a hypothesis stated so that it can be tested: that AI may reduce the time and expertise needed to discover, validate, chain, and deploy attacks until propagation through shared trust dependencies outpaces identification, containment, and recovery. Its defining feature is that the cryptography works. The attacker obtains valid signatures, trusted identities, or decryptions because they have obtained the ability to *use* a shared trust service for an unauthorized purpose – through a compromised administrator identity, a defective authorization service, a flawed management interface, or a vulnerability in the orchestration around the service. The distinction between stealing a key and obtaining the ability to use it is central: non-exportable keys in hardware modules mean the attacker must keep coming back to the service; they do not mean the attacker cannot sign. A signature produced this way is indistinguishable from a legitimate one, because it is one.

The 2026 evidence bears on the links of this chain unevenly. Discovery and validation are demonstrably cheaper and broader. Chaining is demonstrated in vendor exercises – a full browser exploit chain built in six days for about \$40,000 [15] – and in one attributed campaign [18]. Reliable deployment is mixed, with commentators observing that scarce in-the-wild exploitation of even catalogued critical vulnerabilities suggests attackers still struggle [2], and Google's analysis observing exploitation for only 0.23 percent of all disclosed vulnerabilities, confirming exploitation for only a handful of the 2,076 AI- and LLM-stack vulnerabilities in its study window, and finding exploitation growth driven by weaponization of known bugs rather than new discoveries [14]. The honest summary, which this paper carries forward: AI's effect is demonstrated on the first link of every chain, and is largely unmeasured on the links that determine whether a discovery becomes a systemic failure – reliable deployment, replacement-path finding, and the race between parallel compromise and parallel recovery. The scenarios that follow are constructed so that organizations can test their own side of that race: the containment and recovery links, which are governed by architecture and preparation rather than by any attacker's capability.

3. Scenario Design and the Common Template

The scenarios below cover the distinct failure classes by which AI-accelerated discovery could plausibly convert a local defect or a local compromise into a widespread loss of cryptographic protection. There is no fixed sector focus, because the determining variable is not industry but shared dependency: which libraries, hardware, services, and trust anchors many organizations rely on at once. Each scenario is described against the same template, so that they can be compared directly and used as a set in planning.

The template opens with the shared dependency and its concentration – what many organizations rely on together, and how concentrated it is – and locates the precise component and protocol layer involved, at the layer vocabulary of Section 2.1. It then names the AI-accelerated step and its 2026 evidence, tying the link of the chain that AI plausibly compresses to demonstrated capability, and states the security property lost – confidentiality, authenticity, or integrity, kept distinct, following the source report's discipline that these are different losses with different recovery. From there it traces propagation and blast radius, including what bounds the failure; the detection signals and why conventional monitoring misses them; and the recovery burden in sequence – the key rotation, revocation, re-issuance, and revalidation mechanics, in the order they must run. Each scenario closes with plausibility and historical precedent, naming the closest real events and honestly labeling gaps in the evidence, and with a short plain-language takeaway for leadership, so the executive thread runs through the technical material.

Every scenario carries the same standing label: **hypothetical, no known incident, architectural and defensive level only**. Where a scenario is modeled on real 2026 advisories or historical events, those precedents are cited as evidence for plausibility, not as instances of the scenario. The AI-accelerated step in each scenario is likewise anchored to demonstrated 2026 capability, and where the evidence for a step is weak or absent – as it is for attacker replacement-path finding – that weakness is stated rather than papered over.

4. The Scenarios

4.1 Scenario 1 – The Clone in the Chain: certificate-verification bypass in a widely embedded TLS library

Hypothetical scenario. No known incident. Modeled on the class of defect documented in wolfSSL's 2026 advisories [1]; architectural and defensive level only.

The shared dependency and its concentration. The dependency is a commercial-grade embedded TLS/SSL library licensed into thousands of products – network appliances, industrial controllers, automotive telematics, IoT devices, and packaged server software – and additionally compiled into mainstream open-source infrastructure through the vendor's own build profiles. Concentration arises twice: once through the library's product footprint, and again through build-time configuration defaults that enable a compatibility layer across many downstream builds. wolfSSL's advisory for the forged-CA-clone defect notes that the exposure widens to all CA loading under OpenSSL-compatible defaults, a setting enabled in its nginx, haproxy, Apache, stunnel, bind, ffmpeg, and distribution build profiles [1] – a concrete illustration of how one configuration flag concentrates a defect across an ecosystem.

Component and protocol layer. The component is the X.509 certificate-verification and trust-matching path of the TLS handshake: the code that decides whether the chain presented by a peer terminates at a certificate authority the client has been configured to trust. This is the implementation layer – the algorithm (RSA, ECDSA) and the construction (X.509, TLS) remain sound. The defect class is a logic error in trust matching: verification succeeds on attributes it should not, such as a certificate's asserted CA name, while ignoring the material it should compare, such as the public key. CVE-2026-93302, the real-world anchor for this scenario, is precisely such a defect, rated High, affecting wolfSSL versions 5.3.0 through 5.9.2 [1]. The same 2026 advisory set contains two sibling defects in the same verification layer: a certificate-chain verification bypass in the OpenSSL compatibility layer, in which an attacker holding *any* leaf certificate from a trusted CA could forge a certificate for any name, and an ECCSI verifier accepting forged signatures for any message for any identity [1].

The AI-accelerated step and its 2026 evidence. The compressed step is discovery: AI-assisted review finding authentication-bypass logic in verification code faster than the human review cadence that produced and maintained it. The 2026 evidence is direct – the five CVEs above were found via an AI laboratory's open-source vulnerability-finding program, whose wolfSSL submissions ran 72 of 74 reports valid [1][11], and eight more CVEs in the same library came from a security firm working with an AI laboratory [1]. The plausible extrapolation, for which the NGCC week supplies breadth evidence, is that the same review capability applied to other verification implementations would produce the same class of finding on the same tempo [6].

The security property lost. Authenticity of peer authentication, and integrity of data on connections that traverse the defective path: an active attacker can impersonate any server (or peer) to an affected client. Confidentiality is a distinct property and is not lost by the bypass itself in its certificate-verification form; a man-in-the-middle who defeats authentication can then negotiate keys with both sides, so session confidentiality follows operationally from the lost authenticity, but the cryptographic property directly lost is authentication. The pre-shared-key variant documented in CVE-2026-93304 loses authenticity with the attacker never holding the secret at all [1].

Propagation and blast radius. Propagation is set by the dependency graph: every product embedding an affected version in an affected configuration, everywhere a network attacker can obtain a man-in-the-middle position. The tail is long because embedded devices are updated on hardware replacement cycles, not days. Blast radius is bounded by version and configuration – which is itself the operational problem, since identifying which products embed which library version and build flags requires exactly the bill-of-materials most organizations lack.

Detection signals, and why they are hard to see. There is no signature-verification failure to alarm on; the handshake succeeds, the session establishes, and every mechanical test of the connection passes. Certificate transparency, the web PKI's detection machinery for unauthorized issuance, does not help for two reasons: no certificate authority was compromised – the forged certificate never needed to be issued by a real CA – and many affected clients (embedded devices, appliances) do not participate in transparency monitoring or report telemetry at all. The available signals are behavioral and external: anomalous certificates presented only to selected client populations, discovered by scanning or by the advisory itself.

Recovery burden. Sequence: patch the library and rebuild every affected product; identify which connections traversed the defective path during the exposure window, which is generally impossible retroactively since no record of rejected-or-accepted peer certificates is kept at scale; rotate any pre-shared keys distributed for configurations affected by the PSK variant; and drive a firmware-update campaign across the embedded tail.

For products that cannot be patched, compensating controls – network-level peer restrictions, tunneling affected traffic through verified paths – are the only remediation. The recovery burden is dominated not by revocation, since no CA key and no client key is compromised, but by fleet identification and rebuild.

Plausibility and historical precedent. This is the scenario with the strongest direct precedent: the anchor defects are real, were found by AI-assisted review in 2026, and were patched within days of report, with no confirmed exploitation in the wild [1]. What is hypothetical is the systemic completion – the defects arriving in a configuration-ubiquitous build, exploitation beginning before advisory, and the embedded tail remaining exposed for years. Every element of that completion has a separate precedent; their conjunction does not exist.

What this means for leadership. Your padlock icon, your VPN logo, and your compliance dashboard all reported "secure" while this class of defect was open, because nothing failed – the check simply did not run. The questions to ask your teams are: which products embed which TLS library and version, in which build configuration, and how long would it take to rebuild and redeploy all of them? If nobody can answer in hours, that is the gap.

4.2 Scenario 2 – The Certified Weak Generator: key-generation flaws in certified HSM, TPM, and secure-element firmware

Hypothetical scenario. No known incident. Modeled on ROCA (2017) [13]; architectural and defensive level only.

The shared dependency and its concentration. The dependency is cryptographic firmware – the key-generation libraries inside hardware security modules, trusted platform modules, and secure elements – supplied by a small number of silicon vendors and embedded, under those vendors' certifications, across many product manufacturers. Concentration is extreme: one vendor's key-generation library can sit inside the trusted-boot path, device identity, and code-signing infrastructure of thousands of downstream products, and in ROCA's case the affected chips were FIPS 140-2 and CC EAL 5+ certified devices shipping since at least 2012 [13].

Component and protocol layer. The component is the RSA (or elliptic-curve) key-generation routine inside certified cryptographic firmware: the code that turns entropy into a key pair. This is the implementation layer again, but with a hardware twist – the defect ships in firmware, is exercised once per key generation, and produces keys that are structurally weak from birth. ROCA's mechanism is the canonical example: generated primes carried a specific structure that made public keys factorable with a Coppersmith-type attack, requiring only the public key and no physical access, with worst-case costs reported at \$76 for a 1024-bit key and about \$40,000 for a 2048-bit key on cloud compute [13]. Notably, the ROCA researchers were explicit that the flaw did *not* depend on a weak random-number generator – all keys generated by the vulnerable chip were impacted [13], which keeps this scenario separate from the entropy scenario in Section 4.7.

The AI-accelerated step and its 2026 evidence. The compressed step is structural discovery in public data: detecting a hidden pattern in key generation by analyzing large populations of public keys, then tracing the pattern to the generating code. This is an analysis task – broad, mechanical, statistical – of exactly the kind 2026 evidence shows AI performing at scale: the NGCC review found, within days, candidate algorithms with seeds too short for their claimed security level and design-level signing-key recoveries, 110 findings from a single agentic workflow [6]; Google's measurement of AI-driven discovery breadth across the vulnerability landscape [14] supports the same extrapolation. The pattern-detection capability itself predates AI – the ROCA team found their pattern with 2017 tooling [13] – so the AI contribution is compression of discovery time from a research project to, plausibly, a routine scan.

The security property lost. Both confidentiality and authenticity, for every key the affected firmware generated: key pairs are recoverable from their public halves, so encrypted data under those keys is readable and signatures under those keys are forgeable. This is the most complete single-key loss in any scenario short of a primitive break – and, as in the quantum case, it is retroactive against everything already protected by the affected keys.

Propagation and blast radius. Every key generated by affected firmware, across every product vendor that embedded it, is exposed – and, as the ROCA researchers showed, vulnerable keys can be pre-selected in bulk from public key populations without factorization attempts, so an adversary can triage targets cheaply before spending compute [13]. Their scans confirmed on the order of 760,000 vulnerable keys in the wild, spanning citizen identity documents, authentication tokens, trusted-boot keys, package signing, and TLS [13]. Blast radius is bounded only by which firmware versions shipped where – an inventory question, again.

Detection signals, and why they are hard to see. Detection is technically possible and operationally rare. Public keys are testable for the structure – the ROCA disclosure shipped detection tools precisely because testing is cheap [13] – but nothing fails until someone factors a key: certificates remain valid, handshakes succeed, signatures verify. The organization-level barrier is provenance: knowing which of your keys were

generated on which device, in which window, under which firmware version. Most inventories do not record this, so the detection signal exists and cannot be joined to the estate.

Recovery burden. The heaviest key-recovery burden in this paper. Sequence: obtain the affected-firmware list from the vendor; test or enumerate every key whose provenance touches affected firmware; generate replacement keys – after applying the firmware fix, or off-device where the fix is unavailable, as ROCA remediation required [13]; re-certify and re-issue certificates at fleet scale; re-enroll device identities where the weak key is the device identity; and re-establish trust in anything long-lived that was signed or encrypted under the old keys. For TPM and secure-element fleets in the field, some hardware will not be re-keyable at all and requires replacement. Certification does not accelerate any of this: the ROCA chips carried FIPS 140-2 and CC EAL 5+ certificates throughout the exposure [13].

Plausibility and historical precedent. ROCA happened in full – certified hardware, structured primes, mass key recovery from public data, multi-vendor propagation [13]. The 2026 evidence adds only the discovery-speed argument [6][14]. This scenario is therefore the strongest of the eight on precedent, and its hypothetical element is narrow: an AI-assisted discovery compressing the ROCA timeline from years of expert research into a routine, repeatable scan.

What this means for leadership. Certification is not the same as assurance against this failure class – the hardware in ROCA was certified, and certified hardware shipped weak keys for five years before discovery. Two decisions matter: require that your inventory record which device generated each key and when, and ask your HSM, TPM, and secure-element vendors for their firmware key-generation review evidence – including whether AI-assisted review has been applied to it.

4.3 Scenario 3 – The Migration Window: post-quantum implementation defects found during migration

Hypothetical scenario. No known incident. Modeled on the ML-KEM-1024 decapsulation defect documented in wolfSSL's 2026 advisories and on KyberSlash (2024) [1][12]; architectural and defensive level only.

The shared dependency and its concentration. The dependency is the first generation of production implementations of the standardized post-quantum algorithms – ML-KEM above all [28] – as they are rolled into TLS, SSH, VPN, and messaging protocols during the migration now mandated for federal systems by 2030 and 2035 [20][21]. Concentration is currently moderate and rising: deployment of post-quantum key establishment is well advanced – more than half of human web traffic to one large network already used it as of Cloudflare's October 2025 measurement [42] – and the implementing ecosystem shares a small number of reference implementations and optimized code paths, so a defect in an optimized variant propagates through every library that adopted it.

Component and protocol layer. The component is the ML-KEM decapsulation routine, specifically its Fujisaki-Okamoto consistency check and its constant-time discipline, in the optimized assembly paths (AVX2, ARM64) that performance-sensitive deployments enable. The layer is implementation: the standardized construction is sound [28]. The defect class is a deterministic comparison error or a secret-dependent timing variance that turns a decapsulation endpoint into a source of information about the static private key. The real-world anchors: CVE-2026-10097, in which the AVX2-optimized ML-KEM-1024 decapsulation compared only 1,536 of 1,568 ciphertext bytes, allowing recovery of a full ML-KEM-1024 private key with approximately 98 percent success from roughly 350 chosen ciphertexts submitted to a decapsulation oracle on a static key; and its ARM64 sibling comparing half the input [1]. The KyberSlash flaws before it recovered keys within minutes from secret-dependent division timing in official reference code [12].

The AI-accelerated step and its 2026 evidence. The compressed step is differential review across the multiple implementations of one algorithm – reference code, optimized variants, ports – where a defect exists precisely because one variant diverges from the others. The 2026 evidence for AI-assisted discovery of logic defects in this very library is direct [1][11], and the NGCC week demonstrates the breadth capability across many implementations at once [6]. For honesty: the two real anchors of this scenario carry no AI attribution [1][12]; what the evidence supports is that AI-assisted review would find such defects faster, and that the defect class demonstrably exists in production post-quantum code.

The security property lost. Private-key recovery, which collapses both confidentiality – of any key establishment the static key protected – and authenticity, of any signature or derivation bound to that key. Retroactivity follows the source report's forward-secrecy discipline [25]: where the protocol used ephemeral key agreement per session, recovering a long-term key does not decrypt recorded forward-secret sessions; where a static KEM key was used with recorded ciphertexts, the recordings are decryptable. The recovery burden therefore depends on protocol configuration, which is exactly what an inventory must record.

Propagation and blast radius. Propagation follows the migration's own deployment graph: the exposed population is every service that enabled the affected optimized path on a static key reachable by submitted ciphertexts. Today that is the early-adopter population; it grows with every migration milestone [20][21]. Two features bound it: hybrid constructions, where deployed with sound combiners, preserve confidentiality as long

as the classical component also holds [27]; and the oracle precondition – an attacker able to submit chosen ciphertexts to a static-key decapsulation endpoint – which many deployments do not expose.

Detection signals, and why they are hard to see. The strongest signal is oracle telemetry: decapsulation request volume, source diversity, and error patterns against static keys, which is monitorable where the service design exposes it. The timing variant is invisible to this monitoring and requires constant-time auditing of the code. The comparison variant produces no error at all – that is the defect – so it is detectable only by differential testing between implementations, which is to say, only if someone is deliberately looking.

Recovery burden. Sequence: patch the implementation across every dependent library and product; rotate every ML-KEM key whose decapsulation endpoint was reachable during the window – which requires knowing which services exposed static-key decapsulation; assess recorded ciphertexts where static keys were used with long-lived confidentiality requirements; and re-establish sessions, which is operationally cheap. The distinctive burden of this scenario is that the rotation and assessment must run *mid-migration*, alongside the migration program itself, with the same teams and the same change windows.

Plausibility and historical precedent. The defect class is real and recent in production post-quantum code [1][12], and the migration that multiplies exposure is government-mandated and under way [20][21] – with the Government Accountability Office finding in October 2026 that none of the 24 largest federal agencies had fully implemented the three key migration practices it examined, and that only one of the 24 had fully addressed the practices for prioritized inventories of systems with vulnerable cryptography [22]. The hypothetical element is the conjunction: an AI-assisted review finding the next defect of this class in a widely deployed optimized path, before the migration completes and while a large static-key population exists.

What this means for leadership. The newest cryptography in your estate is also the youngest code in your estate, and the two must not be confused: migrating to ML-KEM is right, and the risk is in the implementation, not the standard. The decisions that matter: require hybrid deployment where the protocol supports it [27], require vendors to evidence differential testing against independent implementations, and make sure your migration plan does not create long-lived static post-quantum keys exposed to arbitrary ciphertext submission.

4.4 Scenario 4 – The Authorized Signer: abuse of the authorization plane of a software-update signing service

Hypothetical scenario, extending the source report's Section 4 systemic construction. No known incident; historical precedent in validly signed, trojanized updates [34]. Architectural and defensive level only.

The shared dependency and its concentration. The dependency is a managed software-release signing service: the service that signs the software and firmware updates a vendor ships, whose relying parties are every customer's update client. Concentration is per-vendor but total within that perimeter – one signing authority whose output is trusted by every installed instance – and the trust boundary is definable: the service accepts a build artifact and an authenticated request from a release pipeline and returns a signature that customers' systems treat as proof of genuineness. Its keys live in hardware security modules and cannot be exported; its algorithms, classical or post-quantum, remain sound throughout.

Component and protocol layer. The component is not the cryptography but the *authorization plane*: the identity layer that decides who may request a signature, the management plane that configures the service, the client libraries that mediate requests, and the open-source components all of these embed. The precondition for the scenario is an architectural one: the authorization layer comes to treat an attacker's request as coming from a legitimate release pipeline – through a phished operator credential, a management interface trusting attacker-controlled input, or a forged federated identity.

The AI-accelerated step and its 2026 evidence. This is the clearest place AI raises the probability of the scenario: reaching signing authority is an ordinary find-and-chain-software-defects problem, and that capability was demonstrated throughout 2026 – AI-assisted discovery turning a flaw judged too hard to exploit since 2024 into reliable pre-authentication code execution [2][3]; a full browser exploit chain in six days for about \$40,000 [15]; an attributed campaign chaining two public flaws to gain root on 25 servers with an AI coding agent deployed on the actors' own infrastructure [18]; AI-discovered vulnerabilities enabling remote code execution at twice the general rate [14]. The scenario's later stages – propagation and recovery – are governed by architecture that AI does not change; its earlier stage is the one the 2026 evidence bears on.

The security property lost. Authenticity of updates, and integrity of the software supply chain: artifacts the vendor never intended receive valid signatures, and update clients accept them as genuine. Confidentiality is not directly lost. The defining property of the failure, following the source report, is that the attacker never steals a key – non-exportable keys mean the attacker must keep coming back to the service; they do not mean the attacker cannot sign – so every mechanical test of the signature passes.

Propagation and blast radius. A valid signature is accepted as authentic by every client configured to trust the signer, but reach is potential exposure, not accomplished compromise: staged rollout, artifact-promotion policy, independent release approval, tenant boundaries, and client-side checks such as transparency-log presence or reproducible-build matching can each catch an unexpected build before installation. The compromise reaches relying parties for whom those controls are absent or defeated; where they are strong, an abused signing authority can be caught before installation. Once propagating, the artifact spreads at the reach of the dependency graph, carried by signatures that are by every mechanical test genuine.

Detection signals, and why they are hard to see. There is no signature-verification failure to alarm on. The signals are behavioral: an unexpected artifact in a transparency or monitoring log, an anomalous signing pattern in volume or timing, a build that does not reproduce. Two design properties decide whether these signals exist: signing that is rate-limited and observable, and logs held where a compromise of the service cannot alter them. Where the logging infrastructure is itself reachable from the environment being monitored, the primary signal is hostage to the compromise.

Recovery burden. The hardest recovery sequence in this paper, and the one established doctrine covers best on paper and worst in practice: establish the scope of what was signed during the exposure window from trustworthy historical logs; suspend the compromised signing authority without halting operations that depend on it; revoke and rotate keys and credentials; re-establish a trusted signing and distribution path that does not rely on the compromised service; redistribute new trust anchors to every relying party through channels that assumed the old anchors were good; revalidate or rebuild the artifacts signed during the window, since a patch does not un-sign a malicious release; and separate legitimate past signatures from forged ones using timestamps and revocation records. NIST SP 800-57 Part 1 Revision 5 and RFC 3647 both document this doctrine – key-compromise recovery, replacement of root trust anchors, community-wide effects, recertification of subjects [23][24]; the gap the source report identifies is operational, not doctrinal: whether it is exercised, coordinated across the dependency, and able to run at machine speed.

Plausibility and historical precedent. The precedent class is real and non-AI: trojanized updates that were built and digitally signed through the vendor's legitimate pipeline were distributed to customers in the SolarWinds intrusion, prompting an emergency federal directive [34]. The signing-service scenario is that precedent with the compromise moved earlier – from the build pipeline to the signing authorization – and with AI compressing the find-and-chain step that reaches it [2][14][15][18]. No completed instance of the AI-accelerated form exists in the public record.

What this means for leadership. Your code-signing keys being in hardware, un-exportable, and certified is necessary and not sufficient: the security of the key is the security of everything that can authorize its use. Ask three questions: who and what can request a signature – listed exhaustively; where the signing logs live – somewhere the service cannot touch; and how you would ship a trusted update if the update channel itself is suspect. If the third question has no tested answer, the recovery does not exist yet.

4.5 Scenario 5 – The Borrowed Key: cloud KMS and HSM management-plane flaws granting cross-tenant use of non-exportable keys

Hypothetical scenario. No known incident of cross-tenant mass key misuse in a major cloud KMS. Architectural and defensive level only.

The shared dependency and its concentration. The dependency is the managed key-management and HSM services of the major cloud providers, which concentrate signing and decryption authority for large tenant populations behind one control plane per provider. Relying parties are tenant applications that call the service to encrypt, decrypt, and sign. Concentration is higher than any other scenario except the CA system: a single management-plane defect can, in principle, expose operations across the provider's entire tenant base, and tenants have no alternative implementation of the dependency – the key lives where it lives.

Component and protocol layer. The component is the management and control plane: the authorization services, key policies, tenant-isolation logic, and administrative workflows that decide which principal may perform which cryptographic operation with which key. The data plane – the hardware modules holding non-exportable keys and executing operations – is exactly what its design protects, and it is not what fails. The failure is an authorization defect: cross-tenant use of a key by a principal that should never have been granted it, through a policy evaluation error, an isolation boundary flaw, or an administrative workflow that trusts attacker-controlled input.

The AI-accelerated step and its 2026 evidence. The compressed step is the same as Scenario 4's: discovery of authorization and validation defects in ordinary software at the 2026 tempo. Control planes are large, continuously modified software systems of exactly the kind AI-assisted review now examines at machine speed, and the 2026 evidence – discovery volume doubling, authentication-bypass and logic defects found in production cryptographic software, intake processes overwhelmed [1][14][17] – supports the extrapolation to the management-plane software of

key services. The nearest real-world analogue for the *effect* – a key trusted for the wrong scope because validation logic did not check it – is the 2023 Storm-0558 intrusion, in which a stolen consumer signing key was accepted for enterprise tokens because the validation path did not enforce issuer and scope, with the flaw in the checking code rather than the cryptography [29].

The security property lost. Depends on the operations the authorization defect exposed: decryption of stored ciphertexts (confidentiality of tenant data), or signing as the tenant (authenticity of whatever the tenant's key vouches for – its own artifacts, its own identities). The scenario's defining feature, shared with Scenario 4, is that the key is never stolen: the operations are performed by the service, correctly, on behalf of an unauthorized caller, so every use is a legitimate use cryptographically.

Propagation and blast radius. Set by which tenants' keys the defect made usable, and silent in operation: cross-tenant use appears in audit logs as ordinary API calls from unusual principals. Blast radius is the intersection of the defect's scope with the tenants whose data or signatures the attacker chose to use – attacker-selected, not defect-determined, which makes scoping the window after discovery genuinely difficult.

Detection signals, and why they are hard to see. The signals are audit-log analytics: cryptographic operations by principals, sources, or patterns inconsistent with the tenant's baseline. Three things make them hard. The volume of legitimate operations is enormous, so anomaly detection must be tuned. The management plane that authorizes operations frequently also fronts the logging, so a sophisticated compromise can degrade the primary signal. And tenants typically see only their own log stream, so the cross-tenant pattern – the actual signature of this scenario – is visible only to the provider, whose incentive and obligation to disclose is a contractual and regulatory question the tenant should have settled in advance.

Recovery burden. Sequence: rotate affected keys at scale; re-encrypt stored data under new keys – a compute-heavy operation whose cost scales with stored volume and which many organizations have never priced; re-issue anything the keys signed; audit every operation in the exposure window to determine what was decrypted or signed; and notify across the affected tenant population if the defect was cross-tenant. Re-encryption is the distinctive burden here: unlike certificate rotation, data encrypted under a misused key cannot be made safe by re-issuing the key – the ciphertext must be rewritten.

Plausibility and historical precedent. The component class – control-plane authorization defects in cloud services – recurs in the public record at low severity and small scope, and tenant-isolation flaws are a known finding category; no public event matches this scenario's mass cross-tenant key-use form, and it is labeled hypothetical accordingly. The effect precedent (Storm-0558's scope-validation flaw [29]) and the discovery-tempo precedent [1][14] are real. The provider-side concentration that would amplify the blast radius is a documented structural property of cloud key services, and it is what makes this scenario worth planning for despite the absence of precedent.

What this means for leadership. "Non-exportable" answers the question "can the key be stolen?" and not the question "can the key be used by the wrong party?" – and the second question is the one that matters here. Ask your providers: what authorization-plane controls and audits exist around key use; where audit logs are written and who can alter them; what their cross-tenant blast-radius disclosure commitments are; and then price the re-encryption of the data you keep under their keys, because that number is your real exposure.

4.6 Scenario 6 – The Minted Identity: token forgery in an identity-federation provider

Hypothetical scenario. No known incident of exactly this form; effect precedent in the 2023 Storm-0558 forged-token intrusion [29]. Architectural and defensive level only.

The shared dependency and its concentration. The dependency is an identity-federation provider – the single token issuer whose assertions many applications trust, inside one organization or across an entire SaaS ecosystem. Relying parties are every application that accepts the provider's tokens. Concentration is the point of federation: one issuer, many relying parties, so a defect or compromise at the issuer converts one local failure into authentication failure across the whole relying population.

Component and protocol layer. The component is the token issuance and validation path: the signing keys that vouch for tokens (SAML assertions or OIDC tokens), the issuance logic that decides which claims are true of which subject, and – distributed across every relying party – the validation logic that decides what a token may be trusted for. The failure forms are two: issuance under an attacker's influence, or validation accepting a token outside the scope it was issued for. The Storm-0558 precedent is the second form: forged authentication tokens were accepted for enterprise mail because the validation path did not enforce issuer and scope against the key that signed [29]. The wolfSSL record shows the first form's defect class in another domain: verification code accepting assertions it should reject, across identity, chain, and message verification [1].

The AI-accelerated step and its 2026 evidence. The compressed step is, again, discovery: finding validation-logic and authorization defects in the issuer's and relying parties' software at machine tempo. The 2026 evidence is the same discovery record [1][11][14], with the specific observation that the most consequential real-world token-forgery event on record rested on a validation flaw in ordinary code [29] – the exact class of defect AI-assisted review demonstrably surfaces in verification paths.

The security property lost. Authentication – who is present – and the authorization derived from it. Confidentiality of data is not lost cryptographically: the attacker is granted access, and the application serves data exactly as it would to the legitimate user. This distinction matters operationally: there is no key to rotate to make past access un-happen, only sessions to invalidate and access to audit.

Propagation and blast radius. Every application accepting the issuer's tokens, for as long as the flaw or compromise is open, with propagation silent because the tokens are cryptographically valid: sessions establish, audit trails record logins that look legitimate, and nothing fails. Blast radius is bounded by the token scopes the defect made forgeable and by which relying parties enforced their own checks – a relying party that independently verifies audience and scope narrows the damage, exactly as the client-side verification in Scenario 4 does.

Detection signals, and why they are hard to see. The signals are mismatch analytics: issuance records that do not correspond to authentication events; use patterns impossible for the legitimate subject – geography, velocity, device; tokens presented with claim sets the issuer never issues; and reports from relying parties of tokens that fail secondary checks. The difficulty is that the mismatch lives across two systems – issuer logs and relying-party logs – that many organizations do not join, and that the token itself, being valid, raises no alarm anywhere.

Recovery burden. Sequence: rotate the issuer's token-signing keys; invalidate sessions and force re-authentication across the entire relying population – an operation whose scale is set by every active session everywhere; audit the issuance window to identify forged-token use; and, hardest of all, recover the recovery path: responders and administrators typically authenticate through the very federation being repaired, so break-glass credentials held independently of the provider are not a convenience but the precondition for responding at all.

Plausibility and historical precedent. Storm-0558 is a real, high-consequence instance of forged tokens accepted through a validation flaw, affecting about 25 organizations including government agencies [29]. The 2026 discovery record supplies the tempo evidence [1][14]. The hypothetical element is the conjunction with AI-accelerated discovery and a wider relying population than any recorded incident.

What this means for leadership. Your identity provider is cryptographic infrastructure, whatever the label on the contract says. Three questions: are its token-signing keys, key-rotation history, and validation logic in your cryptographic inventory; do your applications enforce their own audience and scope checks, or do they trust the token wholesale; and – the one many organizations fail – can your administrators log in when the identity provider is the thing that is broken?

4.7 Scenario 7 – The Tainted Well: entropy and RNG defects in common libraries or operating systems

Hypothetical scenario. No known incident at the stated scale; strong historical precedent in the 2008 Debian OpenSSL flaw [32] and the 2012 widespread weak-key findings [31]. Architectural and defensive level only.

The shared dependency and its concentration. The dependency is the entropy path itself: the operating system's random-number generator and the seeding logic of the cryptographic libraries that consume it. Concentration approaches the theoretical maximum – a single defect in one OS release or library version can affect every key generated by every system running it, with no respect for vendor, sector, or algorithm, because the weakness is upstream of all of them.

Component and protocol layer. The component is the entropy source and the deterministic random-bit generator seeded from it, as consumed at key-generation time. The failure forms are a defective entropy path in a library (the 2008 Debian OpenSSL flaw, in which a change to the RNG made keys predictable for two years of releases, requiring regeneration of all keys created on affected systems [32]) and an entropy-starved state at generation time – the boot-time entropy hole documented for headless and embedded devices, where long-term keys are generated before the kernel has accumulated real entropy [31].

The AI-accelerated step and its 2026 evidence. The compressed step is detection in public data at scale: weak or repeated keys are discoverable by analyzing large populations of public keys – the 2012 scan computed pairwise GCDs across 11,170,883 distinct RSA moduli and recovered private keys for 0.50 percent of TLS hosts and 1.03 percent of SSH hosts examined, the SSH share concentrated in DSA host keys, with nearly all affected devices embedded or headless units from dozens of manufacturers [31] – and that analysis is mechanical, parallel, and exactly the kind of bulk pattern-finding the 2026 evidence shows AI performing: the NGCC review flagged seeds too short for claimed security within days [6], and discovery breadth across implementations is measured across the landscape [14].

The security property lost. Both confidentiality and authenticity, for every key generated in the weak state, because predictable keys break both at once. Retroactivity is total and silent: everything ever protected by an affected key – every ciphertext, every signature, every session – is exposed from the moment of generation, without any event occurring.

Propagation and blast radius. Set by the defect's distribution and the generation window. A library defect propagates with the software release, so the affected population is every system that ran it during the window – for the Debian flaw, approximately two years of keys [32]. A boot-time entropy defect propagates with hardware type and image, so it clusters in fleets of identical embedded devices generating keys at first boot [31]. In both forms, nothing about the affected systems looks different until keys are actually recovered.

Detection signals, and why they are hard to see. Public-key population analysis is the detection mechanism, and it is real: shared factors, repeated keys, and statistical anomalies in public keys are testable in bulk [31]. The organizational barrier is provenance again – the detection result ("this class of keys is weak") must be joined to the estate ("which of our keys belong to that class"), and the join requires knowing which keys were generated where, when, on what – a record few organizations keep. Internally, repeated-key detection across the estate's own certificate and key stores is possible and almost never run.

Recovery burden. Sequence: enumerate the affected keys, which requires the provenance records; regenerate them; re-issue and re-distribute the associated certificates, host keys, and device identities; re-enroll devices whose weak keys are their born-in identity – which for some hardware means replacement, as with Scenario 2; and assess everything encrypted or signed under affected keys. The distinctive burden is the enumeration: a two-year generation window across a distributed estate, with no boundary event to date it, makes "which keys" the dominant cost of the whole recovery.

Plausibility and historical precedent. Both precedent events are real and non-AI: the Debian flaw [32] and the boot-time entropy population findings [31]. The 2026 evidence adds machine-speed detection of exactly the statistical signature involved [6][14]. The hypothetical element is the combination of a wide distribution with AI-accelerated discovery of it – plausible in direct proportion to how much public-key analysis an adversary can now automate.

What this means for leadership. This scenario fails without any event occurring: no breach, no advisory, nothing on any dashboard – until keys start falling. The one decision that changes your exposure is provenance: record where and when every key was generated, on what OS and library version, and you convert an unbounded discovery problem into a bounded enumeration. If your key inventory lacks a generation-provenance field, you have no defense against this scenario at all.

4.8 Scenario 8 – The Root of the Root: certificate authorities, transparency logs, and recovery paths, including the Merkle Tree Certificate transition

Hypothetical scenario. No known incident in the post-quantum era; historical precedent in the 2011 DigiNotar certificate-authority compromise [30]. Architectural and defensive level only.

The shared dependency and its concentration. The dependency is the public certificate-authority system and its transparency machinery, trusted by every browser and TLS client at once, together with the recovery mechanisms – escrowed keys, break-glass paths, out-of-band update channels – that would be used to repair any of it. Concentration is the highest of any dependency in this paper: a handful of authorities are trusted by every client, so one authority's issuance compromise propagates to every relying party. The transition now under way adds a new surface: Let's Encrypt has committed to Merkle Tree Certificates as its path to post-quantum-safe web PKI, with certificates issued in batches under a single signature per batch, transparency built into issuance rather than bolted on, staging targeted for late 2026 and production in 2027 [33] – machinery whose failure modes are necessarily less exercised than the system it replaces.

Component and protocol layer. The components are the CA's issuance systems – the domain-validation automation, portals, and operator identity infrastructure that decide whether a certificate should be issued – and the transparency-log and aggregation infrastructure that makes issuance public. In the Merkle Tree Certificate design, the batch-signing "landmark" path and the publication/mirroring path carry the corresponding trust. The failure is at the periphery of the cryptography: the issuance decision is ordinary enterprise software, and its compromise produces fraudulent certificates that are, from every cryptographic viewpoint, perfectly issued. The DigiNotar precedent is exact in shape: attackers reached the CA's issuance systems through its periphery, generated at least 531 rogue certificates for 140 distinct names, and a rogue wildcard certificate for a major web service was used in a man-in-the-middle interception of roughly 300,000 users, the overwhelming majority in Iran, before the fraud was detected publicly; the authority's certificates were distrusted globally and it entered bankruptcy within weeks [30].

The AI-accelerated step and its 2026 evidence. The compressed step is compromise of the CA periphery – the portals, validation automation, and operator identity paths that are conventional web and enterprise software – through find-and-chain at the 2026 tempo [2][14][15][18]. The evidence for the tempo is direct; the CA-specific application of it is the hypothetical extrapolation.

The security property lost. Authenticity of the web PKI: certificates for names their owners never requested, trusted by every client. Where interception follows, confidentiality of the intercepted sessions is lost as well, as in the DigiNotar interception [30] – the same authentication-to-confidentiality cascade as Scenario 1, at infrastructure scale.

Propagation and blast radius. Every client that trusts the authority, bounded by the transparency machinery that detects unauthorized issuance – for the web PKI, where certificate transparency is enforced, and not for the private and enterprise PKI that trusts the same or similar authorities without it. Recovery in the web PKI has a worked precedent: global distrust of the compromised authority, mass re-issuance by other authorities, and browser updates shipped on emergency timelines [30]. The harder propagation cases are the non-web populations – enterprise roots, device firmware trust stores, private CAs – that lack transparency monitoring entirely.

Detection signals, and why they are hard to see. Certificate transparency is the signal: unauthorized issuance for monitored domains appears in public logs, and domain owners who monitor them can catch fraud before interception. The weaknesses are coverage and timing: DigiNotar's rogue certificates went undetected for weeks, with the authority's own tampered logs hiding the issuance [30]; monitoring is only as good as the domain owner's participation; and during the Merkle Tree transition, the monitoring population itself is split between old and new machinery, with the new aggregation and mirroring paths unexercised at incident scale [33].

Recovery burden. For the web PKI: distrust the authority, re-issue affected certificates through others, ship client trust-store updates, and use timestamps and revocation records to sort legitimate from fraudulent certificates – machinery that exists and has run. For private and device PKI: rotate enterprise roots and redistribute trust anchors through firmware and management channels that assume the old anchors were good – the recovery-runs-through-the-compromised-thing problem of Scenario 4. And across all of it: verify that the recovery paths themselves – escrow, break-glass credentials, out-of-band channels – are independent of the infrastructure being recovered, which the source report names as the most frequently overlooked property in the entire trust landscape, precisely because those paths are exercised least and are often reachable from the environment they are meant to repair.

Plausibility and historical precedent. DigiNotar is a complete, catastrophic instance of the failure [30]; repeated authority-distrust events since have kept the machinery exercised. The Merkle Tree Certificate transition is real, scheduled, and explicitly designed to change the failure modes of certificate transparency [33]. The hypothetical element is the AI-tempo periphery compromise against an authority, coinciding with the transition's unexercised monitoring paths.

What this means for leadership. Three practical exposures hide behind "we use TLS": your organization's own domains (are you monitoring certificate transparency for them – the cheapest detection you can buy); your private PKI and device trust stores (which have no transparency log and inherit this failure mode silently); and your recovery paths (if your break-glass and out-of-band mechanisms are reachable from your ordinary environment, they will fail exactly when needed).

5. The Scenarios Compared with the Quantum Threat

The source report compares the AI and quantum threats along explicit dimensions rather than by asserting that one is "as big as" the other, and the discipline is worth repeating here: breadth of mathematical vulnerability is not breadth of practical exploitation; the two threats are not distinguished by *which* security properties they can destroy – a quantum computer that recovers a signing key forges software signatures exactly as an abused signing service does; and retroactivity is a property of attacking key establishment or key generation, not a property unique to quantum computing. The table below ranks the eight scenarios against a quantum (Shor) break on the source report's Section 5.2 dimensions. Each cell is compressed; the scenario sections carry the detail.

Scenario	Breadth	Property lost	Attacker needs	Economics	Persistence after fix	Retroactive exposure	Recovery
Quantum (Shor) break	RSA, DH, ECC as families; per-key exposure still parameter- and resource-dependent	Confidentiality and authentication for public-key uses	A fault-tolerant quantum computer – does not exist [20]	Nation-states, years after first demonstration	Migration closes new exposure; recorded traffic stays exposed	Recorded key exchanges, wholesale	Replace algorithms across the estate, on schedule
1. Clone in the Chain (library verification bypass)	Products and builds embedding the defective version and configuration	Authenticity; integrity on the path; confidentiality cascades operationally	A network position; knowledge of client trust configuration	Ordinary actors once found; discovery cheap in 2026 [1]	Patch closes future exposure; sessions in the window stay suspect	Only sessions traversed during the window	Patch and rebuild fleet; rotate PSKs; embedded tail persists
2. Certified Weak Generator (keygen flaw in certified hardware)	Every key from affected firmware, across all embedding vendors	Both confidentiality and authenticity, per key	Public keys only; no physical access [13]	Cloud-scale compute for bulk targets [13]	New keys are sound; all keys generated before the fix stay recoverable	Total, from generation onward	Re-key fleet, re-certify, re-enroll device identity; some hardware unrecoverable
3. Migration Window (PQ implementation defect)	Services enabling the affected optimized path on exposed static keys	Key recovery: confidentiality of KEM outputs; authenticity bound to the key	A reachable decapsulation oracle or timing measurement [1][12]	Ordinary actors once found	Patch closes it; keys recovered in the window are gone; recorded static-key ciphertexts exposed	Recorded sessions under static keys, not forward-secret ones [25]	Patch mid-migration, rotate KEM keys, assess recordings
4. Authorized Signer	Every relying party lacking	Authenticity of updates;	Authorization to use the	Well-resourced	Revocation and re-	Past signatures	Revoke, rotate, re-

Scenario	Breadth	Property lost	Attacker needs	Economics	Persistence after fix	Retroactive exposure	Recovery
(update-signing authorization abuse)	independent client-side verification	integrity of the supply chain	signing service, reached through its periphery	actors; concentrated payoff	rooting close it; artifacts signed in the window persist	need revalidation	root, redistribute, revalidate – hardest case [23][24]
5. Borrowed Key (cloud KMS management-plane flaw)	Tenants whose keys the defect made usable	Confidentiality of decrypted data and/or authenticity of tenant signatures	A management-plane authorization defect and a foothold	Well-resourced actors; per-provider concentration	Key rotation closes it; data decrypted in the window stays exposed	Everything decrypted or signed in the window	Rotate keys, re-encrypt stored data , audit window operations
6. Minted Identity (federation token forgery)	Every application accepting the issuer's tokens	Authentication; downstream authorization; confidentiality via granted access	A validation flaw or issuer compromise [29]	Ordinary actors once found	Key rotation and session invalidation close it; past access cannot be un-granted	Access in the window, auditable only from logs	Rotate issuer keys, invalidate sessions fleet-wide, audit issuance window
7. Tainted Well (entropy defect)	Every key generated in the weak state, across all vendors and algorithms	Both confidentiality and authenticity, per key	Public-key population analysis [31]	Bulk analysis, cheap relative to payoff	New keys are sound; every key from the window stays recoverable	Total, silent, from generation	Enumerate by provenance, regenerate, re-issue, re-enroll; enumeration dominates
8. Root of the Root (CA / transparency / recovery paths)	Every client trusting the authority; private PKI lacks the transparency bound	Authenticity of the PKI; confidentiality via interception [30]	CA periphery compromise at machine tempo	Well-resourced actors; highest concentration in the paper	Distrust and re-issuance close it; fraudulent certificates made in the window persist	Interception and fraud in the window	Distrust, mass re-issuance, redistribute trust anchors; verify recovery-path independence

Read along the dimensions, three rankings fall out, and they differ by dimension in the way the source report warns they must.

On **speed and warning time**, every scenario outranks the quantum threat, which is the central practical claim of this paper. A quantum break requires hardware that does not exist and whose arrival no peer-reviewed or government estimate dates [20]; each scenario requires only demonstrated 2026 discovery capability applied to a shared dependency [1][2][6][14]. The scenarios with the shortest distance from discovery to systemic effect are 1, 3, and 6, whose anchors are real 2026 defects in exactly the relevant layer, needing only the completion of propagation.

On **breadth of exposure per event**, the ranking runs 8, 5, 4 (infrastructure-wide, bounded only by independent client-side controls), then 7 and 2 (every affected key, bounded by generation provenance), then 1, 3, 6 (bounded by version, configuration, or token scope). Only a full quantum break exceeds all of them at once – but 2 and 7 match its per-key totality for the affected populations, and 2's precedent (ROCA) demonstrates that the affected population can reach hundreds of thousands of keys through certified hardware alone [13].

On **retroactivity**, the quantum threat's most-feared property – decryption of recorded traffic – is matched within their populations by 2 and 7 (every affected key, from generation onward) and partially by 3 and 5 (recorded ciphertexts and decrypted data within the window), while the authenticity-side scenarios (1, 4, 6, 8) require instead the retrospective sorting of legitimate from forged artifacts, which demands trustworthy timestamps, revocation records, and transparency logs. Retroactivity, in other words, is a property of what was attacked, not of what attacked it – which is why the provenance and history-validation capabilities in Section 6 serve both threats at once.

On **recovery difficulty**, the ranking is nearly the reverse of breadth: 4, 5, and 8 are hardest because recovery must run without the compromised service and, in 5, because ciphertext must be rewritten rather than re-keyed; 2 and 7 are next, dominated by fleet enumeration; 1, 3, and 6 are bounded by patching plus rotation. A scheduled quantum migration is, by comparison, a planned change under control – the source report's most operationally significant observation is that the recovery doctrine for trust failures exists in NIST SP 800-57 and RFC 3647 [23][24] but is not carried into the migration programs, and is largely unexercised [22].

6. Defenses and Recovery Capabilities by Scenario

The source report's Section 6 separates the actions a post-quantum migration already requires from the additional capabilities a systemic trust failure would demand, and warns that the single word "agility" conceals several distinct operations: algorithm replacement, parameter change, implementation repair, key replacement, and restoration of trust are not the same capability, and strength at one does not transfer. This section maps those capabilities to the scenarios, then adds the scenario-specific controls that the template analysis surfaced.

Inventory extended to authorization, dependency, and provenance. The cryptographic bill of materials – algorithm and parameters, implementing library and version, key lifetime, confidentiality lifetime, protocol, and named owner – is the foundation every scenario leans on; federal agencies are required to prioritize such inventories under Executive Order 14412, and the Government Accountability Office found that only one of 24 large agencies had fully addressed the inventory practices it examined [21][22]. The three extensions this paper's analysis adds: record **which services and identities can authorize a signing or decryption operation** (the perimeter of Scenarios 4, 5, and 6); record **shared and concentrated dependencies** – which authorities, update signers, key managers, identity providers, libraries, and firmware trust anchors the organization relies on (Scenarios 1, 4, 5, 6, 8); and record **key provenance** – where, when, and on what each key was generated (Scenarios 2 and 7, where it is the difference between a bounded enumeration and an unbounded one).

Migration discipline and parameter agility. Migration should continue on its mandated schedule [20][21]; AI uncertainty strengthens the assurance around it rather than pausing it. The scenario-specific content: prefer finalized standards – ML-KEM, ML-DSA, SLH-DSA [28][43][44] – over candidates in evaluation; extend agility to parameters, since the likeliest cryptanalytic outcome is margin erosion forcing mid-deployment parameter change; and do not confuse agility with unrestricted runtime selection, which is a downgrade surface – NIST's crypto-agility guidance frames the goal as replacement and adaptation *while preserving security and ongoing operations* [26].

Hybrid constructions with the properties named. Hybrid key establishment and hybrid authentication are separate properties: a hybrid KEM does not make a connection's authentication hybrid, combination security depends on the combiner and attacker model, and organizations must use reviewed, standardized combinations rather than inventing their own [27]. For Scenario 3, hybrid deployment is the single most effective structural control – it preserves confidentiality as long as the classical component holds, bounding the PQ defect's blast radius [27] – and its limits should be stated with equal precision: a hybrid does nothing about the shared implementation, entropy, authorization, and key-management failures of Scenarios 4, 5, and 7, and a compromised signing service signs with whatever algorithms it holds.

Cryptographic software as a critical asset class. Cryptographic libraries, TLS and SSH stacks, certificate-validation code, and key-management software should carry the shortest patch service levels in the organization, with intake adapted to AI-generated submission volume – AI-assisted triage, and immediate patching of easily discoverable defects rather than embargo [1][14][17]. AI-generated patches themselves require independent validation before deployment: one controlled study found only 26 percent of AI patches remediated the flaw without introducing a regression, and crash-suppression testing inflated measured success by 1.83 times [35][36] – for a cryptographic library, validation should include differential testing against an independent implementation, which is also the standing control for Scenarios 3 and 7.

Restoration of trust as a distinct, rehearsed capability. The capability list from the source report, which the scenarios above exercise in different combinations: map cryptographic authorization; identify affected versions and configurations within hours of an advisory; revoke and rotate at scale, knowing in advance how long it takes; re-establish trusted signing, validation, and distribution paths independent of a compromised service; rebuild or revalidate affected artifacts, since a patch does not un-sign a malicious release; determine which historical transactions require review using timestamps, revocation records, and transparency logs; maintain protected recovery mechanisms with verified independence; and exercise the whole sequence with the clock running [23][24].

The table below compresses the mapping.

Scenario	Primary preventive controls	Primary recovery capabilities
1. Clone in the Chain	Library-and-version CBOM; build-configuration records; verification-path testing in acceptance	Fleet identification in hours; rebuild and redeploy; PSK rotation; compensating controls for the unpatchable tail
2. Certified Weak	Key-provenance inventory; vendor firmware assurance evidence; public-key testing at issuance and enrollment	Bulk key enumeration by provenance; off-device or post-fix re-key at fleet scale; re-certification and

Scenario	Primary preventive controls	Primary recovery capabilities
Generator		device re-enrollment
3. Migration Window	Hybrid deployment where the protocol supports it [27]; differential testing across implementations; no long-lived static KEM keys exposed to arbitrary submission	Patch mid-migration; rotate KEM keys; assess recorded ciphertexts under static keys; re-establish sessions
4. Authorized Signer	Authorization-perimeter inventory; phishing-resistant authentication for signing principals; artifact-and-purpose-bound authorization; rate-limited, observable signing; independent log custody	Scope-from-logs; suspend authority without halting operations; revoke and rotate; out-of-band redistribution of trust anchors; artifact revalidation
5. Borrowed Key	Provider authorization-plane assurance evidence; independent log sinkage; contractual cross-tenant disclosure terms	Key rotation at scale; priced re-encryption of stored data; window-operation audit; tenant notification cascade
6. Minted Identity	Issuer keys and validation logic in the CBOM; relying-party audience/scope enforcement; issuance-to-use analytics; break-glass credentials held independently	Issuer key rotation; fleet-wide session invalidation and forced re-authentication; issuance-window audit; responder access independent of the provider
7. Tainted Well	Key-provenance inventory; repeated-key and statistical checks across the estate's own key stores; entropy-path validation for embedded fleets	Provenance-bounded enumeration; regeneration and re-issuance across the generation window; device re-enrollment
8. Root of the Root	Certificate-transparency monitoring for owned domains; private-PKI transparency equivalents; recovery-path independence verification	Authority distrust and mass re-issuance; trust-anchor redistribution through firmware and management channels; historical sorting via timestamps and revocation records

7. Tabletop Exercise Injects

The source report proposes a CSA tabletop exercise whose premise is deliberately the one migration planning does not cover: a widely trusted signing or key-management service is compromised while its underlying algorithms remain sound, and the exercise is about containment and recovery, not about how the compromise occurred. The exercise tests six capabilities – identifying exposure, exercising authority, distributing replacement credentials and software, validating history, communicating confirmed scope and residual uncertainty, and measuring the time to contain and restore. The injects below extend that design so the exercise covers the full scenario set of this paper, each inject drawn from one scenario and each targeting one or more of the six capabilities. They are offered in the same spirit as the source proposal: as a bounded addition to CSA's existing exercise work, not a new program, and with the standing note that defender-side intake is itself under machine-speed pressure – one laboratory's public disclosure ledger showed 29,439 vulnerability reports received between November 2025 and October 2026, of which 6,157 became disclosed findings and 516 were patched, a reminder that triage capacity is part of the recovery race [16].

Inject (announced mid-exercise)	Drawn from	Capability tested
While the first advisory is being scoped, a second AI-attributed advisory lands against a different cryptographic library embedded in the same product line	Scenario 1	Identify exposure – inventory depth and parallel scoping under multiple simultaneous advisories
Thirty percent of affected devices run firmware versions for which no patch will be released	Scenario 1	Identify exposure and communicate – compensating controls, replacement budget, and honest statements about the unpatchable tail
The affected-firmware list is uncertain, and the organization's key inventory has no field recording where or when keys were generated	Scenarios 2, 7	Identify exposure – key-provenance enumeration under uncertainty, the dominant recovery cost of both scenarios
The vulnerable-key population is found to include the hardware-backed code-signing root	Scenario 2	Exercise authority – suspending signing without halting releases; trust-anchor rotation and re-enrollment
The only channel able to distribute new trust anchors at fleet scale is the update service under investigation	Scenarios 4, 8	Distribute replacement – out-of-band path readiness, the hardest requirement of the systemic pathway
Administrator authentication for the responders themselves runs through the identity provider under investigation	Scenario 6	Distribute replacement and exercise authority – break-glass credentials held independently, verified before the incident
Leadership asks, mid-exercise, how long and at what cost the affected encrypted data stores can be re-encrypted	Scenario 5	Measure – whether re-encryption, the scenario's dominant cost, has ever been priced
Fraudulent certificates are discovered for the organization's own domains, and its private PKI has no transparency log	Scenario 8	Validate history and identify exposure – transparency monitoring, mass re-issuance, sorting legitimate from fraudulent issuance
As the first revocation campaign completes, validly signed artifacts appear from a signing path not previously known to the organization	The source report's §3.4 hypothesis	The unmeasured link – attacker replacement-path finding; tests whether re-entry detection exists at all
A regulator demands a public statement while scope is only partially known	All scenarios	Communicate – separating confirmed scope from residual uncertainty without over- or under-stating either

Two design notes for exercise owners. First, the ninth inject tests the specific proposition the 2026 evidence leaves open – that a model which found one authorization gap can be rerun to find the next, compressing the attacker's re-entry time below the defender's recovery time – and it should be scored honestly: many organizations will find they have no re-entry detection, which is the finding the exercise exists to produce. Second, the exercise's output should be measured, not narrated: time to identify exposure, time to exercise authority, time to distribute replacement trust, and time to restore, per scenario – because a recovery capability that has never been run with the clock on is a plan, not a capability.

8. Conclusions and Recommendations

Could AI cause a systemic loss of cryptographic protection before a quantum computer can break public-key cryptography? The scenarios in this paper support the same three-part answer the source analysis reaches, made concrete across eight failure classes. First, no practical cryptanalytic break of a deployed standard by AI has been demonstrated, and neither its probability nor timing can responsibly be estimated – the 2026 cryptanalytic record is one original result against a candidate scheme, engineering leverage against legacy parameters, and breadth against unreviewed competition fields [6][7][8][9][10]. Second, the pathways with real evidence behind them are the implementation pathway – where AI-assisted review produced authentication bypasses and forgery conditions in a production TLS library within a single year [1][11] – and, as a hypothesis warranting preparation, the systemic trust pathway, where sound cryptography is turned against those who rely on it through shared signing, identity, and key-management dependencies. Third, what distinguishes the AI scenarios from the quantum threat is not the security properties at risk but the shape of the risk: the quantum threat needs hardware that does not exist and arrives on a public schedule with a mandated migration attached [20][21]; the scenarios here need nothing that does not already exist and have no schedule at all. The honest limits should travel with the conclusion: discovery is the demonstrated link, reliable deployment and replacement-path finding are not, only a handful of the 2,076 AI-stack vulnerabilities in Google's measurement were confirmed exploited at all [14], and the intake figures that alarm defenders measure load, not fix rates [16]. The scenarios are preparation instruments, not predictions.

The recommendations reduce to the eight leadership decisions of the Executive Summary, restated here in the vocabulary the scenarios gave them. Extend the cryptographic inventory to authorization, dependency, and provenance – the three fields that decide whether Scenarios 2, 4, 5, 6, and 7 are a bounded recovery or an unbounded one. Treat cryptographic software as a critical asset class on the patch clock AI has set, with AI-assisted triage adapted for intake and independent validation – including differential testing – for patches [1][14][16][35][36]. Continue the post-quantum migration on its mandated schedule, preferring finalized standards [28][43][44], extending agility to parameters [26], deploying hybrid constructions where the protocol supports them and naming precisely what they do and do not protect [27], and refusing to create long-lived static post-quantum keys exposed to arbitrary ciphertext submission. Build revocation, rotation, re-rooting, redistribution, and revalidation as measured, rehearsed capabilities – doctrine that already exists in NIST SP 800-57 and RFC 3647 [23][24] and that migration programs do not carry [22]. Verify the independence of recovery paths rather than assuming it. Monitor certificate transparency for owned domains, and treat private PKI and device trust stores as carrying the same failure mode without the detection machinery. And run the tabletop exercise of Section 7 with the clock on, because the measure of resilience against every scenario in this paper is not whether the cryptography can be replaced but how quickly trust can be revoked, re-rooted, and redistributed – a capability built and exercised in advance or not possessed at all.

CSA Resource Alignment

This paper's framing – that concentration in shared dependencies, not mathematics, is what turns a local cryptographic failure into a systemic one – aligns most directly with two recent CSA artifacts surfaced for this topic. CSA's research note *FortiBleed: Anatomy of a 110-Million-Credential Harvesting Campaign* analyzes how a campaign against one widely deployed line of perimeter devices, beginning from a handful of footholds, scaled to an ecosystem-level event spanning tens of thousands of compromised systems – and its remediation guidance (session termination, credential rotation across the exposure window, phishing-resistant authentication, and removal of management interfaces from internet exposure) is the operational counterpart to the dependency-mapping this paper recommends. The blast radius of every scenario in Section 4 is set by exactly the concentration dynamic that note documents [38]. CSA's *Agents in the Wire: AI Agents as Enterprise Insider Threats* addresses the complementary propagation question: its "toxic agent flow" pattern documents how a single compromised agent cascades through trusting peers into an enterprise-wide threat by way of ordinary, authorized interaction channels, and its guidance maps onto the propagation and containment analysis of Sections 4.4 through 4.8, where the compounding variable is trust that is granted once and inherited everywhere [37]. On the preparedness side, the *Frontier Ready Assessment Framework* measures organizational readiness for AI-speed vulnerability operations, including the patch-tempo and recovery capabilities that Section 6 identifies as decisive across all eight scenarios [39]. For control mapping, the *AI Controls Matrix v1.1* provides the cryptography, key-management, and vulnerability-management control objectives against which the inventory, agility, and trust-restoration practices recommended here can be assessed [40]. The source analysis this paper extends, *Before Quantum: AI and the Risk of Systemic Cryptographic Failure*, is a companion document in the same research thread, not yet publicly released at the time of writing; its own published predecessor, the research note *AI Cryptanalysis and the Cryptographic Assumption Horizon*, first placed the 2026 cryptanalytic results beside the quantum resource estimates [41].

References

- [1] wolfSSL. "[Security vulnerabilities](#)." wolfSSL, retrieved 29 October 2026.
- [2] Wiz Research. "[PHP2Shell: pre-authentication remote code execution in PHP](#)." Wiz, 21 October 2026.
- [3] The PHP Foundation. "[Today's PHP security release and how we're strengthening PHP security](#)." The PHP Foundation, 21 October 2026.
- [4] CISA. "[Critical vulnerabilities in PHP require immediate patching](#)." CISA, 21 October 2026.
- [5] SecurityWeek. "[Node2Shell Brotli flaw triggers patching across Microsoft, Node.js, Python packages](#)." SecurityWeek, 26 October 2026.
- [6] M.-J. O. Saarinen. "[Chinese NGCC algorithms: the first week of AI cryptanalysis](#)." IACR ePrint 2026/2266, 30 September 2026.
- [7] Anthropic. "[Discovering cryptographic weaknesses with Claude](#)." Anthropic, 28 July 2026.
- [8] Z. Straznickas and S. Weis. "[HAWK-n key recovery reduces to SVP in dimension \$n/2+1\$](#) ." IACR ePrint 2026/1593, 3 August 2026.
- [9] E. Lu (Cognition). "[Factoring RSA-260](#)." Cognition, 9 September 2026.
- [10] S. A. Weis. "[Factoring RSA-896](#)." saweis.net, 19 September 2026.
- [11] Anthropic. "[Launching an opt-in vulnerability-finding service for open source](#)." Anthropic, 8 October 2026.
- [12] D. J. Bernstein et al. "[KyberSlash: exploiting secret-dependent division timings in Kyber implementations](#)." IACR TCHES 2025(2), 2025.
- [13] M. Nemec, M. Sys, P. Svenda, D. Klinec, V. Matyas. "[The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli \(ROCA; CVE-2017-15361\)](#)." ACM CCS 2017; CROCS, Masaryk University, 2017.
- [14] Google Threat Intelligence Group. "[Vulnerability discovery and exploitation trends in the AI era](#)." Google Cloud, 30 September 2026.
- [15] XBOW. "[XBOW built a Chrome full-chain exploit](#)." XBOW, 13 October 2026.
- [16] Anthropic Frontier Red Team. "[Coordinated vulnerability disclosure dashboard](#)." Anthropic, snapshot 2 October 2026.
- [17] Help Net Security. "[AI slop submissions force Google to freeze its open-source bug bounty](#)." Help Net Security, 5 October 2026.
- [18] Dream Group. "[Chained PHP2Shell exploit roots 25 servers, including US government servers](#)." Dream Group, 26 October 2026, updated 29 October 2026.
- [19] VirusTotal. "[Sizing the PHP exposure window for CVE-2026-109618/109619](#)." VirusTotal, 21 October 2026.
- [20] D. Moody et al. "[Transition to Post-Quantum Cryptography Standards \(NIST IR 8547, final\)](#)." NIST, 27 October 2026.
- [21] The White House. "[Executive Order 14412: Securing the Nation Against Advanced Cryptographic Attacks](#)." Federal Register, 91 FR 38484–38486, 25 June 2026.
- [22] US Government Accountability Office. "[GAO-27-108740: Federal agencies' progress toward post-quantum cryptography](#)." GAO, October 2026.
- [23] E. Barker (NIST). "[SP 800-57 Part 1 Revision 5: Recommendation for Key Management \(\\$5.5 key-compromise recovery\)](#)." NIST, May 2020.
- [24] S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu. "[RFC 3647: Internet X.509 PKI Certificate Policy and Certification Practices Framework \(\\$4.5.7 compromise and disaster recovery\)](#)." IETF, November 2003.
- [25] E. Rescorla. "[RFC 8446: The Transport Layer Security \(TLS\) Protocol Version 1.3](#)." IETF, August 2018.
- [26] NIST. "[Considerations for Achieving Crypto Agility: Strategies and Practices \(CSWP 39, update 1\)](#)." NIST, 29 June 2026.
- [27] F. Driscoll, M. Parsons, B. Hale. "[RFC 9794: Terminology for Post-Quantum Traditional Hybrid Schemes](#)." IETF, 2025.

- [28] NIST. "[FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard](#)." NIST, August 2024.
- [29] Microsoft. "[Analysis of Storm-0558 techniques for unauthorized email access](#)." Microsoft Security Blog, 14 July 2023.
- [30] Fox-IT. "[Black Tulip: Investigation of the DigiNotar Certificate Authority breach \(report to the Dutch Ministry of the Interior and Kingdom Relations\)](#)." Fox-IT, 13 August 2012.
- [31] N. Heninger, Z. Durumeric, E. Wustrow, J. A. Halderman. "[Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices](#)." USENIX Security Symposium, 2012.
- [32] Debian. "[DSA-1571: openssl – predictable random number generator](#)." Debian Project, 13 May 2008.
- [33] Let's Encrypt. "[A Post-Quantum Future for Let's Encrypt](#)." Let's Encrypt, 3 June 2026.
- [34] CISA. "[Emergency Directive 21-01: Mitigate SolarWinds Orion Code Compromise](#)." CISA, December 2020 (directive closed January 2026).
- [35] PatchBench authors. "[PatchBench \(arXiv:2609.04075\)](#)." arXiv, 3 September 2026.
- [36] 1Password. "[How information quality impacts LLMs' ability to patch complex, novel vulnerabilities](#)." 1Password, 6 August 2026.
- [37] Cloud Security Alliance. "[Agents in the Wire: AI Agents as Enterprise Insider Threats](#)." CSA AI Safety Initiative, 25 March 2026.
- [38] Cloud Security Alliance. "[FortiBleed: Anatomy of a 110-Million-Credential Harvesting Campaign](#)." CSA AI Safety Initiative, 25 June 2026.
- [39] Cloud Security Alliance. "[Frontier Ready Assessment Framework](#)." CSA, 8 October 2026.
- [40] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." CSA, 22 June 2026.
- [41] Cloud Security Alliance. "[AI Cryptanalysis and the Cryptographic Assumption Horizon](#)." CSA AI Safety Initiative, 20 October 2026.
- [42] Cloudflare. "[The state of the post-quantum Internet in 2025](#)." Cloudflare Blog, October 2025.
- [43] NIST. "[FIPS 204: Module-Lattice-Based Digital Signature Standard](#)." NIST, August 2024.
- [44] NIST. "[FIPS 205: Stateless Hash-Based Digital Signature Standard](#)." NIST, August 2024.
- [45] HAWK design team. "[HAWK signature scheme – official project page and withdrawal notice](#)." hawk-sign.info, 29 July 2026.