# BLAKE3-TOTP

One-Time Password Construction — Technical Specification

## Pipeline

Stage 1 Key norm. → Stage 2 Time counter → Stage 3 Keyed hash → Stage 4 Truncation → Stage 5 Code output

↓

Stage 6: Server acceptance policy (window size, rate limiting)

Stages 1–5 generate a code deterministically from a shared secret and the current time. Stage 6 is not part of code generation, but it dominates real-world security more than the hash choice does.

## 1 Stage 1 — Key normalization

BLAKE3 keyed mode requires an exact 32-byte key. The Base32-decoded enrollment secret may be any length, so it is passed through BLAKE3's key-derivation mode — a third native mode, distinct from plain hashing and keyed hashing — built for turning arbitrary-length key material into a fixed-size, context-separated key.

$$\text{raw\_secret} = \mathrm{Base32\_decode}(\text{enrollment\_secret})$$

$$\text{key} = \mathrm{BLAKE3\_derive\_key}(\text{context},\ \text{raw\_secret})$$

```
context = "google-authenticator-libpam 2026-08-16 TOTP key v1"
```

$$|\text{key}| = 32 \text{ bytes, always}$$

**Why not zero-padding?** Key derivation uses BLAKE3's key-derivation mode rather than zero-padding short secrets directly to 32 bytes:

$$\text{key}_{\text{old}} = \text{raw\_secret} \,\|\, \underbrace{0x00 \cdots 0x00}_{32-|\text{raw\_secret}|\ \text{bytes}}$$

Zero-padding is not a brute-force weakness on its own (160 bits of entropy is still enormous relative to the threat model), but it is an unnecessary structural property: every account's key would share an identical, predictable suffix, and BLAKE3 keyed mode has never been evaluated against related-key attacks under that assumption. Key derivation removes the structure entirely — every key is a full-entropy function of the input, and the context string decorrelates keys across accounts and uses.

## 2 Stage 2 — Time counter

Matches RFC 6238. Time is quantized into 30-second steps and represented as a big-endian 8-byte integer counter.

$$\text{STEP\_SIZE} = 30\text{s} \qquad T = \left\lfloor \frac{\text{unix\_time}}{\text{STEP\_SIZE}} \right\rfloor \qquad \text{counter\_bytes} = \text{big\_endian\_8\_bytes}(T)$$

A 20-second step was considered and rejected (§10): it shrinks the replay window by only $\sim 33\%$ while making the scheme more sensitive to client/server clock drift.

## 3 Stage 3 — Keyed hash

$$\text{digest} = \text{BLAKE3\_keyed}(\text{key},\ \text{counter\_bytes})$$

This replaces $\text{HMAC-SHA1}(\text{secret}, T)$ entirely — no inner/outer padding wrapper.

**Why no HMAC wrapper?** HMAC's ipad/opad construction exists to shield Merkle–Damgård hash functions (SHA-1, SHA-256, MD5) from *length-extension* attacks, by hiding the vulnerable intermediate chaining state behind two layers of hashing. BLAKE3 is a tree hash where the key is mixed directly into the root chaining value, so there is no exposed intermediate state to extend from. Wrapping BLAKE3 in HMAC would defend against a problem BLAKE3 does not have — the same reasoning that led NIST to define KMAC for SHA-3/Keccak instead of HMAC-SHA3.

## 4 Stage 4 — Dynamic truncation

Matches RFC 4226. Four bytes are extracted from the digest at an offset chosen by the digest's own final nibble, with the top bit masked off to avoid sign ambiguity across languages/platforms.

$$\text{offset} = \text{digest}[\text{len}(\text{digest}) - 1]\ \&\ \texttt{0x0F}$$

$$\begin{aligned}\text{truncated} = {} & \big(\text{digest}[\text{offset}]\ \&\ \texttt{0x7F}\big) \cdot 2^{24}\ +\ \text{digest}[\text{offset}{+}1] \cdot 2^{16} \\ & +\ \text{digest}[\text{offset}{+}2] \cdot 2^{8}\ +\ \text{digest}[\text{offset}{+}3]\end{aligned}$$

This algorithm is digest-length-agnostic, so it applies whether BLAKE3 outputs 20, 32, or more bytes.

## 5 Stage 5 — Code output

$$\text{CODE\_DIGITS} = 8 \qquad \text{code} = \text{truncated} \bmod 10^8 \qquad \text{(display as 8 digits, zero-padded)}$$

8 digits is RFC 6238's maximum. 12 digits was considered and rejected (§10).

**Verification requirement.** The server must compare the generated code against user input using a **constant-time comparison** (e.g. `crypto_memcmp` / `timing_safe_equal` -equivalent), never `==` or `memcmp`.

## 6 Stage 6 — Server-side acceptance policy

Not part of code generation, but the parameter set that governs real-world attack surface more than any cryptographic choice above.

$$\text{WINDOW\_SIZE} = 3$$

$$\text{RATE\_LIMIT} = \text{mandatory} \Rightarrow \text{fail closed if absent}$$

WINDOW_SIZE $= 3$ means the server accepts codes for $T-1, T, T+1$ — roughly $\pm 90$s, about 7 simultaneously valid codes once clock-skew tolerance is folded in.

**Rate-limit enforcement (fail-closed).** A secret with no `RATE_LIMIT` value is treated as misconfigured and denied, the same way an invalid `RATE_LIMIT` value is rejected. Provisioning must always set a `RATE_LIMIT` value by default; a compatibility "disable" mode should use an explicit high-but-finite limit rather than omitting it — so "no limit present" unambiguously means misconfigured, never intentionally disabled.

## 7 Conceptual security analysis

TOTP's real requirement is narrower than "the hash is strong": given many observed $(T, \text{code})$ pairs, an attacker without the key must not predict code for an unseen $T$ better than random guessing. This is a pseudorandomness/unpredictability property, not collision resistance.

| **Attack model** | **Applies?** | **Verdict** |
|---|---|---|
| Key recovery from $(T, \text{code})$ transcript | Yes — core threat | **Sound.** Relies on BLAKE3_keyed as a PRF under known-output queries. |
| Forgery of code for unseen $T$ | Yes — core threat | **Sound.** truncated $\bmod 10^8$ reveals $\sim 27$ bits per query out of a $\sim 256$-bit digest; PRF assumption $\Rightarrow$ outputs across $T$ look independent. |
| Length-extension | Structural (Merkle–Damgård only) | **Not applicable.** BLAKE3 keyed mode exposes no intermediate chaining state. |
| Related-key attacks | Many accounts share derivation scheme | **Addressed, defense-in-depth.** Key derivation's context string decorrelates keys; no known attack was being fixed. |
| Multi-target brute force (any of $N$ accounts) | Yes | **Accepted limitation** — inherent to fixed-digit OTP, not fixed here. |
| Preimage / 2nd-preimage / collision | Not required by TOTP | **Out of scope by design.** BLAKE3 exceeds SHA-1 here regardless. |
| Quantum (Grover) | Generic symmetric-primitive concern | **Not meaningful.** $256 \to 128$-bit effective security, vastly beyond the rate limiter's few-guesses-per-window model. |

**Net read:** of seven attack models, only three (length-extension, related-key, collision/preimage) are actually affected by the SHA-1→BLAKE3 change, and all three resolve favorably. The other four are determined by rate-limiting policy or are inherent to any fixed-digit OTP scheme — consistent with Stage 6 dominating real-world security more than hash choice.

# 8 Accepted limitations

**Multi-target brute force.** An attacker willing to compromise *any one* of $N$ accounts (not a specific one) effectively gets an $N\times$ larger guessing budget against the "some account succeeds" event, since `RATE_LIMIT` throttles attempts per account, not across accounts on a shared server:

$$P(\text{some account compromised}) \;\approx\; 1-(1-p)^N \qquad \text{for } N \text{ accounts, per-account success prob. } p$$

This is a property of every fixed-digit-count OTP scheme — 6, 8, or 12 digits, SHA-1 or BLAKE3. Mitigation requires a cross-account/per-source control (e.g. rate-limiting by source IP), outside this specification's scope.

# 9 Summary: parameter choices

| Parameter | RFC 6238 baseline | This design | Reasoning |
|---|---|---|---|
| Hash | HMAC-SHA1 | BLAKE3 keyed + `derive_key` | Modernization; HMAC wrapper unneeded, key derivation avoids shared-suffix issue |
| STEP_SIZE | 30s | 30s (unchanged) | 20s only shrinks replay window $\sim 33\%$, worsens clock-drift tolerance |
| CODE_DIGITS | 6 | 8 | RFC 6238 max; more fights the rate limiter for no gain |
| WINDOW_SIZE | 3 | 3 (unchanged) | Standard TOTP acceptance window |
| RATE_LIMIT | opt-in, off by default | mandatory | Closes the real brute-force path |

# 10 Rejected alternatives

- **12-digit codes.** $\bmod 10^{12}$ gives a trillion-value space, but `RATE_LIMIT` throttles to a few guesses per 30s regardless of digit count — the rate limiter, not the digit space, is the binding constraint. More digits only add typing/memorization burden.
- **20-second time step.** Shrinks the replay window by only $\sim 33\%$ relative to 30s while increasing sensitivity to client/server clock drift.
- **HMAC-BLAKE3.** A well-defined generic composition, but redundant — HMAC protects against length-extension attacks BLAKE3's native keyed mode is already immune to by design.

# 11 Implementation checklist — exact constants

Every value below must match exactly when reimplementing this specification in another language — none of them are "reasonable defaults."

1. **Secret encoding.** The distributed secret is a **Base32 string, RFC 4648 alphabet, no padding**. Before decoding: strip all whitespace, uppercase every character. Use a standard RFC 4648 Base32 decoder (not Base32hex, not Crockford Base32).
2. **Context string, verbatim, including capitalization and spacing:**

   `"google-authenticator-libpam 2026-08-16 TOTP key v1"`
3. **Key derivation.** `key = BLAKE3_derive_key(context, raw_secret)`. Output is exactly **32 bytes**. Use BLAKE3's key-derivation mode specifically — not plain BLAKE3 hashing, not keyed BLAKE3.
4. **Time counter.** `STEP_SIZE = 30` (seconds, unsigned 64-bit). `T = unix_time / STEP_SIZE`, using **integer (floor) division** on non-negative integers. `counter_bytes` is `T` encoded as **8 bytes, big-endian, unsigned 64-bit**.
5. **Hash.** `digest = BLAKE3_keyed(key, counter_bytes)` using BLAKE3's **keyed hashing mode** (the second native mode, not key derivation again). Output is the **standard 32-byte BLAKE3 digest** — do not use an extended-output (XOF) length.
6. **Dynamic truncation**, applied to those exact 32 bytes, identical to RFC 4226:

$$\text{offset} = \text{digest}[31] \;\&\; \texttt{0x0F}$$

$$\begin{aligned}\text{truncated} = \big(\text{digest}[\text{offset}] \;\&\; \texttt{0x7F}\big) \cdot 2^{24} + \text{digest}[\text{offset}{+}1] \cdot 2^{16} \\ + \text{digest}[\text{offset}{+}2] \cdot 2^{8} + \text{digest}[\text{offset}{+}3]\end{aligned}$$

7. **Code.** `CODE_DIGITS = 8`. `code = truncated` $\bmod 10^8$, displayed as **8 digits, left-zero-padded** (e.g. 423087 becomes 00423087).
8. **Time source. UTC seconds since the Unix epoch**, no leap-second adjustment. Clock skew is tolerated only by Stage 6's window (WINDOW_SIZE $= 3$, i.e. $T{-}1, T, T{+}1$ are all accepted).

### Worked test vector

Use this known-answer vector to check a reimplementation before attacking a real published secret.

| | |
|---|---|
| `raw_secret` (20 ASCII bytes) | `"12345678901234567890"` |
| same secret, Base32 (RFC 4648, no padding) | `GEZDGNBVGY3TQOJQGEZDGNBVGY3TQOJQ` |
| `unix_time` | 59 |
| `STEP_SIZE` / `CODE_DIGITS` | 30 / 8 (defaults) |
| $T$ | $59/30 = 1$ |
| **expected code** | 60153087 |

If an implementation produces 60153087 for this input, its BLAKE3 key derivation, counter encoding, keyed hash, truncation, and modulo/formatting are all correct.

## Compatibility note

This construction is **not** compatible with RFC 6238. It only interoperates with a client implementing this exact specification — standard authenticator apps (Google Authenticator, Authy, 1Password, etc.) hard-code SHA-1/256/512 per the RFC and cannot produce or verify BLAKE3-based codes.