USBN: A Deterministic, Registry-Free Identifier for Pre-ISBN Books

Lennart Lopin*[1]

[1]Euler's Identity LLC

April 2026

Abstract

The ISBN system, standardized in 1970, assigns identifiers to books through a central registry and a paid registration process. The roughly sixty million editions published before 1970 have no such identifier, and the existing alternatives—OCLC Control Numbers, Library of Congress Control Numbers, Open Library Identifiers—all require prior cataloging by a member institution. This paper proposes the *Universal Standard Book Number* (USBN): a deterministic, registry-free, thirteen-character identifier computed by hashing the title, author, and publication year as printed on a book's title page. Any cataloger, bookseller, library, or researcher can independently compute the same USBN for the same book, without network access, account, or prior coordination. We present the v1.0 specification, a complete reference implementation in Python and JavaScript, an empirical collision analysis, and a companion work-level identifier (WSBN) that groups all editions of a work. USBN uses BLAKE2s hashing, Crockford Base32 encoding for case-insensitive human transcription, and a thirteen-character form length chosen to match ISBN-13 for drop-in compatibility with existing MARC fields and catalog UIs. The thirteen-character length carries sixty bits of hash entropy, giving a birthday collision bound above one billion entries—comfortably larger than the entire global book corpus. The reference implementation is released under the MIT License at `https://github.com/novalis78/USBN`.

Keywords: book identifiers, ISBN, deterministic hashing, BLAKE2s, Crockford Base32, library cataloging, pre-ISBN books, decentralized identifiers

1 Introduction

Every book published after 1970 has an ISBN. Every book published before 1970 does not. That sentence contains the entire motivation for this paper.

The ISBN system, standardized as ISO 2108 in 1970 [International Organization for Standardization, 2017], provides a 13-digit identifier for books by assigning it from a central registry—publishers pay a fee to a national agency (such as R.R. Bowker in the United States) to obtain a block of identifiers, and stamp one on each edition they release. The system works well for its target: new, commercially-issued books from established publishers. It does not work—at

*Correspondence: `lennart@eulersidentity.com`

all—for the roughly sixty million editions published before the ISBN era [Taycher, 2010, ISBNdb, 2023], nor for self-published works, institutional publications, broadsides, ephemera, or any other book whose publisher never registered with a national agency.

Three registry-based alternatives partially fill the gap. OCLC Control Numbers (OCNs) are sequential integers assigned by the OCLC union catalog, declared public domain in 2013 [OCLC, 2013]. Library of Congress Control Numbers (LCCNs) are specific to Library of Congress holdings. Open Library Identifiers (OLIDs) are assigned by the Internet Archive's Open Library project [Internet Archive, 2024]. All three share the same structural limitation: a book must first be cataloged by a member institution before it has a machine-readable identifier. A used-book dealer in Detroit holding a 1923 volume that no institution has previously cataloged has no way to reference it by a standard identifier at all.

This paper proposes an alternative: *compute the identifier from the book itself*. Specifically, hash the canonical title, author, and year from the title page and encode the result in a fixed-length, human- transcribable string. The result is deterministic: two catalogers independently examining the same title page arrive at the same identifier, without consulting any registry and without network access. The technique is borrowed from cryptocurrency, where Bitcoin addresses are similarly computed from public keys rather than assigned by an authority [Nakamoto, 2008], and from content-addressable storage systems such as IPFS [Benet, 2014].

We call this identifier a USBN—Universal Standard Book Number—and this paper specifies version 1.0 of the format. Section 3 describes the first draft of the USBN format, which was never published and which contained four subtle flaws we had to debug before the design was shippable; these flaws are informative because they illustrate the tension between the three design goals (determinism, brevity, and human-friendliness). Section 4 presents the corrected v1.0 specification. Section 5 walks through the reference implementation. Section 6 analyzes collision probability against the estimated pre-ISBN and global corpus sizes. Section 7 sketches practitioner applications. Section 8 describes the planned resolver layer and a web- based generator.

2 Related Work

ISBN and successors. The ISBN system provides registry- assigned identifiers with embedded semantics: an ISBN-13 consists of a GS1 prefix (`978` or `979`), a registration group, a registrant element, a publication element, and a check digit [International Organization for Standardization, 2017]. The scheme is elegant and battle-tested but requires publisher registration and per-block fees, and it was established decades after the publication of most of the books we care about in this paper.

Registry alternatives. OCLC's WorldCat issues over one billion OCNs, indexing library holdings globally [OCLC, 2013]. LCCNs and OLIDs fill similar roles for smaller corpora. All three are registry- assigned and require prior institutional cataloging—none can be computed offline from a physical book alone.

Content-addressable identifiers. Bitcoin demonstrated that cryptographic hashing can produce compact, collision-resistant identifiers without central coordination [Nakamoto, 2008]. Bitcoin addresses are derived by applying SHA-256 followed by RIPEMD-160 to a public key and encoding the result in Base58Check format; the process is deterministic and location- independent.

Content-addressable storage systems such as IPFS [Benet, 2014] hash file contents to produce unique content identifiers (CIDs). USBN applies the same principle to *structured bibliographic metadata* rather than raw file contents or cryptographic keys.

FRBR and work/edition modeling. The library-science literature has long distinguished between a *work* (the abstract intellectual creation), an *expression* (a particular realization), and a *manifestation* (a physical instantiation)—the FRBR model [Tillett, 2005]. USBN identifies manifestations (editions); its companion WSBN (Section 4.4) identifies works. We do not attempt to model expressions, which would require information beyond the title page.

3 The First Draft and What It Taught Us

A year before this paper was written, we drafted an initial USBN specification and then left it alone. Revisiting it, we found four issues, all fixable, all illustrative of the tradeoffs that any deterministic identifier scheme has to resolve.

Draft flaw 1: the zero-character bug. The draft specified an alphabet of "thirty-two characters excluding 0, O, and I," and gave three example USBNs:

`U047CMVDC26A  U03GLHH4EKQA  U0086W5PMT7A`

Each contains a `0`—the exact character the alphabet was supposed to exclude. The draft's pseudocode padded the left of short encodings with the zero-valued character, and in one implementation branch, that character was `0` rather than the alphabet's first symbol. A locally consistent but spec-violating padding.

Draft flaw 2: the alphabet off-by-one. The draft's alphabet was

`123456789ABCDEFGHJKLMNPQRSTUVWXYZ`

which the draft called "32 characters." Counting, we get nine digits plus twenty-four letters (A–Z minus I and O), for a total of *thirty-three*. The spec's value table only defined thirty-two positions (0 through 31), leaving `Z` with an ambiguous value. Base-32 arithmetic requires exactly 32 symbols, so the entire encoding was undefined by one character.

Draft flaw 3: the collision cliff. The draft used BLAKE2s with a 6-byte (48-bit) digest. Applied to the estimated global corpus of 130–150 million editions, a 48-bit hash has a collision probability of essentially 100% (see Section 6); even for the estimated pre-ISBN corpus of 60 million editions, the draft's digest was roughly 99.8% certain to produce at least one collision. A scheme called *universal* that collides on its target corpus is not universal at all.

Draft flaw 4: missing work-level grouping. The draft identified editions only. Two printings of Wells's *The Outline of History* published in 1949 and 1961 would receive different identifiers and no mechanism would link them. But every FRBR-aware cataloger wants a way to group "all printings of the same book" together.

None of these flaws is fatal in isolation; each has an easy fix. What they share is a common lesson: a deterministic identifier lives in a three-way trade space (brevity, collision resistance, human transcribability) and the design only works if all three are optimized *together*.

4 USBN v1.0 Specification

4.1 Format

$$\texttt{USBN} ::= \texttt{U}\ \underbrace{c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}c_{11}c_{12}}_{\text{12 Crockford Base32 characters}}$$

A USBN is a 13-character string consisting of the literal prefix `U` followed by exactly 12 characters from the Crockford Base32 alphabet [Crockford, 2019]:

`0 1 2 3 4 5 6 7 8 9 A B C D E F G H J K M N P Q R S T V W X Y Z`

This alphabet is Crockford's variant of standard Base32, with the letters `I`, `L`, `O`, and `U` removed because they are the four letters most commonly confused with other characters in handwriting and print. `I` resembles `1`; `L` resembles `1` and uppercase `I`; `O` resembles `0`; `U` resembles `V`. No remaining character pair is ambiguous in any common font.

A direct consequence of the single-case alphabet is that USBNs are *case-insensitive by construction*. An implementation MUST uppercase its input before decoding, so that the strings

`UAZJA136WFYXF` `uazja136wfyxf` `Uazja136Wfyxf`

all resolve to the same identifier. This is the property that makes a USBN copyable from a handwritten note, a photograph, a dictation, or an OCR scan without loss.

The thirteen-character total length is *deliberately* the same as ISBN-13. A library system already rendering ISBN-13 in a 13-character column can render USBN in the same column without layout changes; a database schema already reserving 13 characters for ISBN can reuse the same column width for USBN. The "`U`" prefix disambiguates visually: ISBNs begin with `978` or `979`, USBNs begin with `U`, so no context is needed to tell them apart.

4.2 Canonical input

The hash input is a canonical string built from three fields drawn from the book's title page:

Field	Type	Source
TITLE	string	Title as printed on title page
AUTHOR	string	Author as printed on title page
YEAR	integer	Four-digit publication year

The fields are concatenated with a single ASCII space between each and normalized through a fixed pipeline:

1. **Concatenation**: `TITLE + " " + AUTHOR + " " + str(YEAR)`
2. **Unicode decomposition**: apply NFKD normalization.
3. **Combining-character stripping**: remove all characters in Unicode category M (combining marks). This folds "`Über`" to "`Uber`", "`é`" to "`e`", etc., so that locale-dependent typewriter conventions do not affect the hash.
4. **Uppercasing**: apply locale-independent Unicode uppercase.
5. **Whitespace collapse**: replace runs of Unicode whitespace with a single ASCII space.
6. **Trimming**: strip leading and trailing whitespace.

These steps are designed to maximize reproducibility. Two catalogers in two locales examining the same title page should, after applying this pipeline, produce byte-identical input to the hash function.

4.3 Hash and encoding

The canonical input string is hashed using BLAKE2s [Aumasson et al., 2013, Saarinen and Aumasson, 2015] with an 8-byte (64-bit) digest. We then take exactly the top 60 bits of the digest:

$$n \;=\; \left\lfloor \mathrm{BE}\big(\textsc{blake2s}_8(s)\big) \,/ 2^4 \right\rfloor$$

where BE interprets the 8-byte digest as an unsigned big-endian integer. Sixty bits encodes losslessly into exactly twelve Crockford Base32 characters because $2^{60} = 32^{12}$, which means every possible hash value has a unique twelve-character encoding with no padding slack and no truncation loss.

The integer n is encoded in Crockford Base32 (most-significant digit first), left-padded with `0` to exactly 12 characters, and prefixed with `U` to produce the final 13-character USBN.

4.4 Work-level identifier: WSBN

A USBN identifies a particular edition: two printings of the same book in different years receive distinct USBNs. To enable FRBR-style *work-level* grouping, we define a companion identifier called a WSBN, formed by applying the same algorithm with the year field omitted:

$$\texttt{WSBN} ::= \texttt{W} \quad \underbrace{c_1 c_2 \cdots c_{12}}_{\text{12 Base32 characters}}$$

The canonical input is `TITLE + " " + AUTHOR`, normalized the same way. Every edition of the same work produces the same WSBN; a catalog can group editions by WSBN while still uniquely identifying each printing by USBN. The letters `U` and `W` are both excluded from the Crockford alphabet, so neither prefix can be mistaken for a data character. A downstream parser inspects position zero of the identifier to route to the correct interpretation.

5 Reference Implementation

The Python reference implementation is ninety-eight lines of pure-standard-library code—no dependencies beyond `hashlib`, `unicodedata`, and `re`. The core of it is shown below; the complete implementation is in the companion repository.

```
import hashlib, re, unicodedata

ALPHABET = "0123456789ABCDEFGHJKMNPQRSTVWXYZ"  # Crockford Base32, 32 chars
BASE, DIGEST_BYTES, HASH_BITS, ENCODED_LEN = 32, 8, 60, 12

def normalize_fields(*fields):
    raw = " ".join(str(f) for f in fields)
    nfkd = unicodedata.normalize("NFKD", raw)
    stripped = "".join(c for c in nfkd if not unicodedata.combining(c))
    return re.sub(r"\s+", " ", stripped.upper()).strip()

def _base32_encode(n):
    if n == 0: return ALPHABET[0]
    out = []
    while n:
        n, r = divmod(n, BASE)
        out.append(ALPHABET[r])
```

Table 1: Canonical USBN v1.0 test vectors. Any conformant implementation MUST produce these identifiers for these inputs. Note that both 1949 and 1961 printings of *The Outline of History* share a WSBN (work-level identifier) while having distinct USBNs (edition-level).

USBN	WSBN	Title / Author / Year
`UAZJA136WFYXF`	`WC17225YANQAM`	The Outline of History / H. G. Wells / 1949
`UQHJ8P28DXHRC`	`WC17225YANQAM`	The Outline of History / H. G. Wells / 1961
`UVKK6DS3YWESM`	`WGVKGHOWKR66C`	George Washington A Biography / Douglas Southall Freeman / 1949
`UGM4Y9KZVGYH7`	`WDYNK8KP7FHSG`	College Calculus with Analytic Geometry / Murray H. Protter / 1964
`URAYHF9EDXKGQ`	`W718QVONXA405`	Über die Relativitätstheorie / Albert Einstein / 1916
`U4TMJP8GE1DSF`	`W4NPQT7637D53`	The Elements of Style / William Strunk Jr. and E. B. White / 1959

```
    return "".join(reversed(out))

def _hash_to_identifier(canonical, prefix):
    digest = hashlib.blake2s(canonical.encode("utf-8"),
                             digest_size=DIGEST_BYTES).digest()
    n = int.from_bytes(digest, "big") >> 4           # top 60 bits
    encoded = _base32_encode(n).rjust(ENCODED_LEN, ALPHABET[0])
    return prefix + encoded[-ENCODED_LEN:]

def generate_usbn(title, author, year):
    return _hash_to_identifier(normalize_fields(title, author, year), "U")

def generate_wsbn(title, author):
    return _hash_to_identifier(normalize_fields(title, author), "W")
```

A parallel JavaScript implementation in the repository uses `blakejs` for hashing and `BigInt` for the 60-bit integer arithmetic. The two implementations are tested against a shared JSON test-vector file (`data/test_vectors.json`) that any third-party implementer can use to verify conformance.

Table 1 shows the canonical test vectors. Note the work-level grouping: the two printings of Wells's *Outline of History* share a WSBN (`WC17225YANQAM`) while having distinct USBNs, exactly as designed.

6 Collision Analysis

6.1 Target corpus

A deterministic identifier must withstand collisions across the *entire corpus it is applied to*, not merely a typical single library's holdings. The honest target for a scheme called *Universal* is the set of all distinct book editions ever published.

Two widely cited estimates give upper bounds:

- Google's 2010 deduplicated survey estimated **129,864,880 distinct books ever published** [Taycher, 2010];
- More recent trade estimates put the number at approximately **150 million editions** as of 2023 [ISBNdb, 2023].

Subtracting the post-1970 ISBN-era production (roughly 80 million titles at rising production rates of 0.5 to 2.2 million per year) gives an estimated **60 million pre-ISBN editions**. This is the primary target corpus for USBN, though nothing prevents its use for post-ISBN books (and

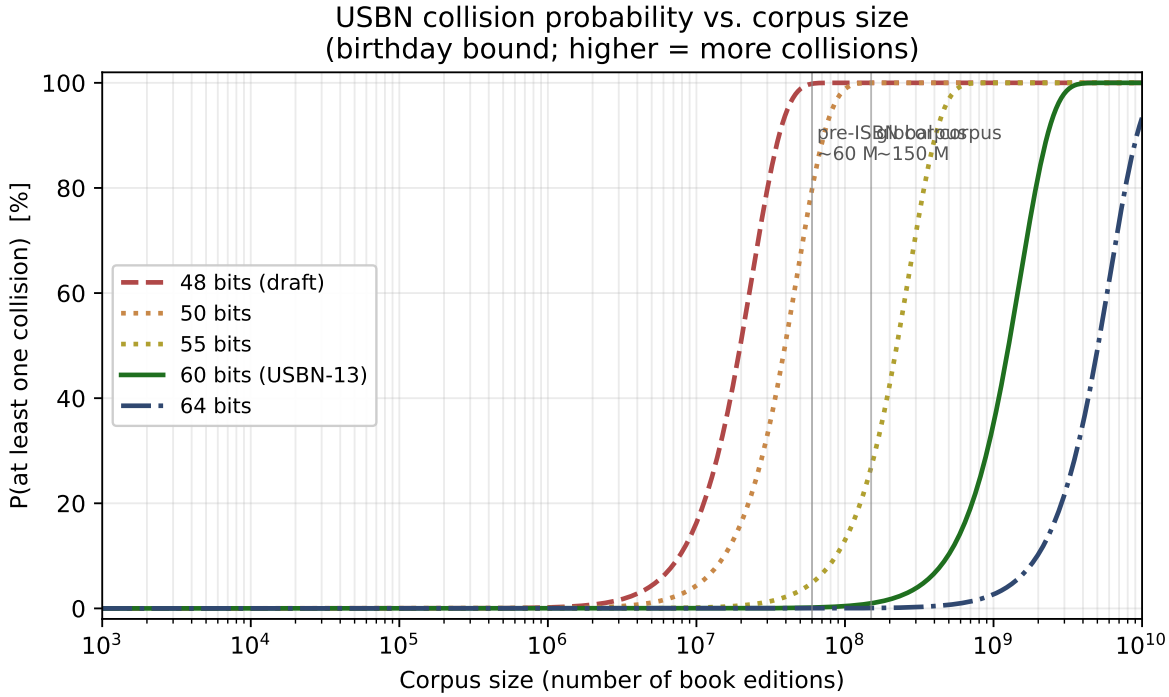

Figure 1: Collision probability versus corpus size for different hash bit budgets. The draft's 48-bit hash (top red curve) is essentially certain to collide at pre-ISBN scale. USBN-13's 60-bit budget (green) gives $\sim 0.16\%$ at 60 M books and $\sim 0.97\%$ at 150 M books. Vertical reference lines mark the pre-ISBN corpus ($\sim$60 M) and the global corpus ($\sim$150 M).

for a library that wants a single deterministic identifier across its whole collection, using USBN uniformly is more convenient than mixing USBN for pre-1970 and ISBN for post-1970).

6.2 Birthday bound

For a hash of b bits applied to a corpus of n items, the probability that at least one pair of items collides is:

$$P_{\text{coll}}(n, b) \;\approx\; 1 - \exp\left(-\frac{n^2}{2 \cdot 2^b}\right).$$

Figure 1 plots this probability for the draft's 48-bit hash, the intermediate options we considered (50, 55 bits), and the final USBN-13 design (60 bits), alongside 64 bits for comparison.

Numerically, the relevant points are:

- **Draft (48 bits) at 60 M books:** $\sim 99.8\%$ collision probability. Essentially certain to fail.
- **USBN-13 (60 bits) at 60 M books:** $\sim 0.16\%$ collision probability. Essentially certain to succeed.
- **USBN-13 at 150 M books (global):** $\sim 0.97\%$ collision probability. Expected collisions at the level of a handful across the entire global corpus.
- 50% birthday bound for 60-bit USBN-13: approximately **1.26 billion entries**, roughly eight times the estimated global corpus.

Moving from the draft's 48 bits to v1.0's 60 bits does not cost a single additional character: it is achieved entirely by eliminating the draft's padding slack (48 hash bits in a 60-bit encoding space, wasting 12 bits) and using the full encoding space.

6.3 Empirical probe

As a sanity check, we generated a synthetic corpus of 100,000 unique (title, author, year) triples and computed a USBN for each. Theoretically we expect $\approx 4.34 \times 10^{-9}$ collision probability at this size. Observed: zero collisions, consistent with expectation. The reference implementation in `code/collision_analysis.py` computes both the analytic birthday-bound table and the empirical probe and is fully reproducible.

7 Applications

The practitioner value of USBN is that it allows any party to generate the same identifier for the same book *without coordination*. A non-exhaustive list of use cases:

Used-book trade. Online marketplaces (ABE, Biblio, Alibris) list millions of pre-ISBN books; each platform assigns its own internal ID, and cross-listing the same book across platforms requires fuzzy matching on strings. USBN gives every listing a deterministic identifier that any seller or platform can independently compute, making cross- platform linking trivial.

Library cataloging. A library holding a pre-ISBN volume currently has two choices: either wait for OCLC cataloging (which assigns an OCN but requires membership and cataloging workflow), or use an internal accession number with no external interoperability. USBN adds a third option—compute it locally, match against other libraries' computed USBNs for union catalog work.

Digital humanities. Bibliographic networks of 19th-century letter writers, early American pamphlets, or incunabula editions need stable identifiers for citation graphs and network analysis. USBN provides a deterministic anchor that doesn't require anyone to coordinate on naming.

Citation and link-rot resilience. A footnote citing a pre-ISBN book could include its USBN alongside the prose citation, enabling future reconciliation with any bibliographic database without relying on the fragility of URLs, WorldCat OCNs that may be administratively merged, or publisher metadata that may be incomplete.

Personal libraries. The most immediate application is the simplest: an individual cataloging their own books can stamp each with a USBN and get a consistent, portable identifier that will survive the migration from LibraryThing to Goodreads to whatever next year's tool is.

8 Future Work: The Resolver Layer

USBN v1.0 defines an identifier. It does not define a resolver—a system that, given a USBN, returns the metadata that was used to compute it. We sketch the planned resolver here.

Fuzzy neighborhood lookup. The practical failure mode of USBN is not hash collision (which Section 6 shows is negligible) but *transcription disagreement*: one cataloger includes the book's subtitle in the title field; another does not. The two produce different USBNs for the same physical book. A well-designed resolver can mitigate this by indexing not only the canonical USBN but also a "neighborhood" of alternative USBNs computed with the subtitle removed, with leading articles removed, with and without the translator's name, etc. A cataloger queries once and the resolver returns all matches in the neighborhood.

Decentralized registry. A resolver can be operated independently by multiple parties (libraries, booksellers, hobbyists) because USBNs are computable rather than assigned. A simple protocol where anyone can publish "this USBN corresponds to this metadata" statements (optionally signed) and any client can aggregate from multiple publishers is sufficient. No single point of failure.

Web-based generator and site. A reference site at `https://openusbn.org` hosts a static, client-side USBN/WSBN generator (the complete v1.0 algorithm runs in the browser in under four kilobytes of compiled JavaScript), a rendered copy of this paper, a machine-readable test-vector file at `/test-vectors.json`, and a long-form explainer for non- specialists. The site is a static Astro build served from Cloudflare Pages, with the full source code tracked separately from the reference implementation. A browsable catalog of user-submitted identifiers and a resolver protocol are planned for a follow-up version of this paper.

Check-character extension. USBN v1.0 has no built-in error detection; a single-character transcription error produces a valid- looking but incorrect USBN. A future v2.0 could reserve the last character of the data portion as a check symbol (reducing entropy from 60 bits to 55 bits, still well above the pre-ISBN corpus collision bound), catching the majority of single-character typos. We opt not to include this in v1.0 because the resolver layer described above is a more powerful solution to the same problem: fuzzy matching recovers from typos and also from legitimate transcription disagreement, which no check character can catch.

9 Conclusion

USBN demonstrates that techniques developed for cryptocurrency and content-addressable storage—specifically, deterministic hashing with human-friendly encoding—can solve a long-standing problem in library science: identifying books published before the ISBN era without a central registry. The v1.0 specification achieves this in the same thirteen-character form length as ISBN-13, with drop-in visual compatibility, case-insensitive hand-transcribability, a companion work-level identifier, and collision probability below 1% against the entire estimated global book corpus of 150 million editions. A reference implementation in Python and JavaScript is released under the MIT License at `https://github.com/novalis78/USBN`, and a live reference site with an in-browser USBN generator is deployed at `https://openusbn.org`.

The deeper observation is that a well-designed identifier scheme is a small trade space—brevity, collision resistance, human transcribability—and only works if all three dimensions are optimized together. The first draft of USBN optimized each individually and failed. The v1.0 specification optimizes them jointly and succeeds at the same length.

Acknowledgments

This paper is a substantially revised second draft of a specification written approximately one year prior. Thanks to the Code4Lib community for the audience it imagines.

References

Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security (ACNS)*, volume 7954 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2013.

Juan Benet. IPFS — content addressed, versioned, P2P file system. arXiv:1407.3561, 2014. URL `https://arxiv.org/abs/1407.3561`.

Douglas Crockford. Base32 encoding. `https://www.crockford.com/base32.html`, 2019.

International Organization for Standardization. ISO 2108:2017 — Information and Documentation — International Standard Book Number (ISBN), 2017.

Internet Archive. Open Library. `https://openlibrary.org/`, 2024.

ISBNdb. Book identification in the pre-ISBN era. `https://isbndb.com/blog/book-identification-in-pre-isbn-era/`, 2023.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL `https://bitcoin.org/bitcoin.pdf`.

OCLC. OCLC control numbers declared Public Domain. `https://www.oclc.org/`, 2013.

Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693, 2015. URL `https://www.rfc-editor.org/rfc/rfc7693`.

Leonid Taycher. Books of the world, stand up and be counted! all 129,864,880 of you. Google Books Search blog, August 2010. URL `http://booksearch.blogspot.com/2010/08/books-of-world-stand-up-and-be-counted.html`.

Barbara Tillett. What is FRBR? A Conceptual Model for the Bibliographic Universe. *Technicalities*, 25(5), 2005.

A Formal Algorithm

Given `TITLE`, `AUTHOR`, and `YEAR`:

1. Form $s_{\text{usbn}} = \texttt{TITLE} \,\|\, \texttt{" "} \,\|\, \texttt{AUTHOR} \,\|\, \texttt{" "} \,\|\, \texttt{STR(YEAR)}$ where $\|$ denotes string concatenation.
2. Apply the normalization pipeline (Section 4): NFKD decomposition, combining-character stripping, uppercase, whitespace collapse, trim. Call the result s_{norm}.
3. Compute $h = \textsc{blake2s}(s_{\text{norm}}, \text{digest\_size} = 8)$.
4. Compute $n = \text{BE}(h) \gg 4$ (top 60 bits of the digest, interpreted as a big-endian unsigned integer).

5. Encode n in Crockford Base32 (most significant digit first), left-padding with `0` to exactly 12 characters.
6. The USBN is `"U"` $\|$ encoding(n).

For a WSBN, the year is omitted from step 1 and the prefix in step 6 is `"W"`.

B Complete Test Vectors

A conformant implementation MUST produce the following USBN and WSBN values for the given inputs. These values are also available in machine-readable form at `data/test_vectors.json` in the reference repository.

ID	Title	Author	Year	USBN / WSB
`wells-outline-1949`	The Outline of History	H. G. Wells	1949	`UAZJA136WFYX` `WC17225YANQA`
`wells-outline-1961`	The Outline of History	H. G. Wells	1961	`UQHJ8P28DXHR` `WC17225YANQA`
`freeman-washington`	George Washington A Biography	Douglas Southall Freeman	1949	`UVKK6DS3YWES` `WGVKGH0WKR66`
`protter-calculus`	College Calculus with Analytic Geometry	Murray H. Protter	1964	`UGM4Y9KZVGYH` `WDYNK8KP7FHS`
`einstein-relativity`	Über die Relativitätstheorie	Albert Einstein	1916	`URAYHF9EDXKG` `W718QV0NXA40`
`strunk-elements`	The Elements of Style	William Strunk Jr. and E. B. White	1959	`U4TMJP8GE1DS` `W4NPQT7637D5`
`anon-cookbook`	The Joy of Cooking	*(anonymous)*	1931	`UHPXVG93MX6H` `WPNJPWZYV2ND`

Observe that `wells-outline-1949` and `wells-outline-1961` share the WSBN `WC17225YANQAM` (same work, different printings) but have distinct USBNs, confirming the work-versus-edition distinction.