Enhancing Legal Document Security and Accessibility with TAF

Renata Vaderna
Independent Researcher
vrenata8@gmail.com

Dušan Nikolić
University of Novi Sad
nikolic.dusan@uns.ac.rs

Patrick Zielinski
New York University
patrick.z@nyu.edu

David Greisen
Open Law Library
dgreisen@openlawlib.org

BJ Ard
University of Wisconsin–Madison
bj.ard@wisc.edu

Justin Cappos
New York University
jcappos@nyu.edu

***Abstract*—The digital age has caused more and more services to be accessible online. A key exception to this has been access to the law, which remains published on paper or aging online platforms. Jurisdictions that have adopted digital law platforms often face difficulties with ensuring the security of their law online.**

In this paper, we introduce TAF, a system designed to secure legal repositories against unauthorized changes, and ensure the integrity of the law. Unlike prior archival or update frameworks, it is the first system that is designed for a threat model where an attacker fully controls the hosting repository. It also binds each signe repository state to publisher-defined legal dates, enabling verifiable as-of-date retrieval. First, TAF enables a repository of legal documents to remain accessible and authenticatable, no matter how long has passed since its publication. Second, TAF enables the independent verification of changes to a legal repository by anyone with read access to the repository. Third, TAF remains usable by users without a technical background or knowledge of cybersecurity.

TAF builds on the software-update guarantees of TUF, the version-control structure of Git, and a strong notion of time, where time is treated as signed data bound to specific repository states. TAF transforms the entire evolution of legal documents into an authenticatable, timestamped sequence of states, ensuring that every version, past or present, can be cryptographically verified. This property is not provided by Git or TUF alone.

We demonstrate that TAF is secure, scalable and performant, analyzing its behavior in various attack scenarios and its performance on large legal repositories, as well as ease of use. As a testament to TAF's security properties and performance, TAF is in production use by 14 jurisdictions in the US, including the City of Baltimore, the State of Maryland and Washington, D.C.

I. Introduction

Legal documents govern nearly every aspect of modern life. The written law is, in effect, the operating system of society. Yet in 2025, most laws are still published on paper, and their digital versions are often hosted on aging websites [1]–[4]. Many of these sites lack essential safeguards such as cryptographic signatures, checksums, timestamps, versioning, or audit mechanisms. They also provide no way for mirrors or users to independently verify that the content they access matches what was originally published. As a result, a single server-side mistake or an attack such as ransomware or account compromise can silently alter the legal record, leaving courts, citizens, and businesses with no way to validate what the law originally said.

At the same time, digital access to the law has become essential. From renewing a driver license to filing taxes, citizens increasingly expect digital access to government services and legal information. Yet digital law repositories, which serve millions of daily requests, still lag far behind modern software-update systems in integrity and security, which routinely use cryptographic signatures, version tracking, and multi-party trust models to prevent tampering [5]–[7].

A. Motivation

Legal systems must preserve not just the current version of the law, but also its precise wording at past points in time, as courts often rely on historical texts to interpret or apply precedent. [8], [9]. By comparison, most other fields do not demand this level of historical precision or permanence. Academic papers and books are generally static once published, and prior versions are rarely cited or preserved. Even source code repositories such as Git, [10] only weakly enforce the ordering of commits to different branches [11]. Legal documents, by contrast, are living texts—frequently updated, with past versions retaining legal significance.

The Uniform Electronic Legal Material Act (UELMA) [12], enacted in over 20 U.S. states, formalizes many of these concerns. It affirms that electronic legal materials must carry the same trust as their physical counterparts and remain verifiable and accessible over time. Taken together, these legal, institutional, and technical realities make the ability to retrieve and verify every prior version of a legal document not just desirable, but foundational.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241002
www.ndss-symposium.org

Requirement 1: Version Authenticity and Access

The current and all historical versions of legal documents must remain accessible and cryptographically authenticatable, together with the legally relevant date(s) for which each state is valid.

Moreover, government-maintained legal repositories are attractive targets for cyber threats and consequences of tampering can be severe. Attacks on these systems could undermine the legal process, reduce public confidence in its fairness and transparency, and ultimately threaten the foundations of democratic governance. Notable examples include the 2007 cyber attacks on Estonia and the 2008 attacks on the country of Georgia, both of which disrupted digital infrastructure— including legal databases and critical government services [13], [14]. More recently, ransomware attacks on public institutions, such as the 2021 attack on the Washington, D.C. Metropolitan Police Department, have raised concerns about the security of sensitive government-managed content [15].

In addition to external threats, there is the risk that institutions responsible for publishing or preserving legal materials could alter content after publication—either unintentionally, due to pressure, or perhaps because of a malicious insider who is bribed to change the law. This risk is not limited to the original publisher. Third parties, such as archives, libraries, or distributed institutions tasked with mirroring legal repositories, may also be subject to internal failures, external compromise, or political influence over time. Governments have historically altered records to shape public perception—not only in authoritarian regimes, but also in more subtle forms within democracies [16]–[19].

Requirement 2: Tamper Evidence and Institutional Independence

Legal documents must be verifiably protected against unauthorized modification. Their integrity must remain provable without relying on the ongoing trustworthiness of any individual publisher, mirror, or hosting institution.

Even the most secure archiving system will fail in practice if it cannot be used and maintained by its intended users. Legal publishers, librarians, and other staff who manage law repositories are often not technical specialists. A system intended to protect legal materials must therefore be designed with usability as a central concern. The process of publishing new documents, signing metadata, rotating cryptographic keys, or pulling and verifying updates should be straightforward and well-documented. If these tasks require following a fragile sequence of steps or understanding technical details, users might avoid performing them. Security best practices such as key rotation may be delayed or skipped if they are perceived as too complex to complete without assistance. Institutions that mirror or archive legal content should be able to set up their environments with minimal overhead. When errors occur, the system must provide clear, actionable diagnostics that help users understand and resolve the issue.

Requirement 3: Usability and Operational Clarity

All core operations, including publishing updates, verifying authenticity, managing signing keys, and diagnosing validation failures, must be simple enough for non-technical users to perform reliably.

B. Our Contributions

To meet the challenges outlined above, we developed The Archive Framework (TAF), a system for securing digital repositories that store official legal materials. TAF ensures archival integrity by combining two popular and widely used systems: The Update Framework (TUF) [20] and Git, and integrating a reliable, signed notion of time together with domain-specific legal functionality.

It is important to note that, although TAF builds on concepts from both Git and TUF, its security properties go well beyond a simple sum of the guarantees provided by the two systems. Git offers a Merkle-tree structure and preserves historical versions, but its security model is weak. History can be rewritten, commits can be removed, branches can be moved to unrelated points in the graph, and a malicious server can present different users with different versions of the commit history [11]. TUF addresses a complementary problem by securing softdate metadata and ensuring that users obtain the correct latest version. It protects against attacks such as rollback, freeze, mix-and-match, but it does so only for the current state. TUF does not authenticate the evolution of a repository over time and cannot prevent reordering, deletion, or rewriting of older states.

By combining Git and TUF in a way that attaches TUF-style authenticity to each point in Git's version sequence, TAF produces a linked chain of signed, ordered states. In doing so, TAF binds Git's structure of ordered versions with TUF's authenticatable metadata to create a tamper-evident timeline of all past states. Any attempt to remove, alter, reorder, or insert earlier states becomes detectable, as it breaks the chain of linked and authenticatable records. Likewise, attempts to spoof or replace content are prevented, since producing a valid state requires generating the corresponding correctly signed metadata. This construction directly addresses the limitations of both Git and TUF described above and provides the properties needed to securely maintain the historical record of the law.

On top of the authenticatable sequence of states, TAF layers a temporal component that links each snapshot of the entire corpus of the law with its relevant legal dates. In legal practice, a single law may carry multiple such dates—its enactment date, its effective date, and the dates on which later amendments altered it. Practitioners often need not only the historical text but also the version that was perceived to be in force at the time an event occurred. Because statutes are

frequently republished, corrected, or codified after the fact, "what the law was in 2020" may differ depending on which publication was available then. TAF captures this directly: a legally relevant date may correspond to multiple signed snapshots, each reflecting how the law was understood and published at different points in time. Because these dates are part of the signed metadata, TAF enables authenticatable temporal queries that neither Git, nor TUF, nor existing archival frameworks support.

Furthermore, TAF builds the technical foundation for a decentralized network of trustworthy institutions, such as libraries and archives, to mirror and validate legal repositories in a tamper-evident manner. To reduce operational complexity, TAF includes tooling such as a command-line interface, Git hooks, and clear error reporting to support safe use and key management with minimal technical effort.

TAF is already securing the legislative histories of fourteen U.S. jurisdictions, including Washington, D.C. (https://code.dccouncil.gov), the City of San Mateo (https://law.cityofsanmateo.org), the City of Baltimore (https://codes.baltimorecity.gov), and the State of Maryland (https://regs.maryland.gov).

The rest of the paper is structured as follows. Section III formalizes the security objectives of TAF and presents its threat model, grounded in real-world legislative workflows. Section IV and Section V describe the design and implementation of TAF. Section VI evaluates TAF across multiple deployments, demonstrating its scalability, performance, and ability to detect high-impact threats.

II. Background

TAF builds on two well-established technologies: Git, a distributed version control system used to track and retrieve historical revisions of legal content, and TUF, a metadata signing system originally developed to secure software updates against key or server compromise. On top of that, TAF treats time as signed data rather than as an authority: deployments attach law-related dates to each authenticated state, allowing queries by date against these signed fields. his section briefly reviews each foundation, highlighting the capabilities TAF inherits and the unique functionality it adds.

A. Version Control Integration

Long-term legal preservation depends on more than storing the latest version of legal documents, since courts and legislatures may rely on the exact wording of past texts (see Section I). Therefore, a trustworthy legal repository must retain not only the current revision, but also every prior revision in a form that can be independently retrieved and verified.

Achieving this requires an append-only history to prevent silent changes, a content-addressable structure that links each revision to its exact contents and predecessors, and robust support for retrieving historical states by identifier. In practice, lightweight branching is also valuable, as it allows jurisdictions to stage edits before publication without affecting the authoritative record. While other version control systems could meet these needs, TAF currently uses Git due to its widespread adoption, mature tooling, and established presence in public-sector projects [21]–[26]. Using Git also allows jurisdictions to reuse existing hosting services such as GitHub or internal servers, avoiding the need to deploy new infrastructure.

Git by itself is not a preservation or security solution. Repository history can be rewritten, hosting accounts may be compromised, and Git continues to rely on SHA-1 by default, despite known collision vulnerabilities [27]. While a transition to SHA-256 is underway, existing repositories cannot be seamlessly migrated without breaking commit identifiers, limiting the applicability of this transition for legal archives. TAF addresses these limitations by layering TUF over Git: each repository state is cryptographically signed, and deployments are encouraged to require a threshold of signatures from independent keys. This preserves Git's convenience while adding the integrity and long-term verifiability required for legal materials.

B. The Update Framework (TUF)

TUF is a security framework designed to protect software update mechanisms against key compromise and other repository-level attacks [28]–[32]. It supports a range of update models and provides tooling that integrates easily into both new and existing deployments. TUF has been adopted by major organizations, including IBM, VMware, Microsoft, Docker, Cloudflare, Google, and Amazon [32]–[36], and has graduated from the Cloud Native Computing Foundation.

TUF secures updates by attaching cryptographically verifiable metadata to each repository state. This metadata includes trusted signing keys, file hashes, version numbers, and expiration dates. Clients use this information to verify both the integrity and freshness of update files before applying them. Different metadata files serve distinct purposes and are signed by separate roles. In TUF, a role is a defined set of responsibilities and corresponding signing keys used to produce or validate a specific type of metadata. This separation of responsibilities [37], combined with threshold signing for critical roles [31], [37] and support for role delegation [30], [31], [37], ensures resilience even in the face of key compromise or infrastructure failure. For example, the root role in most deployments requires a signature threshold (e.g., 3 out of 6), with the corresponding keys stored offline on hardware devices such as YubiKeys. Delegation enables selective trust without requiring key sharing, a principle well established in distributed systems [38]–[48].

While TUF supports a wide variety of deployment scenarios, it is designed to validate only the latest version of a repository. Specifically, TUF considers old metadata to be a replay or freeze attack and thus intentionally prevents it from being validated. This makes it unsuitable for workflows that depend on the ability to retrieve and verify the state of a repository at a specific point in time (as is required in the legal domain). TUF also assumes continuous communication with a repository. If the original publisher becomes unavailable, clients have no standard mechanism to authenticate the

repository's content. This requirement renders TUF unsuitable for archival of legal documents. Finally, TUF assumes a level of technical proficiency typical of software engineering teams, which does not always hold in legal publishing environments.

III. Threat Model

Digital law repositories are an unusually attractive target: a single silent edit can alter how statutes are interpreted. Yet the publishers and archivists who manage this content often lack deep security expertise. Before presenting our concrete design, we therefore formalize the security objectives, assets, assumptions, and adversaries that guide TAF. This section first outlines the long-term goals and trusted components, then enumerates realistic attacker classes and the specific threats they pose. The subsequent design and evaluation sections map each countermeasure back to these threats.

Goals. TAF is designed to ensure the integrity and availability of legal materials over time. A single undetected change today can shape court rulings for years, so long-term integrity and verifiable provenance are essential for Requirements 1 and 2. Concretely, TAF must preserve the integrity of every revision and provide cryptographically verifiable provenance that does not depend on the ongoing availability or trustworthiness of the original publisher. It must also ensure that published legal texts remain accessible and verifiable for at least a decade, with the current reliance on Git's integrity mechanisms limiting stronger guarantees. Content must remain independently accessible and provable if and when the original publisher ceases operation.

Assets & Stakeholders. Four primary asset types underpin these goals. *A1* refers to the legal documents themselves, along with any associated multimedia or presentation-layer assets. *A2* includes the TUF metadata and revision history that record and authenticate changes over time. *A3* encompasses all signing keys defined by TUF. *A4* consists of update logs and backup copies maintained by independent institutions, such as libraries or archives, to support long-term verification and recovery. Safeguarding A1 and A2 directly supports Requirement 1, while protecting A3 and A4 enforces Requirement 2. Because the people who manage these assets are usually legal or archival staff rather than security engineers, their workflows must satisfy the usability mandate in Requirement 3. The direct stakeholders are government publishers, preservation institutions and end-users who rely on access to accurate and verifiable legal texts.

Trust Assumptions. TAF is designed to limit the number of components that must be trusted over extended timeframes. If the most important keys are stored offline on hardware-backed devices and protected by multi-party signing thresholds, the risk of compromise is low. The underlying storage layer provided by the version control system is expected to offer decade-scale integrity guarantees. The chosen cryptographic primitives (RSA-4096, Ed25519, and SHA-256) are considered secure for at least ten years. All other components, including hosting infrastructure and online keys, are treated as potentially fallible or adversarial.

Adversary Classes & Capabilities. We distinguish three attacker classes, ordered by increasing likelihood but varying in scope and mode of access. *The nation-state adversary (Adv-1)* has the ability to compel or compromise major infrastructure providers, such as cloud platforms or DNS authorities, and can observe or manipulate network traffic at scale. However, they are assumed to lack access to offline signing keys. *The insider adversary (Adv-2)* includes rogue publisher employees with signing privileges, or external attackers who compromise an employee's account, machine, or physical environment. This adversary may gain access to a single signing device, such as a YubiKey, and can perform targeted tampering from within the publication pipeline. *The external attacker (Adv-3)* exploits supply-chain vulnerabilities in platforms such as GitHub, CI/CD infrastructure, or application servers to gain limited control over hosted repositories or build systems.

Adv-1 most directly threatens Requirement 2 (tamper evidence), Adv-2 challenges both Requirements 2 and 3 by targeting institutional independence and key-management workflows, and Adv-3 endangers Requirement 1 by attempting to corrupt specific revisions. All adversaries are assumed capable of observing, modifying, replaying, or dropping network traffic.

Threat Enumeration. We distill the full STRIDE analysis into the six highest-impact threats affecting TAF, summarized in Table I.

Out-of-Scope. TAF does not harden user accounts or hosting platforms; it assumes those layers may be breached and concentrates on detecting unauthorized content and preventing clients from applying malicious updates. For example, a compromised account or signing key may lead to the publication of an invalid repository state, but such tampering will be rejected unless the metadata meets the configured signing thresholds. Large-scale denial-of-service attacks that block all network access remain out of scope, as do cryptanalytic breaks of RSA-4096 or Ed25519 before 2035. Section VIII discusses migration paths for those eventualities.

IV. System Design

TAF is designed to ensure the integrity and long-term verifiability of digital repositories containing legal documents and any associated assets a publisher may require, such as presentation files or auxiliary metadata. This section explains how these guarantees are supported over longer timeframes and covers: (1) the structure of TAF repositories, (2) the mechanisms that enable decentralized preservation and independent verification across institutions, and (3) the update and verification components, along with the tooling that helps non-technical users maintain system integrity.

A. TAF Repository Layout

TAF distinguishes between two types of repositories: *target repositories*, which store legal documents and associated assets, and *authentication repositories*, which define and verify the valid state of those repositories.TAF does not restrict how content is organized, letting publishers structure repositories as

TABLE I: Summary of Key Threats and Mitigations

ID	Threat Description	Mitigation Strategy
TM-1	Malicious updates by an adversary who gains access to a repository hosting account or the publisher's machine (Adv-1, S/T)	Updates must be signed by trusted parties; out-of-band verification is performed by independent institutions
TM-2	An earlier version of the law is reintroduced to hide or undo recent changes (Adv-2, R)	All valid versions of the law are cryptographically pinned
TM-3	Theft of a single signing key used to authorize updates (Adv-2, R)	Critical updates require multiple signatures; keys can be revoked and replaced
TM-4	The original publisher becomes unavailable or abandons the system (Adv-1, D)	Trusted institutions mirror the law and verify consistency across repositories
TM-5	Exposure of unpublished drafts or signing keys by a corrupt employee (Adv-2, I)	Multiple keys are required for updates; sensitive keys can be stored offline and rotated if needed
TM-6	Compromise of online components in the publishing process to inject unauthorized or malicious content (Adv-3, E)	Updates must be signed using offline keys; online systems involved in publishing cannot authorize changes
TM-7	Temporal deception via as-of-date mislabeling (Adv-1/Adv-2/Adv-3, S/T/R)	Signed temporal fields in targets; strict commit-by-commit validation; monotonic snapshot/timestamp checks; mirror update logs detect post-hoc rewrites

needed. The authentication repository contains cryptographically signed data describing the expected states of each associated target repository over time.

Each authentication repository is both a fully compliant TUF repository and is maintained under version control. This repository type was introduced to address the limitations of relying solely on Git or hosting platforms like GitHub for integrity. Legal documents are high-value assets, and the systems responsible for storing or managing them must withstand threats such as compromised accounts, stolen credentials, or unauthorized changes—threats explored further in Section III.

While TAF includes the standard TUF metadata files and follows the TUF specification for how target files are handled and signed, it imposes additional requirements on the structure and semantics of target files to support legal archiving. Additionally, TAF strongly encourages the use of offline keys for high-privilege roles, following TUF best practices for minimizing the impact of key compromise.

To determine which target repositories to validate and download, each authentication repository includes a special target file called *repositories.json*. Each repository is referenced using a `namespace/name` format, where the namespace typically corresponds to the jurisdiction or government body responsible for publication—for example, `dccouncil/law-xml`. This structure helps distinguish between repositories in multi-jurisdiction deployments. These human-readable identifiers do not authenticate the publisher; repository provenance must still be established through out-of-band mechanisms discussed later. The file is signed using a threshold of signatures, with keys ideally stored on separate hardware devices, ensuring that no single key compromise can authorize malicious repository updates.

Another target file expected by TAF is *mirrors.json*, which defines where the repositories are hosted. Because this file is signed, it reduces the risk of redirect-based attacks or accidental use of unofficial sources. Publishers may also include fallback locations to ensure availability if the primary host becomes unreachable.

For each target repository listed in repositories.json, there is a corresponding target file that records the most recent valid state of that repository. This includes an immutable identifier for the repository state, typically a commit hash, and the associated branch if applicable. Because a repository state is considered valid only when its identifier appears in signed metadata, updating a target repository must be coupled with updating the corresponding target file and signing the update with a threshold of signatures. Moreover, the authentication repository is version-controlled, meaning that its history establishes a cryptographically verifiable record of which repository states were considered valid over time.

Storing only an immutable identifier for each repository state serves an important purpose: it avoids treating every individual file in a target repository as a separate TUF target. This design delegates responsibility for internal structure and file-level integrity to the underlying storage layer (Git), which is particularly important for repositories containing large volumes of content, such as the DC Council's multi-gigabyte legal text repository with tens of thousands of files [49]. Tracking individual files in TUF metadata would introduce significant performance and maintenance overhead.

Target files may also include publisher-defined temporal metadata, such as effective dates, enactment dates, or publication timestamps (see Figure 1). When present, these

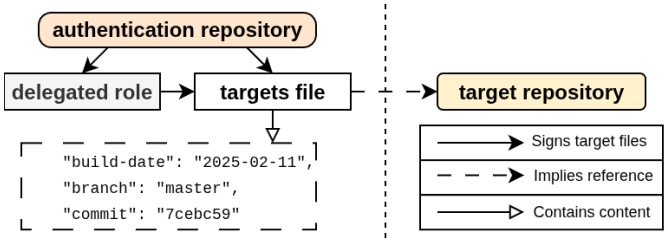

Fig. 1: Per-repository trust linkage. A delegated role in the authentication repository signs a targets file that records the state of a target repository.

timestamps are signed alongside the rest of the metadata. In

real-world deployments, this practice allows clients to retrieve the law as it was understood on a specific date.

Furthermore, TAF supports delegation of target roles, a feature inherited from TUF that allows signing responsibilities to be assigned to distinct roles. This lets organizations distribute signing authority according to operational responsibilities; for example, one role may handle updates to legal texts while another signs changes to presentation-layer assets (see Figure 2). Delegation also improves security by limiting the scope of each key and reducing the impact of a compromise.

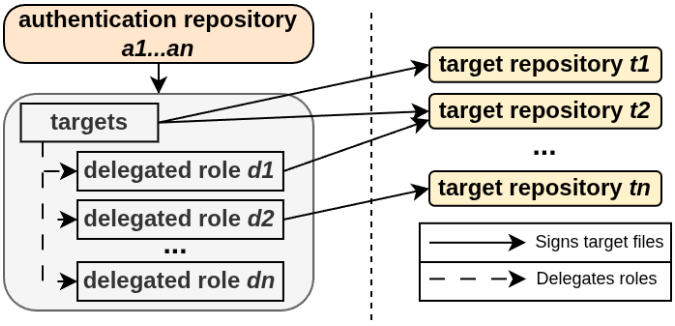

Fig. 2: The top-level *targets* role delegates trust to other roles for each target repository. Delegated roles sign metadata binding target files to specific commit hashes.

B. Interlinking Authentication Repositories

Each publisher using TAF typically maintains a set of repositories, including an authentication repository and one or more associated target repositories.This setup works for distributing their legal materials securely in the near term, but if there is only a single official copy of these repositories, or even if multiple redundant copies exist but are all maintained by the same entity, such a setup is unlikely to stand the test of time. With time, there is also the risk of the original publishing entity abandoning the repository or acting maliciously, for example under external pressure. To mitigate these risks, TAF supports the establishment of a broader network of trustworthy institutions, such as law libraries and archives, each of which maintains its own copy of repositories published by others.

TAF reuses the existing structure of authentication repositories to support this functionality. Specifically, it introduces a special target file called *dependencies.json*, which lists external authentication repositories in the same `namespace/name` format used in *repositories.json*. The URLs for these repositories are resolved using the same *mirrors.json* mechanism.

Each entry in *dependencies.json* includes an identifier corresponding to a verified state of the referenced authentication repository. Because namespace identifiers in TAF are not cryptographically bound to real-world identities, consumers must perform an out-of-band verification to ensure that the repository was actually published by the intended institution and not spoofed by an attacker. This process typically involves coordinating with the original publisher to confirm the repository's authenticity and establish a known-good starting point. Once this trust is bootstrapped, all subsequent repository states are validated using signed metadata recorded in the authentication repository. The mechanics of this validation are described in the next subsection.

In addition to *dependencies.json*, each referenced authentication repository has its own associated target file that records the validated states retrieved during previous updates. This establishes a cryptographically verifiable log of cross-repository validation events. Maintaining this log allows mirrors to record which repository states they validated at the time of update and detect if the publisher later removed or rewrote part of their authentication history. However, if multiple institutions mirror the same repository and later disagree about its contents, it may not be possible to automatically determine which version is correct. A government could pressure all domestic publishers and hosting entities to alter history, while a foreign mirror, beyond that government's reach, may preserve the authentic record.

Several architectural decisions described in this and the previous subsection address key threats outlined in Table I. Rollbacks (TM-2) are prevented by identifying valid repository states using immutable commit hashes. Single-key compromise (TM-3) and infrastructure attacks (TM-6) are mitigated through threshold signatures and offline key storage. Unauthorized updates (TM-1) and publisher disappearance (TM-4) are addressed through signed metadata and the use of independently maintained backup copies. The impact of leaked keys or premature publication of draft content (TM-5) is reduced through delegated roles, key rotation, and multi-party signing.

C. Update Process

The updater is TAF's core component that ties the system together. Its job is to securely clone each authentication repository, along with all referenced authentication and target repositories, onto the user's local machine. Each time the updater runs, it checks for changes and performs validation before applying updates to local copies. Validation ensures that the information recorded in the authentication repository matches the actual state of each associated repository. This is TAF's main mechanism for detecting unauthorized changes.

Every update cycle begins by validating the authentication repository itself. Since this repository contains data about the expected state of all associated target repositories, and potentially other authentication repositories, it must be trusted before any further validation can proceed. Each authentication repository is expected to be a valid TUF repository at every revision. To ensure this, the updater performs sequential validation: rather than checking only the latest metadata, it verifies the entire range of new revisions one by one. If any revision in the update range fails validation, the updater halts and retains only the last known valid revision. The actual validation between two revisions follows the TUF specification [50]. It checks whether metadata files are properly signed, version numbers increase correctly, and target files are valid according to the signed metadata.

Once the authentication repository has been validated, the next step is to verify each referenced target repository using the TUF metadata and target files whose validity was confirmed in the previous step. The validation of target repositories involves checking whether the recorded state identifiers (such as commit hashes) in the target files match the actual state of each target repository. The overall workflow is depicted in Figure 3.

To avoid re-validating the entire repository history on each run, the updater records the last successfully verified state of each repository in a local configuration file. This file is treated as untrusted and is always verified against the current state of the repositories before resuming from that point. If any discrepancies are found, the updater falls back to validating the full history from the beginning.

If the authentication repository references additional authentication repositories, each one is validated recursively using the same procedure: first verifying the authentication repository itself, then checking its associated target repositories. The updater also verifies that the referenced repository includes the known valid state that was designated during initial out-of-band verification. If this state is not present, the update fails. For every successful update to a referenced authentication repository, a corresponding target file is created or updated in the referencing repository to record the revisions that were pulled and verified. This forms a signed update log, akin to a transparency log, allowing future verifiers to trace and audit the sequence of updates.

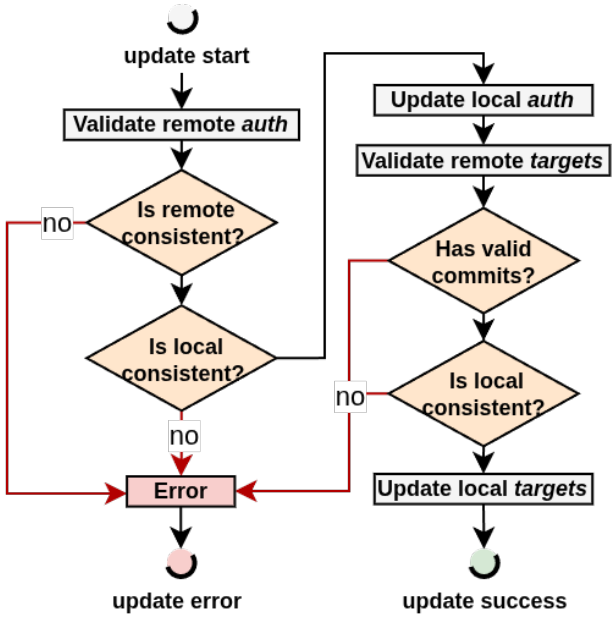

Fig. 3: Updater workflow in TAF. The updater validates the authentication repository and all referenced target repositories, checking both remote and local consistency and aborting on the first mismatch.

Although the update process is conceptually straightforward, real-world use introduces several complications. In the best case, all remote repositories provide valid, up-to-date metadata and the local clones remain unchanged since the previous updater run. In practice, however, users may make manual changes during regular publishing workflows. This may happen due to urgency, automation scripts, or lack of awareness. Users might also apply updates locally without running the updater, revert changes, delete history, or update a single repository in isolation. They may also commit changes without pushing them. These actions can leave repositories in divergent or inconsistent states.

TAF's updater is designed to detect and repair such inconsistencies by comparing the local state of each repository to the expected state recorded in the last validated signed metadata. If local changes are found that do not match verified history, such as unpushed commits, missing updates, or unauthorized rewrites, the updater halts and reports the discrepancy. An optional flag enables automatic repair by removing unverified commits and restoring the repository to its last valid state.

While the updater can detect and resolve inconsistencies within a single environment, reconciling conflicting repository states across multiple institutions is a much harder problem. As discussed in the previous section, a coordinated rewrite, possibly under external pressure, could lead to a majority of actors hosting a modified version of history. In such cases, the updater cannot determine which version is correct, and conflict resolution is outside the scope of the current system. Finally, to reduce the risk of introducing invalid updates during routine use, TAF installs a pre-push Git hook in each authentication repository. If validation fails, the push is rejected.

D. Temporal Fields and Timeline Semantics

TAF target files can include publisher-defined temporal metadata, such as effective or applicability dates, which are signed alongside the rest of the TUF metadata. This allows clients to answer as-of-date queries (for example, "what was the law on January 1, 2023?") by identifying the latest repository state whose signed temporal fields indicate legal validity on that date. These dates are defined outside TAF, typically by the codification process, which determines which timeline each repository state belongs to. TAF itself records the branch field in the target metadata, enabling the preservation of multiple legal timelines within the same repository. Future-dated builds are committed to separate branches and remain inactive until their effective date arrives. A continuous integration (CI) process evaluates the local time policy at each execution and merges the future-dated branch into the canonical branch only when the date is reached. TAF does not include a secure time oracle for determining the current time. Instead, it assumes that the deployment environment provides time input, and defining a secure time-source interface is deferred to future work.

Temporal enforcement: When a future-dated state becomes canonical is determined by the deployment's CI policy. TAF verifies whatever state the publisher has designated as valid, including its signed temporal fields.

E. Tooling for Repository Management

Even with a robust updater, the system would offer limited value if those responsible for publishing the law or maintaining repositories found it difficult to keep them in a valid state during regular updates or key rotations. To address this, TAF provides a command-line interface (CLI). Through this CLI, users can run the updater, which emits clear, actionable error messages when validation fails. For example, highlighting mismatches such as "expected commit A, but found commit B". In addition to validation, the CLI supports all core maintenance operations on the authentication repository, including updating expiration dates, resigning metadata, rotating keys, managing roles, and registering new repositories. These tools are designed to satisfy Requirement 3 (Usability and Operational Clarity), ensuring that users can maintain secure repositories without needing deep technical expertise.

For more advanced use cases, TAF also provides a Python API that enables developers to integrate authentication repository management into external systems. For instance, a publishing entity may use the API to automatically update TUF metadata as part of their own law publishing pipeline.

F. Security Argument

TAF provides authenticity, integrity, and verifiable history for all repositories it manages. This subsection states the assumptions under which these guarantees hold and summarizes why they follow from the system design.

Assumptions and authenticity guarantees. TAF assumes a trust-on-first-use model: each hosting institution performs an out-of-band check to obtain a known-good initial state of a publisher's authentication repository before adding it to the set of repositories it mirrors. (for example, by coordinating directly with the publisher or verifying DNS records under the publisher's control). Publishing a bare commit hash on a website is not sufficient to rule out a spoofed repository. Third parties that clone publishers directly must perform the same check, whereas cloning from an institution that has already done so only requires cross-checking the recorded trusted state. We also assume that multiple copies of each repository exist in practice, including offline backups, and that the cryptographic algorithms used by TUF and the Git hash functions remain secure on a decade time scale; extending protection beyond that horizon is left to future work.

Once an initial state is trusted, later states are verified using signed TUF metadata and target files that bind specific commit hashes. The updater validates each new revision in sequence. Any attempt to introduce unauthenticated content or rewrite a previously validated state produces a mismatch in signatures, version numbers, or commit identifiers and causes validation to fail. Attempts to remove newer content are also detectable: a previously validated offline copy reveals that the remote repository is missing commits that were authenticated earlier, and hosting institutions maintaining their own validated update records can detect the same discrepancy.

Cross-repository security. For dependencies on other authentication repositories, the same argument applies. A referencing repository records validated states of external repositories in its own signed target files. Any attempt to rewrite or remove part of a dependent repository's authentication history produces an observable inconsistency.

Under these assumptions, TAF preserves the long-term verifiability and integrity of repository contents for as long as the assumptions hold, with Git providing the limiting time horizon.

Long-term validity of signatures. TUF already supports algorithm agility: deployments can rotate keys and adopt stronger algorithms over time, while historical signatures remain valid as long as their primitives are still secure. In TAF, once a past state has been authenticated, it stays verifiable even as newer configurations evolve. Our goal is to maintain cryptographic strength on decade-scale horizons, with operators expected to migrate to newer schemes through routine key rotation when older primitives weaken. TAF is also among the first real-world deployments to adopt post-quantum signature schemes.

V. Implementation

TAF is implemented in Python and currently supports Git as its only version control backend. The system is structured around core components that represent Git repositories, TUF repositories, and authentication repositories that incorporate both. These components form the foundation for the updater, a programmatic API, and a command-line interface (CLI) designed for both technical and non-technical users. TAF is already deployed in 14 jurisdictions, including city, state, and tribal governments.

Git operations are handled via `pygit2`, and TUF repositories are implemented in accordance with the full TUF specification using `python- tuf`, including metadata management, role delegation, and signature verification. Signing operations can be performed using hardware-backed keys (YubiKeys), with YubiKey support provided via `yubikey-manager`, although users may choose to use software-based keys for specific roles depending on operational needs or threat models.

TAF operates at the repository level and is agnostic to any document-level electronic signature format used for individual statutes or regulations. In practice, jurisdictions that already rely on advanced or qualified e-signatures (for example, PAdES [51]–[54] for signed PDFs) can layer those schemes on top of TAF's metadata, treating repository signatures as complementary provenance for the underlying documents.

To simplify setup for non-technical users, each TAF release includes self-contained executables that bundle the system with a Python interpreter. These packages are built and published through a continuous integration pipeline, which also produces a PyPI package.

The codebase is extensively tested, with unit tests covering core components such as TUF logic and Git operations, and integration tests validating the updater, API, and CLI across a range of valid and invalid repository states. All tests run through continuous integration on every push. The updater

has been optimized through parallel cloning, incremental validation, and an option to validate authentication repositories independently when full verification is not needed.

A long-term archive is only as strong as its signing keys, which can age out, be misplaced, or be exposed during staff turnover. To keep the repository trustworthy after such events, TAF makes key rotation routine: a single CLI command replaces an old key, updates metadata, and ensures clients accept the change without disruption. By turning a risky security task into simple maintenance, TAF closes the ”single key theft”” threat (TM-3) and allows non-technical staff to recover quickly from compromise or expiry.

In parallel, TAF's integration with version control is designed for long-term flexibility. Although Git is currently the only supported backend, all Git-specific logic is encapsulated in a single module. This allows the version control layer to be replaced with minimal changes to the updater or metadata logic, as long as the alternative system supports core concepts such as state identification, revision history traversal, access to file contents at a specific revision, and cryptographic integrity guarantees for stored content. Git's ongoing SHA-1 to SHA-256 transition, and its lack of canonical repository semantics, meaning the inability to determine whether two repositories represent the same complete and trusted history, remain limitations, revisited in Section VIII.

VI. Evaluation

Because TAF is used on real-world repositories, we chose to use them instead of demo ones created specifically for the paper. Before evaluating TAF, we consulted with our IRB for approval. Our IRB office determined that this project is not one that constitutes human-subjects research and does not require IRB review as our work did not involve human subjects research as defined by 45 CFR 46.102(e) [55].

We evaluate TAF along three dimensions that correspond directly to the threat scenarios defined in Section III: (1) detection of unauthorized or tampered updates (TM-1 to TM-3), (2) resilience to publisher or mirror failure (TM-4), (3) confidentiality and protection of the publishing pipeline (TM-5 and TM-6) as well as protection against temporal deception in as-of-date queries (TM-7).

A. Security Analysis

TAF defends against attacks common in Git-based distribution systems, with a security model specifically focused on the long-term authenticity and availability requirements of digital legal materials. TAF's threat model considers several potential attacks that could compromise the repositories or the signing infrastructure.

To aid in understanding the threat scenarios considered and the corresponding TAF behaviors, Table II summarizes the attacker capabilities, their attempted actions, the resulting system behavior, and whether the attack would be detected by the TAF updater. This overview provides context for the detailed examples that follow.

One common scenario is the compromise of a publisher's hosting platform account or development machine (TM-1 in Table I). If an attacker pushes commits directly to a target repository without updating the authentication metadata, the updater does not attempt to pull new data. However, if it detects unrecorded changes, it raises an error and retains the last authenticated state. Listing 1 shows the single-line message a client sees in this situation.

Listing 1: Updater output on unexpected target commits

```
Target repository mohicanlaw/law-html does not
    allow unauthenticated commits,
but contains commit(s) 2648
    aef6db9bacb6c5ce8de49b3015492eb9e0c3
on branch publication/2025-02-20
```

When the attacker also modifies the corresponding *targets/*.json* file (“Target + targets file”), the update is still blocked because the signatures in that file no longer match the hashes recorded in *snapshot.json*. A more sophisticated adversary might edit every metadata file (“All metadata files”), but without the necessary private keys the signature-threshold check fails and the updater halts.

If snapshot and timestamp keys are stolen (“Snapshot / timestamp keys”), an attacker can attempt a rollback or freeze attack by serving older metadata. In practice this fails because TAF validates commits strictly monotonically and verifies that file hashes never regress; the version or hash mismatch is caught immediately.

We also emulated a coordinated force-push that deletes authenticated commits from *both* authentication and target repositories (“Consistent force-push”). A fresh clone appears valid, but any client that had already advanced beyond the removed commit detects the missing hash on its next validation pass and raises an error until the publisher restores the history.

The only scenario that escapes automatic detection is a full compromise of all signing keys with enough key material to satisfy every role's threshold (“Full key compromise”). We note, however, that such an attack requires physical or coercive access to multiple hardware tokens distributed across different custodians, a bar significantly higher than typical software update systems.

Across all tested rows in Table II, the updater either (i) leaves clients on a known-good state or (ii) aborts before any unauthenticated content can be applied. No silent acceptance or partial rollback was observed.

B. Usability and Operational Robustness

Beyond protecting against intentional attacks, TAF also aims to minimize accidental mistakes during the publishing process. It is not uncommon for maintainers to make unintended changes that leave a repository in an invalid state. To reduce this risk, the TAF updater installs a `pre-push` Git hook whenever an authentication repository is cloned. This hook runs a complete validation check before allowing any push. While an attacker could bypass this hook, its presence helps eliminate the vast majority of accidental errors that would otherwise invalidate the repository.

TABLE II: Evaluated threat scenarios and updater behavior. TM-1 through TM-6 correspond to the threat model in Section III. "Detected" indicates that the updater halts with an error.

Threat	Adversary action	Updater outcome	Detected
TM-1	*Compromised account or machine with push permission.* Attacker commits directly to a target repository.	Updater does not pull target changes; metadata unchanged.	Yes
TM-1	*Compromised account or machine with push permission.* Attacker commits to a target repository and updates the corresponding `targets/*.json` target file.	Signatures fail verification; update rejected.	Yes
TM-1	*Compromised account or machine with push permission; attacker understands TUF structure.* Attacker commits to a target repository and edits `targets`, `snapshot`, and `timestamp`.	Fails signature threshold check; update rejected.	Yes
TM-2	*Machine or account with force-push permission compromised.* Attacker removes commits consistently across target and authentication repositories.	Fresh clone validates; existing clones raise error until repaired.	Partial
TM-3, TM 6	*Partial infrastructure compromise leaks snapshot and timestamp keys; attacker can also push to repositories.* Attacker serves outdated metadata to induce rollback or freeze.	Version/hash mismatch detected; update rejected.	Yes
TM-5	*Full compromise of signing infrastructure and repositories; attacker controls threshold keys for all roles.* Attacker signs and pushes a malicious update that meets the signature threshold.	Updater accepts; no alarm raised.	No
TM-7	*Adversary edits the temporal fields in the signed target file to mislabel the as of date.*	Commit-by-commit validation finds the mismatch. Snapshot and timestamp checks fail. Update rejected.	Yes

Although not a security concern, one of the most common problems publishers encounter in practice is the accumulation of unintended or temporary commits in their local repositories. This often happens during testing, or day-to-day use of the repositories, when a publisher may commit changes locally without intending to publish them. When the updater is run in this state, it raises a validation error because the local repository diverges from the authenticated state recorded in the metadata. They might also manually update individual repositories without running the updater, which can leave the system in an overall invalid state.

To address this without requiring manual intervention or a full re-clone, TAF supports a `--force` flag. When invoked, the updater will forcibly align the repository with the most recent validated state, and then apply any available updates. The option provides a safe recovery path for publishers who are aware of the inconsistency and intend to proceed. This significantly improves operational robustness while maintaining the integrity guarantees required for secure publication.

C. Performance

This subsection quantifies the runtime cost of TAF 's updater and answers four questions:

(i) How long does a worst-case "fresh clone + full history validation" take on real deployments? (ii) How does that cost scale with the number of authenticated commits and target repositories? (iii) What is the overhead of day-to-day incremental updates once a repository already exists on disk? (iv) How portable is performance across operating systems and commodity hardware?

We present measurements from five production instances, derive asymptotic behavior ($O(T) + O(N)$ vs. $O(\Delta N)$), and discuss OS-level variations observed in the field.

Experimental set-up. All benchmarks ran on a bare-metal workstation with an Intel Core i7-14700K (20 cores, 28 threads, 33 MiB L_3, 5.6 GHz turbo), 32 GiB DDR4-3200, and a Samsung 990 PRO 1 TB PCIe 4.0 NVMe SSD. No virtualization was used. Hyper-Threading and Turbo Boost were left enabled. Repositories resided on the same NVMe volume, eliminating I/O interference. The updater uses one worker per CPU core by default.

Workloads. We selected five real deployments covering 295–3346 authenticated commits. For each repository we measured (i) clone + validate wall time (`taf repo clone`), and (ii) pure validation time, obtained by subtracting Git transfer time. Target repository commit counts are reported to show dataset scale, although only commits referenced in authentication metadata are validated.

TABLE III: End-to-end cloning and validation performance of the TAF updater.

Jurisdiction	Auth Cmts.	Clone (s)	Val. (s)	Thpt. (c/s)	Targets Cmts.†
cityofsanmateo/law	3346	60.3	20.8	160.9	5958
sanipueblo/law	2336	37.4	14.8	157.8	4742
mohicanlaw/law	643	13.6	3.9	165.3	592
DCCouncil/law	582	230.0	3.6	163.9	1241
tmchippewa/law	295	11.4	2.1	142.5	523

†Total commits across all target repositories.

Complexity. Let N be the number of authentication commits since the last trusted state and T the size of Git objects transferred. The first invocation (`clone`) is therefore $\mathcal{O}(T) + \mathcal{O}(N)$, dominated in practice by network and storage I/O for T. Table III captures this worst-case path. Subsequent incremental runs skip Git transfer and resume validation from

`last_validated_commit`, making cost strictly $\mathcal{O}(\Delta N)$, where $\Delta N \ll N$ in routine operation. Empirically, San Mateo updates that add a single commit complete in < 300 ms, comparable to `git pull`. This keeps publication latency low while retaining full cryptographic auditability.

Results. Table III shows that authentication commit validation sustains roughly 150–165 commits/s across the entire range of repository sizes. City of San Mateo, the largest deployment, validates 3346 commits in 20.8 s. Extrapolated linearly, a 50 000 commit archive would finish in about 5 minutes. Clone time is dominated by Git object transfer and scales with raw repository size. For example, DC Council hosts sizable HTML and XML assets, so transfer inflates updater time to 230 s, even though validation itself completes in 3.6s. Once repositories are cached locally, subsequent incremental runs validate only new commits as noted above.

Scalability and portability. Because validation cost grows linearly with N and is parallel across repositories, throughput improves with core count. Peak memory stayed below 250 MB RSS even for the largest workload. *OS variation.* Our measurements use Linux; production deployments show that the identical workload on Windows incurs a 1.5–2 $\times$ slowdown due to pygit2 file-system overhead and less aggressive mmap caching on NTFS. The asymptotic behavior is unchanged, but operators targeting Windows servers should provision proportionally faster storage or longer maintenance windows.

Take-away. The updater's first clone path is bounded by network I/O. Pure validation is fast, linear, and parallelizable. Even on commodity hardware, and despite slower pygit2 performance on Windows, TAF accommodates updating legislative histories without disrupting daily publication schedules or over burdening archival mirrors.

D. Deployment Experience

TAF has been deployed across 14 publishers, including Washington, D.C. and the City of San Mateo. Washington, D.C. began working with our team in 2017 to modernize the preparation and publication of its statutes, and the District's subsequent adoption of UELMA created a formal requirement for long-term verifiability. This prompted the development of TAF in 2019 as an open-source framework for secure legal publication. An IMLS-backed rollout at the National Indian Law Library has brought eight tribal governments online since 2020, and a statewide deployment for Maryland is currently underway. Together, these deployments mean that TAF has been thoroughly battle-tested under real-world conditions. In the following discussion, we use "users" to refer to the publishing staff who operate TAF within their organizations.

In all deployments, TAF has been integrated into the existing publication workflows of the publishing entities. When publishers first begin using TAF, they complete a one-time setup that includes configuring the authentication repository. As part of their normal publishing pipeline, publishers generate updated XML, HTML, and other output formats and commit these changes to their repositories. TAF does not replace this process. Instead, it acts as the final step in the pipeline and does so with minimal exposure of its inner workings. When ready to publish updates, users are prompted to insert signing YubiKeys, and TAF handles the rest.

The observations below summarize how users perceived TAF in practice and how its design aligned with their existing mental models.

To begin with, all users participated in training sessions introducing TAF. Some initial feedback reflected assumptions carried over from previous systems. Several publishers were hesitant to grant broader repository access, since in their earlier workflows database access usually implied write access. Once they understood that TAF restricts modifications to signed updates and that read access cannot alter content, they became comfortable allowing broader visibility into their repositories.

The permanence of history was also a shift. Publishers were accustomed to systems where mistakes could be silently edited or overwritten. With TAF, corrections appear as new signed commits rather than deletions. Once this was framed as an archival requirement, users generally viewed it as a benefit. If a version of the law was published as official, even for a single day, it was still official for that period, and erasing it due to an unintentional error would be misleading—a perspective TAF helped publishers adopt.

Publishers also appreciated being able to check whether two copies of a repository are identical or if one is lagging behind. Authenticated clones and detection of missing history or rewrites clarified which copy is authoritative, addressing a long-standing operational pain point. Preparing to adopt TAF led several publishers to review decades of materials, during which they identified and corrected issues such as missing paragraphs and incorrect enumerations dating back to the 1970s.

Additionally, TAF is being trial-integrated into public library kiosks to provide citizens with verifiable offline access to current and historical versions of the law. These deployments will introduce a new class of users outside the publishing entities, and feedback from public-facing installations is not yet available.

VII. Related Work

Numerous systems have tackled the problem of securely storing and distributing digital content, but none fully address the combined needs of long-term preservation, verifiable provenance, and time-based access required in the legal domain. Traditional digital preservation tools emphasize durability and redundancy, but lack cryptographic guarantees or mechanisms to authenticate content provenance. In contrast, other systems built on TUF prioritize fast updates and short-term integrity, without preserving verifiable history. Notably, no prior system supports cryptographically verifiable retrieval of a corpus as it existed at a given point in time. This section reviews both categories and provides a detailed comparison with TAF in the context of archival tools, along with an overview of related TUF-based efforts.

A. Existing Digital-Archiving Solutions

LOCKSS (Lots of Copies Keep Stuff Safe) [56] is a long-standing digital preservation system originally built around a peer-to-peer polling protocol among mutually distrusting nodes. Content is ingested through web crawling and automated content registration based on publisher input [57]. Successive crawls may capture multiple copies of the same URL, but LOCKSS does not track or relate these as meaningful versions of a specific document. LOCKSS 2.x introduces a substantial architectural rework that simplifies deployment and improves local fixity checking, but neither version of LOCKSS authenticates ingested content against a trusted origin.

DSpace is a widely adopted open-source repository platform to manage and preserve digital content [58]. Content is organized into items, each comprising multiple bitstreams (files) and associated metadata. Ingest is performed by authorized users through a web interface, command-line tools, or API-based deposit protocols. While multiple versions of a digital item can be represented as separate items and manually linked via metadata, DSpace does not provide native version tracking. For preservation, it computes and stores per-bitstream checksums in its internal database and includes a fixity checking task to detect local corruption. However, it does not establish provenance for ingested content, like LOCKSS.

Archivematica is an open-source tool for packaging content into archival formats, typically used with external backends such as LOCKSS or cloud object stores [59]. It performs format validation and embeds checksums for fixity checking, but does not establish cryptographic provenance or tamper-evident history. **Preservica** is a proprietary platform offering ingest and fixity checking, but its internal mechanisms are not openly auditable [60]. As neither system provides a standalone preservation solution with open and integrated guarantees, we exclude them from closer comparison.

B. Other TUF-Based Systems with Related Goals

Repository Service for TUF (RSTUF). RSTUF is an open-source project that streamlines TUF deployment and update workflows via a microservice backend [61]. It improves usability for systems adopting TUF but is not designed for long-term preservation or full-history validation, focusing instead, like TUF, on the secure delivery of the latest update.

gittuf [62]. gittuf is a security layer for Git repositories that enables the definition and application of a write access control policy for Git repositories. gittuf's metadata is based on that of TUF, but has some additional extensions compared to that of the TUF standard. While seemingly related to TAF, gittuf applies TUF-like policies to a git repository while TAF stores historical TUF metadata (and secure time information) in a git repository. Put differently, gittuf focuses on improving the security of a git repository by applying a limited set of TUF policies (revolving around the targets role), thus it is inherently geared for rapid, repeated releases where policy checks involve only the current policy at that moment. In contrast, TAF contains what is effectively a series of full TUF repositories (all four TUF roles) and time metadata. Thus TAF is more suited for periodic releases of information that needs more robust protection against tampering.

C. Comparison with Related Systems

Table IV compares TAF to archival and TUF-based systems across five core properties relevant to legal publishing. We include Git and TUF, along with derived tools like RSTUF and gittuf, to illustrate which relevant properties they support. However, these tools are not designed as standalone archival systems and do not meet the full requirements for long-term archival integrity. Our primary comparison centers on archival systems, LOCKSS and DSpace.

TABLE IV: A comparison of various systems and TAF across core properties: ① Retrieving and authenticating past states, ② Cryptographic provenance , ③ Support for authenticated as-of- date queries), ④ Decentralized validation of indepentent mirrors, ⑤ Repo-compromise resilience.

System	①	②	③	④	⑤	Domain
LOCKSS	**Y**	N	N	P	P	Archiving
DSpace	P	N	N	N	N	Archiving
Git	P	N	N	N	N	Version control
TUF	P	**Y**	N	P	P	Software updates
RSTUF	N	**Y**	N	N	P	Software
gittuf	**Y**	**Y**	N	N	P	Source code
TAF	**Y**	**Y**	**Y**	**Y**	**Y**	Legal documents

Y = Yes, N = No, P = Partial

Archival systems vary in how they conceptualize preservation, including what counts as trustworthy storage and which aspects of content and history they aim to preserve. Both LOCKSS and DSpace focus on long-term access to individual files. While DSpace organizes items into collections, and both systems retain multiple instances over time, they do not automatically relate these as meaningful, linked versions or maintain a coherent, cryptographically verifiable repository-wide state. These designs implicitly assume infrequent updates or that the latest version is sufficient for most users. In contrast, TAF models the archive as a sequence of signed, linked snapshots, enabling authenticated, time-bounded queries over the entire repository.

Both systems treat ingestion as the starting point of trust. Once content is ingested, its integrity is protected, but the correctness of the ingested version is assumed rather than verified. DSpace and LOCKSS rely on basic access control, so a single compromised admin can alter the archive. As discussed above, TAF requires not just repository access but a quorum of valid signatures.

An additional pillar of long-term preservation is the existence of multiple independently verifiable mirrors. But majority-based repair assumes honest nodes—an unrealistic guarantee in adversarial legal settings. TAF also aims to involve a network of institutions, but resolving disagreements among them remains an open challenge. DSpace offers no comparable mechanism for cross institutional validation.

D. Performance and Deployment Comparison

To complement our security analysis we evaluated the performance and deployment characteristics of TAF compared to DSpace and LOCKSS. Due to fundamental differences between the tools not all aspects are directly comparable. In particular TAF performs full-state cryptographic validation at download time whereas DSpace and LOCKSS do not reverify integrity upon retrieval. As a result metrics like download time and validation behavior reflect fundamentally different models and are not directly comparable. We therefore focus on ingest performance and setup complexity.

We benchmarked ingest performance using the current version of a medium-sized legal repository from the City of San Mateo (56,890 files, 601.7MB). All tests were run on a modern Linux laptop. For LOCKSS, we hosted the files via Python's HTTP server to enable crawling (adding a slight overhead of 0.1s per request). Ingest took just under two days (3s per file). For DSpace, we followed its item-based model by grouping 5 related files (e.g., multiple formats of a law) into 11,325 items, which were imported via the CLI in approximately 5 hours. With TAF, ingestion involves committing and pushing content to a set of Git repositories. A pre-push validation hook runs when pushing to the authentication repository, and the entire process completes in under 5 minutes.

System	Input	Time	Notes
TAF	56,890 files	<5 min	Git clone + validate
DSpace	11,325 items	∼5 h	Grouped import via CLI
LOCKSS	56,890 files	∼2 days	HTTP crawl

TABLE V: Ingest performance for 56,890-file dataset on a Linux laptop (20 cores, 512GB SSD, 16GB RAM).

The results are summarized in Table V and demonstrate that existing tools are not optimized for injecting large numbers of granular items, a design choice preferred in this context as it enables precise change tracking and efficient retrieval of individual legal documents and their respective versions.

Additionally, we note that both LOCKSS and DSpace require substantially more involved installation and configuration, often involving multiple dependent services, orchestration frameworks, or manual setup steps not required for TAF.

VIII. Limitations and Future Work

Git object hashes. Git still stores objects under SHA-1, which now carries practical collision risk. Since Git v2.29 the core tool can create repositories that use the stronger SHA-256 hash format [63]. Converting a large legislative history, however, would break every existing commit identifier. For the deployments studied in this paper we continue to serve SHA-1 repositories while monitoring the field for real-world pre-image attacks; a controlled transition will be scheduled once SHA-256 repositories are widely supported by hosting platforms and tooling.

Cryptographic agility. NIST recommends moving to quantum-resistant primitives by 2035. Git currently recognizes only SHA-1 and SHA-256, so future moves to SHA-512/256 or a post-quantum hash will require upstream changes followed by a one-time re-signing of authentication metadata. TAF's repository abstraction isolates most hash handling, making such a migration a bounded engineering task rather than a redesign.

Publisher-defined dates. In current deployments, a continuous integration (CI) process or scheduled task merges future-dated branches into the canonical timeline once the effective date is reached, according to the publisher's policy. While TAF does not currently enforce time authenticity, the introduction of a pluggable time-source interface—such as authenticated network time or multi-party time attestations—is left as future work. This limitation does not compromise the verifiability of signed temporal assertions already embedded in the target metadata.

Long-term horizon. Present deployments target a ten-year audit window, matching hardware refresh cycles for the hosting jurisdictions. Extending that guarantee to several decades will require periodic cryptographic upgrades and packaging the verifier toolchain itself in a signed, self-contained bundle. Work on that "century profile" is under way.

Ongoing engineering. Two prototypes are in progress: (i) a usability study of key rotation and rollback recovery performed by non-technical clerks and (ii) a pilot integration of *in-toto* with TAF so that the entire legislative build pipeline—from raw source files to the signed, published state—carries end-to-end supply-chain provenance [64]. These enhancements are incremental and do not affect the security claims evaluated in this paper.

IX. Conclusion

Access to trustworthy law is a cornerstone of any modern democracy, yet the tools traditionally used for software and data security have rarely been applied to legal texts. The Archive Framework (TAF) bridges that gap. By combining Git's proven version-control semantics with TUF's compromise-resilient metadata, TAF turns every statutory revision into a cryptographically and independently verifiable record. Publishers gain a familiar workflow for drafting and releasing amendments; libraries and archives gain a lightweight mechanism for mirroring and auditing entire legal corpora; and end-users, from judges to researchers, gain confidence that the statute they see today will match the one cited years from now.

TAF's design addresses three persistent challenges in this domain. First, it guarantees long-term authenticity without relying on any single host or cloud provider, making it robust to political change or infrastructure failure. Second, it offers tamper evidence and rapid key-revocation workflows so that even highly resourced adversaries cannot silently rewrite history. Third, it packages these guarantees behind a CLI and API tailored for clerks and archivists rather than security engineers, easing real-world adoption.

Early deployments across fourteen jurisdictions show that the system scales from small municipal codes to multi-gigabyte state registries with no changes to core logic. While

limitations remain, most notably Git's evolving hash functions and the need for post-quantum cryptographic agility, TAF provides a concrete, practical path forward. Work on automated hash migration, divergence-resolution protocols, and formal verification will extend its protective horizon from years to decades and ultimately to the century-scale timelines demanded by the legal domain.

TAF demonstrates that the security principles long used to safeguard software updates can and should be applied to the law, turning digital statutes into durable, self-authenticating public records.

Acknowledgments

We would like to thank the reviewers for their feedback, as well as the jurisdictions where TAF is deployed for helping bring this paper and TAF to life.

This material is based upon work supported by the United States National Science Foundation (NSF) and Department of Education under Grant Nos. 2247829 and P200A210096. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

References

[1] California Legislative Information, "California legislative information," 2025. [Online]. Available: https://leginfo.legislature.ca.gov

[2] California Office of the Attorney General, "Publications - california department of justice," 2025. [Online]. Available: https://oag.ca.gov/publications

[3] New York State Office of the Attorney General, "Publications library," 2025. [Online]. Available: https://ag.ny.gov/libraries-documents/publications

[4] Montana Legislative Services Division, "Montana code annotated archive," 2025. [Online]. Available: https://archive.legmt.gov/bills/mca/index.html

[5] Uptane Project, "Uptane: Securing software updates for automobiles," 2025. [Online]. Available: https://uptane.org

[6] OpenBSD Project, "Signify: Cryptographically sign and verify files," 2025. [Online]. Available: https://man.openbsd.org/signify

[7] Sigstore Project, "Sigstore: Signing software releases," 2025. [Online]. Available: https://www.sigstore.dev

[8] W. N. Eskridge Jr, "Dynamic statutory interpretation," *U. Pa. L. Rev.*, vol. 135, p. 1479, 1986.

[9] V. F. Nourse and J. S. Schacter, "The politics of legislative drafting: A congressional case study," *NYuL Rev.*, vol. 77, p. 575, 2002.

[10] "Git," https://git-scm.com.

[11] S. Torres-Arias, A. K. Ammula, R. Curtmola, and J. Cappos, "On omitting commits and committing omissions: Preventing `git` metadata tampering that (re)introduces software vulnerabilities," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug 2016, pp. 379–395. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/torres-arias

[12] Uniform Law Commission, "Uniform electronic legal material act (uelma)," https://www.uniformlaws.org/committees/community-home?communitykey=02061119-7070-4806-8841-d36afc18ff21, 2011, accessed May 2025.

[13] R. Ottis, "Analysis of the 2007 cyber attacks against estonia from the information warfare perspective," in *Proceedings of the 7th European Conference on Information Warfare*. Academic Publishing Limited Reading, MA, 2008, p. 163.

[14] V. Napetvaridze and A. Chochia, "Cybersecurity in the making–policy and law: a case study of georgia." *International & Comparative Law Review/Mezinárodní a Srovnávací Právní Revue*, vol. 19, no. 2, 2019.

[15] E. Caroscio, J. Paul, J. Murray, and S. Bhunia, "Analyzing the ransomware attack on dc metropolitan police department by babuk," in *2022 IEEE International Systems Conference (SysCon)*. IEEE, 2022, pp. 1–8.

[16] A. Applebaum, *Iron Curtain: The Crushing of Eastern Europe, 1944–1956*. New York: Doubleday, 2012.

[17] P. Kenez, *The Birth of the Propaganda State: Soviet Methods of Mass Mobilization, 1917–1929*. Cambridge: Cambridge University Press, 1985.

[18] A. Roberts, *Blacked Out: Government Secrecy in the Information Age*. New York: Cambridge University Press, 2006.

[19] D. E. Pozen, "Deep secrecy," *Stanford Law Review*, vol. 62, no. 2, pp. 257–339, 2010.

[20] The Update Framework (TUF), "The update framework (tuf)," https://theupdateframework.io, 2025, accessed: 2025-05-17.

[21] GitHub, "About github," https://github.com/about, 2024, accessed May 2025.

[22] ——, "100 million developers and counting," https://github.blog/news-insights/company-news/100-million-developers-and-counting/, 2023, accessed May 2025.

[23] 18F, "18f github repositories," https://github.com/18F, accessed May 2025. 18F is part of the U.S. General Services Administration and maintains open source tools and documentation for government digital services.

[24] U.S. Digital Service, "U.s. digital service github repositories," https://github.com/usds, accessed May 2025. The USDS publishes digital government resources on GitHub as part of their mission to improve federal services.

[25] Code for America, "Code for america github repositories," https://github.com/codeforamerica, accessed May 2025. Code for America partners with governments to build and maintain civic technology solutions on GitHub.

[26] Office of Management and Budget, "Federal source code policy: Achieving efficiency, transparency, and innovation through reusable and open source software," https://code.gov/about/policy, 2016, accessed May 2025. This policy requires federal agencies to make custom-developed code available for reuse and encourages publishing on open platforms.

[27] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full sha-1," in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I 37*. Springer, 2017, pp. 570–596.

[28] J. Cappos, S. Baker, J. Plichta, D. Nyugen, J. Hardies, M. Borgard, J. Johnston, and J. H. Hartman, "Stork: package management for distributed VM environments," in *The 21st Large Installation System Administration Conference, LISA'07*, 2007.

[29] J. Cappos, J. Samuel, S. Baker, and J. Hartman, "A look in the mirror: Attacks on package managers," in *The 15th ACM Conference on Computer and Communications Security (CCS '08)*. New York, NY, USA: ACM, 2008, pp. 565–574.

[30] T. Kuppusamy, V. Diaz, and J. Cappos, "Mercury: Bandwidth-effective prevention of rollback attacks against community repositories," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX Association, 2017.

[31] T. K. Kuppusamy, S. Torres-Arias, V. Diaz, and J. Cappos, "Diplomat: Using delegations to protect community repositories," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 567–581.

[32] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable Key Compromise in Software Update Systems," in *The 17th ACM Conference on Computer and Communications Security (CCS '10)*. ACM, 2010.

[33] "Fuchsia software update system," https://fuchsia.dev/fuchsia-src/concepts/packages/software_update_system.

[34] "Mirantis secure registry," https://www.mirantis.com/software/mirantis-secure-registry/.

[35] "Ibm cloud docs: Signing images for trusted content," https://cloud.ibm.com/docs/services/Registry?topic=registry-registry_trustedcontent#registry_trustedcontent.

[36] "tough: a rust client library for tuf repositories," https://github.com/awslabs/tough.

[37] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th*

ACM conference on Computer and communications security. ACM, 2010, pp. 61–72.

[38] E. Wobber, M. Abadi, M. Burrows, and B. Lampson, "Authentication in the Taos operating system," *ACM Transactions on Computer Systems (TOCS)*, vol. 12, no. 1, pp. 3–32, 1994.

[39] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A calculus for access control in distributed systems," *ACM Trans. Program. Lang. Syst.*, vol. 15, no. 4, pp. 706–734, Sep. 1993. [Online]. Available: http://doi.acm.org/10.1145/155183.155225

[40] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 265–310, Nov. 1992. [Online]. Available: http://doi.acm.org/10.1145/138873.138874

[41] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 164–173.

[42] M. Blaze, J. Feigenbaum, and M. Strauss, *Financial Cryptography: Second International Conference, FC '98 Anguilla, British West Indies February 23–25, 1998 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ch. Compliance checking in the PolicyMaker trust management system, pp. 254–274. [Online]. Available: http://dx.doi.org/10.1007/BFb0055488

[43] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "Keynote: Trust management for public-key infrastructures," in *Security Protocols*. Springer, 1999, pp. 59–63.

[44] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "RFC 2693: SPKI certificate theory," 1999, https://tools.ietf.org/html/rfc2693.

[45] ——, "SPKI certificate theory," IETF RFC 2693, September, Tech. Rep., 1999.

[46] N. Li, J. Feigenbaum, and B. N. Grosof, "A logic-based knowledge representation for authorization with delegation," in *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*. IEEE, 1999, pp. 162–174.

[47] N. Li, "Delegation logic: A logic-based approach to distributed authorization," Ph.D. dissertation, New York University, 2000.

[48] N. Li, B. N. Grosof, and J. Feigenbaum, "A Nonmonotonic Delegation Logic with Prioritized Conflict Handling," https://www.cs.purdue.edu/homes/ninghui/papers/old/d2lp.pdf, 2000.

[49] DC Council, "DC Council Legislative Information Repository," https://github.com/DCCouncil/law-xml, 2025, accessed: 2025-05-07.

[50] https://theupdateframework.github.io/specification/latest/.

[51] "Building blocks and pades baseline signature," European Telecommunications Standards Institute.

[52] "Additional pades signatures profiles," European Telecommunications Standards Institute.

[53] "Pades document time-stamp digital signatures (pades-dts)," European Telecommunications Standards Institute.

[54] "Document management — portable document format," International Organization for Standardization.

[55] The Electronic Code of Federal Regulations, "ecfr :: 45 cfr 46.102," https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-A/part-46/subpart-A/section-46.102.

[56] D. S. H. Rosenthal, "Lockss: Lots of copies keep stuff safe," http://www.dlib.org/dlib/november05/rosenthal/11rosenthal.html, 2005, d-Lib Magazine, November 2005.

[57] L. Program, "How lockss works," 2024, accessed: 2025-05-07. [Online]. Available: https://www.lockss.org/use-lockss/how-lockss-works

[58] DSpace Project, "Dspace open source repository software," 2024, accessed: 2025-05-07. [Online]. Available: https://dspace.org

[59] Archivematica Project, "Archivematica digital preservation system," 2024, accessed: 2025-05-07. [Online]. Available: https://www.archivematica.org/en/

[60] Preservica, "Preservica: Active digital preservation," 2024, accessed: 2025-05-07. [Online]. Available: https://preservica.com

[61] "Repository service for tuf," https://github.com/repository-service-tuf/repository-service-tuf.

[62] A. S. A. Yelgundhalli, P. Zielinski, R. Curtmola, and J. Cappos, "Rethinking Trust in Forge-Based Git Security," in *32nd Network and Distributed System Security Symposium (NDSS 2025)*. San Diego, CA: Internet Society, 2025. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/rethinking-trust-in-forge-based-git-security/

[63] J. C. Hamano, "Git 2.29 release notes," https://github.com/git/git/blob/v2.29.0/Documentation/RelNotes/2.29.0.txt, Oct. 2020, introduces experimental support for SHA-256 object format.

[64] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1393–1410. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

APPENDIX A
ARTIFACT APPENDIX

This artifact appendix contains information on the artifact evaluation for The Archive Framework (TAF).

A. Description & Requirements

1) How to access: The artifact for TAF is made up of two parts: the source code for TAF itself, and the scripts which run the experiments for TAF. We have provided two methods to obtain the necessary files. Note that the evaluation, by default, installs TAF from PyPI, but you may manually install TAF from source if you so wish. from here:

Option I, Zenodo

If you wish to obtain the permanently-available copy of the evaluation, it is available on Zenodo: https://zenodo.org/records/17819921.

There are two `.zip` folders: one for the source code of `TAF`, and one for the evaluation scripts, titled `taf-ndss-eval`.

Option II - GitHub

Should you wish to use GitHub instead, you must clone the evaluation repository at https://github.com/renatav/taf-ndss-eval.

TAF's source code is available at https://github.com/openlawlibrary/taf/releases/tag/v0.36.0, should you wish to build from or examine the source.

2) Hardware dependencies: None.

3) Software dependencies: You must install Python 3.8 to 3.12 on your machine, as well as the pip package manager.

4) Benchmarks: None.

B. Artifact Installation & Configuration

We provide two options for getting the evaluation set up: a Dockerfile and manual installation instructions.

Option A, Docker

The evaluation repository contains a `Dockerfile`. Simply run the following commands to build and load the Docker container. You do not need to set up a virtual environment or install any packages with this method.

Run the following commands:

1) `docker build -t taf-ndss-eval .`
2) `docker run -it taf-ndss-eval`

The scripts will be in the current directory of the Docker container when started.

Option B, Manual Installation

After you install a supported Python version (see above), follow these steps to ready the artifact evaluation:

1) Create a new Python virtual environment and activate it.
2) Install TAF by running `pip install taf==0.36.0`.

C. Experiment Workflow

We have packaged each experiment as its own Python file. Each script will walk through the commands that are being run and display the results to the screen. Due to the relative paths used in the evaluation scripts, **you must be in the `scripts/` directory when invoking the security scripts and in the `performance/` directory when invoking the performance scripts**.

D. Major Claims

- (C1): TAF prevents unauthorized repository updates from deceiving clients. Experiments E1 through E7 demonstate this.
- (C2): Unsigned, outdated, or rollback metadata is detected and rejected by TAF. Experiments E3, E5, and demonstrate this.
- (C3): Rollback and replay attacks are prevented through version checks. Experiments E4 through E6 demonstrate this.
- (C4): TAF withstands even partial key compromise, preventing bypassing threshold signing requirements. Experiment E7 demonstrates this.
- (C5): TAF is performant and scales with repository size. Experiment E8 demonstrates this.

E. Evaluation

1) Experiment (E1): [Malicious Update to Target Repository Only] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 1`.

[Results]

In this experiment, an attacker compromises the credentials of a target repository, gaining commit and push access but not control over the authentication repository. They push a malicious update to `law-html`.

When the user runs the updater with default settings, no new commits are fetched because the authentication repository remains unchanged. The publisher runs the updater with a full check and detects the malicious commit, receiving a validation error.

2) Experiment (E2): [Malicious Commit + Authentication Repo Modified (No Signatures)] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 2`.

[Results]

In this experiment, an attacker obtains credentials that allow commit and push access to both a target and the authentication repository, but does not have access to any metadata signing keys. The attacker modifies `law-html` and pushes a malicious update, then manually updates the file recording the last valid commit for that target repository in the authentication

repository and pushes the change. They do not update or sign the corresponding TUF metadata.

When the user or publisher later runs the updater, it detects that the authentication repository has changed and fetches the new metadata. Validation fails because the updates are unsigned, and the user's local repository remains unchanged.

3) Experiment (E3): [Malicious Commit + Metadata Modified (Unsigned)] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 3`.

[Results]

In this experiment, an attacker obtains credentials that grant commit and push access to both the target and authentication repositories, but has not compromised any metadata signing keys. The attacker modifies `law-html` and pushes a malicious update. They then update the file recording the last valid commit for that target and correctly update the TUF metadata (the attacker is familiar with TUF), but cannot produce valid signatures of those updates. The attacker pushes these unsigned changes to the authentication repository.

When the user or publisher runs the updater with default settings, the updater detects the authentication repository update and fetches the incoming changes for validation. Validation fails because the metadata updates are unsigned, and the user's local repository remains unchanged.

4) Experiment (E4): [Repository Rollback via Force Push] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 4`.

[Results]

In this experiment, an attacker obtains credentials that grant commit and push access to both the target and authentication repositories, but does not possess any metadata signing keys. The attacker reverts all repositories to a previous commit and force-pushes the branches, attempting to make an outdated version appear current.

When the user or publisher runs the updater with default settings, the updater detects that the top commit of the remote authentication repository is not present in the local history and halts the update. This prevents the rollback attack from being accepted as a valid state.

5) Experiment (E5): [Metadata Reuse from Older Signed Commit] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 5`.

[Results]

In this experiment, the attacker gains push access to the authentication repository but does not have access to any metadata signing keys. Instead of creating new unsigned metadata, the attacker restores metadata from a previous commit that still contains valid signatures, attempting to make an older repository state appear current.

When the user or publisher runs the updater with default settings, the updater detects that the fetched metadata has a lower version number than the previously validated metadata. The update is rejected, and the rollback to an older repository state is successfully prevented.

6) Experiment (E6): [Publisher Pulls Invalid Update, Then Publishes Signed Metadata] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 6`.

[Results]

In this experiment, an attacker pushes a malicious but invalid update, similar to the one in Scenario 2. The publisher, without using TAF for validation, recklessly pulls these changes and subsequently creates and pushes a new valid, signed update.

When the user later runs the updater with default settings, the updater detects the new commits and begins validation from the oldest one. Because validation fails on the earlier invalid commit, the process halts even though the newest commit is valid.

7) Experiment (E7): [Compromised Snapshot + Timestamp Keys and Partial Root] [5 human-minutes]

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python3 run.py --scenario 7`.

[Results]

In this experiment, the attacker gains commit and push access to the authentication repository and compromises the snapshot and timestamp keys, both of which have a signing threshold of 1, as well as one of the root keys. They add their own targets signing key and use the compromised root key to sign the root, snapshot, and timestamp metadata. Root, however, has a signing threshold greater than one.

When the user runs the updater with default settings, validation fails because the root metadata is signed with only one key. This prevents the partially compromised root from being accepted and blocks the malicious update.

8) Experiment (E8): [Runtime Performance Evaluation] [5 human-minutes]: This experiment simulates TAF operations and the performance impact they have on various repositories.

[Preparation]

Ensure that the prerequisites for the evaluation are installed.

[Execution]

Run `python -m run.py`, inside the `performance` directory.

[Results]

This experiment's results summarize how each partner repository performs relative to the baseline. The tests were conducted on a machine more powerful than an average laptop, so slower performance should be expected on less capable hardware.