Can Two 30-Second Codes Replace Your Password? A Feasibility Study of Double-TOTP Passwordless Login

Two independent 8-digit time-based one-time passwords in a single 30-second window: the brute-force math, the restart rule that makes the second layer worth having, and an honest account of where the design breaks.

Stephan Botes · Cyber Sec · cybersec.org.za

Companion article: cybersec.org.za/article-passwordless-double-totp.html · Every table in this report reproduces offline with the commands in Availability.

ABSTRACT

Passwords are the raw material of account takeover: users choose them weakly, reuse them widely, and hand them to phishing pages wholesale, so most intrusions begin with a credential the attacker should never have had. Industry analysis consistently finds that multi-factor authentication would have prevented the overwhelming majority of these compromises — Microsoft attributes 99.9% of Azure Active Directory account-compromise attacks to missing or absent MFA. That observation motivates a radical simplification examined here: delete the password entirely and authenticate with the factor that actually defends the account, twice.

The scheme under study is a login built from two independent layers of 8-digit, 30-second time-based one-time passwords (TOTP) — two enrollment seeds, two authenticator instances, one validity window — bound together by a single interaction rule: *fail anywhere, start over*. We formalize the online guessing model, derive the expected time to compromise for one and two layers (347 days at the NIST attempt cap for a single 8-digit layer; roughly 950,000 years for the pair under the restart rule; roughly 316,000 years when the verifier tolerates one time step of clock drift), and show why the restart rule — not the digit count — is the load-bearing wall: without it, online security collapses from the square of the code space back to a single layer, and a correct first-layer guess becomes a stay-in ticket. We then map the failure surface a feasibility verdict must include: two seeds on one device fall in one shot and NIST does not count repeated factors as true MFA; recovery inherits every weakness of password reset; real-time phishing relays can still relay both codes inside a window; 8-digit rendering is not universal in the authenticator ecosystem; and the throttling that defeats guessing is itself a denial-of-service lever. A comparison against password+TOTP and FIDO2/WebAuthn positions the scheme honestly: a strong, deployable step toward passwordless login for estates that cannot roll passkeys everywhere — with passkeys as the destination, not the rival. The brute-force math is airtight; the assurance story is a provisioning problem.

1 Introduction

Strip away the exotic techniques in the breach reports and the pattern underneath is boring: an attacker logged in with a credential they should not have had. The credential is usually a password — chosen weakly, reused across systems, or typed into a page that only resembled the real one. Microsoft's analysis of Azure Active Directory traffic put a number on the defense: multi-factor authentication would have stopped 99.9% of account-compromise attacks [6]. Read carefully, that statistic says something sharper than "deploy MFA." It says the password itself contributes almost nothing to the defense of the account. The factor that does the work is the second one.

That reading suggests the simplification this study tests: if the password is the part that fails, stop using it. Replace the password entirely with a second copy of the factor that actually defends — a login consisting of two independent TOTP layers and nothing else. The result is passwordless in the strict sense (no long-lived, human-typed, resettable secret anywhere in the primary flow) while remaining inside the TOTP ecosystem that most organizations already run for MFA. No new client platform, no hardware standards work, no certificate authority: authenticator apps and existing verification back ends are the entire deployment surface.

Feasibility is a precise question, not a vibe. It has three parts, and this report answers each in its own section:

1. **Does it resist online guessing?** §4 derives the expected compromise time under the standard model — bounded attempts per window, uniform code space — for one layer, two layers with the restart rule, and two layers with a drift-tolerant verifier. The headline numbers: 347 days for a single 8-digit layer at the NIST attempt cap; ~9.5×10^5 years for the pair; ~3.2×10^5 years with one step of drift tolerance.
2. **What does the second layer actually buy?** The answer is "almost nothing" unless a specific interaction rule holds. §3.2 isolates that rule — fail anywhere, restart from the username — and §3.3 shows the two quieter attacks (layer-state oracle, replay) that the rule removes for free.
3. **Where does it break?** Resistance to brute forcing is not assurance. §5 walks the five gaps that remain after the math: factor collapse on a shared device, recovery as the weakest link, real-time phishing relay, the 8-digit rendering gap in the authenticator ecosystem, and throttling turned into a lockout weapon.

The verdict, stated early and defended in §9: against the threat of online credential guessing, the scheme is feasible unambiguously — the numbers are geological. As a *replacement* for the password's full social role, it is feasible with asterisks: it is possession-only authentication, its strength depends on how seeds are provisioned, and its recovery path is the old password-reset problem wearing new clothes. It is best read as a strong intermediate point on a roadmap whose end state is phishing-resistant authentication.

2 Background and related work

2.1 Passwords as the attack surface

Three decades of incident data reduce password failure to a handful of mechanisms. **Weak choice:** the most common passwords are still, year after year, sequences of digits and first names, and entropy meters do not change that. **Reuse:** a breach at a low-value site becomes a credential-stuffing run against high-value sites, because users recycle secrets they cannot remember. **Spraying:** a small dictionary of popular passwords tried against thousands of usernames defeats per-password lockouts. **Phishing:** the user is the interface, and a convincing replica of the login page harvests the secret at scale. All four mechanisms share one property: they need a long-lived secret that a human can type. Remove that artifact and the entire class of attacks loses its raw material — which is exactly the move this scheme makes.

The cost of keeping passwords is not evenly distributed. Verifier-side defenses (hashing, pepper, breach-corpus rejection [5]) raise the price of database theft but do nothing about reuse or phishing, because those attack the password as it is typed, not as it is stored. The residual risk concentrates in the channel between the human and the verifier — which is why the 99.9% figure is about MFA, not about password policy [6].

The economics of the stolen-credential market close the loop. Breach corpora measured in billions of username-password pairs are freely circulated; the attacker's marginal cost per stuffing attempt is a fraction of a cent; and success does not require beating any particular site's defenses, only finding the sites whose users reused a breached secret. This is why password problems scale so badly: the attack cost curve rides other sites' breaches. A scheme with no typed secret to harvest, no hash to crack offline, and no reuse across sites — which double-TOTP is — opts out of that market rather than competing in it.

2.2 TOTP mechanics

Time-based one-time passwords are standardized in RFC 6238 [4] on top of the HMAC-based one-time password algorithm HOTP of RFC 4226 [3]. HOTP computes HMAC over an 8-byte counter with a shared secret key (typically 160 bits), applies the standard dynamic truncation to extract a 31-bit integer, and reduces it modulo 10^d to produce a d-digit code. TOTP replaces the counter with T = floor((unix time − T0) / X), where the reference implementation fixes X = 30 seconds and the standard's own test vectors use 8-digit codes.

Three properties of the construction matter for this study. First, the code space is exactly 10^d: eight digits give 100,000,000 values — about 26.6 bits, far below any offline cracking threshold, which is why TOTP security lives entirely in the *online* constraint (few attempts, short window) rather than in the code length. Second, the verification state is tiny: the verifier stores the seed, the current window index, and (per RFC 6238) a record of the last successfully validated code to reject reuse. Third, resynchronization: RFC 6238 permits a verifier to accept codes from neighboring time steps — it recommends accepting at most one step backward — which costs a factor in the guessing math (§4.4) and is a policy dial, not a fixed property.

The construction itself is worth one further glance, because every number in §4 stands on it. The 31-bit dynamic truncation selects 4 bits of the HMAC output as an offset and reads 31 bits from there; reduction modulo 10^d then spreads the value essentially uniformly over 0–10^d–1 (10^d does not divide 2^{31}, so the bias is below one part in two hundred for d=8 — negligible against a/10^d odds of 10^{-6}). The seed is the real secret: 160 bits of HMAC key, provisioned as base32, ten times the entropy of the code it generates. All attacker-visible entropy lives in the code, and all real entropy lives in the seed — which is precisely why guessing attacks are windowed (the code side) and database theft is catastrophic (the seed side), and why a paper about the former must still say what happens when the latter leaks (§5.2, §7).

2.3 What NIST SP 800-63B requires of verifiers

SP 800-63B [5] is the reference for what a verifier must do around any memorized or possessed secret. The provisions that carry this design: verifiers SHALL implement a rate-limiting mechanism that effectively limits consecutive failed attempts (the document's stated cap is 100); verifiers SHALL NOT permit unlimited guessing; throttling feedback SHOULD tell the user how long to wait rather than merely failing; and lookup secrets or OTPs SHALL be single-use where the construction allows it. Separately, the assurance levels define two-factor authentication as combining *different* factor categories — a second instance of "something you have" is still single-factor assurance, however inconvenient it is to steal twice. That provision lands directly on this scheme in §5.1.

OWASP's authentication and OTP guidance [7,8] operationalizes the same ideas for practitioners: aggressive but humane throttling (single-digit attempts per window is a common operating point), generic failure messages, hashed storage of OTP material where feasible, and rejection of previously seen codes.

2.4 Related passwordless schemes

Passwordless authentication is a crowded design space. **Magic links / email codes** relocate the shared secret into a mailbox — the login is now only as strong as webmail, and the reset channel *is* the login channel. **SMS OTP** binds to a SIM, inherits carrier-level takeover (SIM swap), and is explicitly discouraged by NIST for authenticator use. **Push-based approval** replaces typing with tap-fatigue and MFA-bombing, mitigated by number matching. **FIDO2/WebAuthn passkeys** [2] are the cryptographic end state: a hardware-bound key pair, origin-bound by construction, with no shared secret to phish — the only mainstream factor that is phishing-*resistant* rather than phishing-*expensive*.

Double-TOTP occupies a deliberate position in this landscape: it is the strongest passwordless scheme that can be deployed with pure TOTP infrastructure — the authenticator apps and verification servers most estates already operate. Bonneau et al.'s comparison framework [1] judges schemes on usability, deployability, and security; double-TOTP keeps two of TOTP's three deployability advantages (no client platform change, no new vendor) while paying TOTP's usability bill twice, and buys a security increment that only exists if the interaction rule of §3.2 is enforced. The framework's reminder applies verbatim: no candidate replaces passwords on every axis at once, so the honest question is which axes a given estate needs.

2.5 Threat model, and what "feasible" means here

All claims in this report are relative to the following attacker, stated so it can be contested. **In scope:** an online attacker who can submit arbitrary codes to the real verification endpoint; observe timing and failure behavior; command many source IPs (§4.6); run real-time phishing relays against users (§5.3); and attack anything adjacent to the flow — recovery channels, enrollment ceremonies, support desks (§5.2). **Out of scope:** malware on the user's device (which reads seeds or codes directly and defeats most client-side factors); compromise of the verifier's seed database (a different, catastrophic problem analyzed qualitatively in §7); global passive network adversaries; and attacks on TLS itself.

"Feasible" then means three specific things, and nothing more: the login flow resists its in-scope online attackers by arithmetic (§4); the flow's rules are implementable with ordinary engineering and testable with ordinary assertions (§3, §6); and the residual risks are enumerated with the same rigor as the strengths (§5), because a feasibility verdict that only counts wins is marketing. The word is deliberately not "secure" — security is a property of a deployed system including its humans and its recovery path, not of a login equation.

3 Design

3.1 Two layers, one window

The building block is deliberately unexotic: standard TOTP per RFC 6238, 8-digit codes, the 30-second

default step. Enrollment creates two *independent* seeds per account — seed A and seed B — and provisioning should place them on separate devices (a phone and a hardware token, or two phones) for reasons that matter in §5.1. Every login then presents:

- **Layer one.** The username, then token A — 8 digits derived from seed A.
- **Layer two.** Token B — 8 digits derived from seed B, verified against the same 30-second window.
- **Grant.** A session starts only when both layers verify inside the window.

No layer is unusual on its own; a verification server that already does TOTP already contains every primitive. The design's security comes from two properties — the short validity window, and the interaction rule between the layers. The second is where an otherwise correct implementation silently fails.

```
verify-flow
# login flow — double-TOTP with full-restart rule
1. IDENTIFY    accept username   → open attempt, no c
2. LAYER ONE   verify token_a    (8 digits, 30 s step,
               fail → burn the attempt, restart at 1
3. LAYER TWO   verify token_b    (independent seed, sa
               fail → burn the attempt, restart at 1
4. SINGLE USE  consume both codes on every attempt — r
5. THROTTLE    per-account AND per-source caps, cool-d
6. GRANT       both layers passed → issue session, no
```

3.2 The restart rule: fail anywhere, start over

The rule is one sentence: *if any layer fails, the entire attempt dies and the user starts over from the username with two fresh tokens.* A stale or previously seen code is never carried forward. This is not pedantry; it is the load-bearing wall of the design.

Consider the counterexample. Suppose the verifier allows an attacker who submits a correct layer-one code to remain "in" and continue guessing layer two within the lifetime of that layer-one acceptance. The moment that is permitted, the second layer stops adding anything: the online security of the login collapses from the square of the code space back to a single 8-digit code, because the expensive part — guessing layer one — is paid once and then reused for unlimited layer-two attempts. Every number in §4.3 is conditional on this never happening. The restart rule is what makes "two codes" into "two layers."

3.3 What the rule removes for free

Full restart buys more than the squaring. It removes the **layer-state oracle**: a flow that reports "layer one passed, layer two failed" confirms to the attacker both that the username exists and that a live seed produced a correct code seconds ago. That halves the attacker's work (their layer-one search is over), enables account enumeration, and converts the login form into a TOTP-validity oracle. Under full restart, every failure is identical, restarts at the username, and carries no information beyond "not granted."

It also enforces **single use** as a side effect. Both codes must be correct inside the same 30-second window, so any token captured by an attacker is worthless once the window closes — and the verifier additionally rejects any previously seen code after *any* attempt, successful or not. RFC 6238 requires rejecting reuse after successful validation; this scheme hardens that into rejection after every attempt, which closes the race where a legitimate validation and an attacker replay share a window.

3.4 Enrollment and seed provisioning

Two provisions decide the scheme's real-world assurance. **Separation:** seeds should be enrolled on distinct devices by default — phone plus hardware OATH token is the reference split — because both factors are "something you have," and only physical separation makes stealing both harder than stealing one (§5.1). **Depth:** seed material should be provisioned over an authenticated channel, displayed as QR plus base32 string, and accepted only over TLS; enrollment-time generation should use a CSPRNG per RFC 4226's requirements, since the 160-bit seed is the only high-entropy object in the system. The human-visibility rule from TOTP practice carries over: a user should be able to re-enroll a lost device only through the recovery path of §5.2 — never through the login flow itself.

A minimal enrollment ceremony makes the provisions concrete. The user authenticates once over the existing trusted path (or is enrolled in person at provisioning desks for staff estates); the verifier generates both seeds server-side from a CSPRNG and registers them; the two seeds are then transferred to *two different* devices — the ceremony is not complete until both confirm, each by returning one live code, from different hardware. Both-confirm-from-two-devices is the enrollment-time twin of the login rule: it makes separation a verified state rather than a policy aspiration, and it is the last moment the seeds exist outside encrypted storage on the verifier side.

3.5 Drift policy

Clock drift is the practical tax on any time-based scheme. The RFC's own guidance bounds it: a verifier may accept at most one time step of drift to resynchronize a slightly fast or slow clock. The design keeps that bound and nothing more: ±1 step maximum, drift detection recorded per seed (so a chronically drifting device can be flagged and re-enrolled), and — critically — no permanent widening of the acceptance window as a user-retention tactic. §4.4 prices the drift allowance: three codes live at once, the attacker's odds triple, and 9.5×10^{5} years becomes roughly 3.2×10^{5}. That is a cost worth paying once; compounding it is how square roots get taken back.

3.6 Sessions and re-authentication

Sixteen digits per login is heavy, and pretending otherwise is how schemes die in rollout. Friction is a design input, not an afterthought: long-lived sessions with device binding, risk-based step-up (re-authentication on new network, new device, or sensitive action), and deliberately infrequent full logins are what make double-TOTP livable. The throttling that defends the flow is also a denial-of-service lever (§5.5), so cool-downs follow the NIST usability provision — tell the user how long, per account *and* per source — rather than opaque lockouts.

3.7 A worked round

The rule is easiest to trust concretely. Suppose an attacker with one valid username runs a guessing rig against the flow, 3 attempts per window (a strict verifier), and gets lucky enough to guess a correct layer-one code on cycle 40,001 — about 14 days of continuous guessing in.

- **Window N, cycle 40,001.** Token A is correct on the first guess of the window. Token B is wrong twice; the third guess burns the last attempt of the window. **The attempt dies.** Under a carry-over design the attacker would now own a half-open door and could keep trying B — $10^{8}/3$ windows, ~32 years, at zero further cost on layer A. Under the restart rule the door closes.
- **Window N+1, cycle 40,002.** The attacker is back at layer one. Fresh codes for A are required — the guess that succeeded seconds ago is dead, consumed by the failed attempt. Nothing about the 14 days survives: the odds for cycle 40,002 are the same 9×10^{-16} as for cycle 1.
- **Every window after.** Memorylessness (§4.7) is not a rhetorical flourish here; it is the attacker's actual predicament. Fourteen days of work bought one 30-second window in which both codes had to be guessed — and the second guess missed.

The same round shows what the uniform-failure rule hides from the attacker: at no point did the flow say "layer one passed." The attacker cannot know their luck was spent — only that nothing was granted. That is one rule doing three jobs: squaring the space, closing the oracle, and burning the evidence of progress.

4 The brute-force math

4.1 The online guessing model

The model throughout is online guessing: the attacker submits candidate codes to the real verifier and observes granted/not-granted. The assumptions, stated so they can be attacked: codes are uniform on the code space (HMAC truncation makes this true to within a negligible bias); the verifier enforces an attempts-per-window bound a (NIST's stated cap: 100 [5]); each window is 30 seconds; and the restart rule of §3.2 holds so no progress survives a failed cycle. The seed itself is out of reach — 160 bits of HMAC key is not brute-forceable, and the verifier's storage of seeds is a database-theft problem (§5.2, §7), not a guessing problem.

Under the model, each window is an independent round in which the attacker gets a draws from 10^{8} values. The probability of success in one window is $a/10^{8}$, the number of windows to success follows a geometric distribution with expectation $10^{8}/a$, and expected compromise time is $(10^{8}/a) \times 30$ s. An 8-digit code carries 26.6 bits — below the 64-bit floor NIST sets for anything without mandatory rate limiting, which is why throttling is the design's foundation, not an optimization.

4.2 Single layer

TABLE 1 — SINGLE 8-DIGIT LAYER: ATTEMPTS PER 30-SECOND WINDOW

ATTEMPTS / WINDOW	ODDS PER WINDOW	EXPECTED TIME
100 (NIST cap)	1×10^{-6}	~1 year (347 days)
10	1×10^{-7}	~9.5 years
3 (strict)	3×10^{-8}	~32 years

A single 8-digit layer at the NIST cap is measured in months-to-years — respectable against opportunistic

attack, thin against a patient one, and the same order as a good password policy's *claimed* strength with none of the phishing exposure. The lesson of the table is the shape of the curve, not the rows: expected time scales as 1/a, so the operator's throttling dial is the entire security margin of one layer. Six digits would start the same table at one million values — 3.5 days at the cap (§4.5) — which is why the 8-digit choice is not cosmetic.

4.3 Two layers with the restart rule

With the restart rule in force, a failed cycle preserves no state: the attacker returns to layer one with nothing banked. Breaking the pair requires a window in which *both* codes are guessed, and the odds per full cycle are the product of the per-layer odds — the code space effectively squares. Expected time becomes $(10^8/a)^2 \times 30$ s, and each cycle burns another 30 seconds whether or not layer one succeeded.

TABLE 2 — DOUBLE 8-DIGIT LAYER WITH RESTART: ATTEMPTS PER LAYER PER CYCLE

ATTEMPTS / LAYER / CYCLE	ODDS PER CYCLE	EXPECTED TIME
100 each	1×10^{-12}	~950,000 years
10 each	1×10^{-14}	~95,000,000 years
3 each	9×10^{-16}	~1 billion years

The squaring is the whole argument, and it deserves its precondition printed next to the result: *the product holds only while both layers are live in the same window and no partial success survives.* Permit carry-over of a passed layer (the counterexample of §3.2) and the table collapses to Table 1. Permit ten steps of drift instead of one and the effective space shrinks by the same factor as the drift. The math is airtight because the rule makes it airtight — every digit of those exponents is a property of the interaction discipline, not of the codes.

4.4 The cost of clock drift

Accepting one step of clock drift for resynchronization — the RFC's recommended maximum allowance — puts three codes in play at once (t−1, t, t+1), multiplying the attacker's per-cycle odds by three. At the NIST cap the expected time drops from roughly 950,000 years to roughly 316,000; at strict throttling, from a billion to a third of a billion. Online brute forcing is dead either way — that is the point — but the arithmetic discipline is to treat drift allowance as spending: one step is the price of operational sanity, and more is a discount sold to attackers. The design records detected drift per seed and re-enrolls chronic offenders rather than widening the window permanently (§3.5).

4.5 Digit-count sensitivity

TABLE 3 — DIGIT COUNT VS EXPECTED COMPROMISE TIME (100 ATTEMPTS PER LAYER, RESTART RULE)

DIGITS	CODE SPACE	SINGLE LAYER	DOUBLE LAYER
6	10^6	3.5 days	~95 years
7	10^7	34.7 days	~9,500 years
8	10^8	~1 year	~950,000 years

Each added digit multiplies the single-layer margin by ten and the double-layer margin by a hundred. Six digits — the ecosystem default — buys 95 double-layer years: comfortable against opportunism, uncomfortable as a design target, and the reason this study specifies eight. Seven digits is the defensible compromise where 8-digit rendering is unavailable (§5.4); six digits should be considered only with strict throttling and a short window, and said so plainly.

4.6 What throttling buys — and what it costs

Every number above is a function of *a*, so the rate limiter is the actual security kernel of the scheme. Two consequences. First, *a* must be enforced per account **and** per source: an account-scoped cap alone lets a botnet put thousands of attempts across distributed sources, which turns a=100 per account into a much larger effective a. Second, throttling is an attack surface in its own right — an attacker who deliberately fails logins can cool down or lock out the legitimate user (§5.5). The mitigations (per-source caps, cool-down messaging, unlock flows that do not leak account state) are not hardening extras; they are part of the security claim.

The distributed case deserves its own paragraph, because it is where implementations most often misplace the cap. Suppose the operator caps per source at 100 attempts per window but sets no per-account cap, and the attacker commands m = 10,000 machines. The effective attempts per window is $m\times100 = 10^6$, and a single 8-digit layer falls in an expected $10^8/10^6 = 100$ windows — about fifty minutes. The per-account cap is the binding constraint: with a per-account cap of A, total attempts per window cannot exceed A no matter how many sources join, and the double-layer expected time stays $(10^8/A)^2 \times 30$ s. Distributed guessing changes the

attacker's bill, not the defender's odds — provided the account-level number exists and is small.

4.7 Percentiles: the median is not the marketing

Expected values understate the tail that matters to a defender, because the geometric distribution is memoryless: an attacker who has already failed a billion cycles is exactly as far from success as one who started today. The quantiles of the number of cycles to success are ln(1/q)/p for success probability q — a factor of ln(100)/ln(2) ≈ 6.6 between the median and the 99th percentile, and the same factor of order one forever after. The practical reading of Table 4: even the *lucky* tail of the attack distribution is civilization-scale for the double layer, and the median-vs-tail gap is narrow enough that quoting the mean is honest.

TABLE 4 — TIME TO SUCCESS BY PERCENTILE (100 ATTEMPTS PER LAYER, 30 S WINDOWS)

PERCENTILE	SINGLE LAYER	DOUBLE LAYER (RESTART)
50% (median)	240 days	~660,000 years
95%	~2.9 years	~2,900,000 years
99%	~4.4 years	~4,400,000 years

One more economic observation completes the picture: the online model prices the attacker's infrastructure per attempt — request, response, IP reputation — and none of the layers of Table 4 reward bulk infrastructure, because the per-account cap makes parallelism worthless (§4.6). The attacker's rational move is therefore to leave the login flow alone and attack what the model excludes: the seeds, the recovery path, or the human (§5). A scheme that reroutes attackers toward its weakest neighbor has not solved that neighbor's problem — which is why §5 exists.

4.8 The parameter set, with its reasons

TABLE 5 — DESIGN PARAMETERS AND WHAT FIXES THEM

PARAMETER	VALUE	FIXED BY
Code length	8 digits (10^8)	§4.5 — 6 digits gives 95 double-layer years; 8 gives ~10^4× that margin
Time step	30 s	RFC 6238 default; shorter steps cost usability, longer windows help the attacker
Layers	2, independent seeds	§4.3 — squaring; a third layer buys another 10^8 factor nobody needs
Attempts / layer / window	3–5 (cap 100)	§4.2, checklist item 1 — NIST ceiling is 100; the operating point is a strictness choice inside it
Drift allowance	±1 step, recorded	§4.4 — RFC maximum; each extra step divides expected time by 3
Code reuse	rejected after any attempt	§3.3 — RFC requires rejection after success; the rule hardens it
Session policy	long, device-bound	§3.6 — the friction budget that makes 16 digits livable

The table is the design in one place, and it doubles as an audit checklist: a deployment that differs from a row should be able to say why in the language of the right-hand column. Parameters chosen by taste are parameters chosen by whoever persuades the loudest; each row here is nailed to the section that computes its cost.

5 Threat assessment: how it breaks

The login flow survives online guessing overwhelmingly. A feasibility verdict, though, has to include the ways the scheme degrades in practice. Five gaps, in order of severity — each with what it costs the attacker, what it costs the defender, and the signal it leaves. Table 6 summarizes.

5.1 Two seeds can fall in one shot

Both factors are something you have — and NIST's assurance levels are blunt that repeated instances of the same factor do not constitute true MFA, while OWASP flags factor reuse as a known anti-pattern [5,7]. If both seeds live in the same authenticator app on the

same unlocked phone, one compromise of that app defeats both layers at once, and the 950,000-year number describes nothing. Enrollment on separate devices (phone plus hardware token) restores meaningful independence: the attacker now needs two thefts, ideally of differently-protected objects. Separation should be the enrollment default, not a deployment option — the scheme's brute-force story is airtight, but its assurance story is a provisioning problem.

5.2 Recovery becomes the weakest link

Removing the password does not remove account recovery — it moves the entire "reset your password" problem to "re-enroll your seeds." Email-based seed re-issuance is the new password reset: an attacker does not need to touch the beautiful login flow, only the recovery channel in front of it. The design response is to treat recovery with the same rigor as login: recovery codes generated at enrollment, stored hashed, single-use; re-enrollment gated by identity verification of the same strength as the flow it protects; and no recovery shortcut reachable from the login form itself. Whatever remains weakest after this — support-desk social engineering, the email account, the SIM — is now the account's real front door, and should be audited as such.

The recovery controls, stated as requirements. **Codes at enrollment:** a fixed set of single-use recovery codes, hashed with a slow KDF like the seeds themselves, displayed exactly once — the same trust budget as a seed, spent differently. **Rate-limited re-enrollment:** the recovery path gets its own per-account and per-source throttling, independent of the login flow's counters, so exhausting one says nothing about the other. **Delay and notify:** a re-enrollment request triggers user notification on every enrolled channel and imposes a fixed cool-down before taking effect — the window in which a victim who did not request it can veto the takeover. **No grading on the phone:** support desks get a script and no authority to shortcut the ceremony; the workflow that talked helpdesks into resetting passwords over the phone is the same one that will talk them into re-enrolling seeds over the phone.

5.3 Real-time phishing survives

TOTP codes are phishable. A real-time man-in-the-middle toolkit can proxy the genuine login page, relay a victim's live codes to the attacker as the victim types them, and complete the session inside the same 30-second window. The timeline is instructive: the victim types token A at second 6 and token B at second 19 of the window; the relay forwards both as they are typed; the attacker's session grants at second 24 — six seconds of margin. The 8-digit, two-layer design raises the bar — the attacker must relay both codes within one window, against single-use consumption that burns codes on every attempt — but raised is not removed. What the scheme does buy is signal: the typing connection arrives from the victim's IP while the granted session originates from the relay's, a mismatch that a datacenter-IP check or a session-IP-delta control can catch in-line. Passkeys and FIDO2/WebAuthn remain the only common factors that are phishing-resistant by construction (origin binding), not by friction [2]. A deployment that faces serious phishing should treat double-TOTP as a hardening step toward WebAuthn, not a terminus.

5.4 Eight digits are not universally rendered

Most mainstream authenticator apps display six digits only — Google Authenticator included. Deployable 8-digit-capable options exist (Aegis, 2FAS, FreeOTP+, Bitwarden, and OATH hardware tokens), but "bring your own authenticator" is a real rollout constraint, not a footnote: user education, an allow-list, and a documented 7-digit fallback (§4.5's ~9,500 double-layer years) belong in the deployment plan. The RFC is not the constraint — its test vectors are 8 digits — the ecosystem is.

The rollout reads differently by estate. **Staff/enterprise:** hardware OATH tokens make the 8-digit question procurement, not persuasion, and the separation requirement of §3.4 is naturally satisfied by issuing two distinct devices. **Consumer:** the allow-list does real work — the deployer names the four apps that render 8 digits and the enrollment UI walks users into them — and the honest alternative is deploying 7 digits rather than hoping users find an 8-digit app by themselves. **Mixed:** per-user digit length is a verifier-side column, not a code change; the math of §4.5 prices each user's actual margin.

5.5 Friction and lockout abuse

Sixteen digits per login is heavy; frequent full logins will make users resent it, and resentful users vote for the weakest available workaround. Long sessions and risk-based re-authentication (§3.6) are part of the design, not luxuries. Meanwhile the rate limiting that defeats brute forcing is itself a weapon: attackers spamming deliberate failures can cool down or lock out legitimate

users — a denial-of-service on the account. Per-source caps alongside per-account ones, cool-down messages that say how long, and unlock paths that do not leak account existence are the countermeasures; skipping them converts the scheme's strength into someone else's lever.

5.6 Where the scheme sits

TABLE 6 — ATTACK SUMMARY: WHAT EACH BREAK COSTS AND WHAT IT LEAVES BEHIND

ATTACK	PREREQUISITE	ATTACKER COST	DEFENDER SIGNAL	REF
Online guessing	none	~10^{12} cycles	failure storm per account+source	§4
Layer carry-over	verifier bug	one correct layer-A code	none — test the state machine	§3.2
Shared-device seed theft	one unlocked device	malware / brief access	new-session from new device	§5.1
Recovery-channel attack	weak re-enrollment	social engineering	re-enrollment event + notify	§5.2
Real-time relay	phish kit + live victim	one window per session	typing IP ≠ session IP	§5.3
Lockout abuse	username enumeration absent*	spam failures	failure storm, zero grants	§5.5

*Lockout abuse needs a valid username; the uniform-failure rule of §3.3 is what keeps usernames cheap to have and expensive to confirm. The table's shape is the verdict: the rows that remain cheap are the rows that touch people and provisioning, not arithmetic.

TABLE 7 — SCHEME COMPARISON

PROPERTY	PASSWORD + TOTP	DOUBLE TOTP	PASSKEY (FIDO2)
Long-lived typed secret	Yes — the root risk	None	None
Resists online guessing	Password-bound	~10^{5-9} years	Not applicable
Phishing-resistant	No	No (relay-costly)	By construction
True two-factor (NIST)	Yes (2 categories)	No — possession-only	Single-factor possession*
Seed/key theft blast radius	One seed	Two, unless separated	Device-bound
Recovery inherits reset risk	Yes	Yes	Yes (account recovery)
Ecosystem maturity	Universal	Niche (8-digit apps)	Growing fast

*FIDO2 with a PIN or biometric attestation satisfies multi-factor assurance under SP 800-63B; bare discovery-state passkeys are single-factor possession. The table's honest reading: double-TOTP eliminates the attack class passwords die to, but it does not reach WebAuthn's phishing resistance, and like every scheme it concentrates risk in recovery. It is a strong intermediate — the destination is still phishing-resistant authentication.

6 Implementation checklist

For implementers attempting this in anger, these are the requirements that separate the design from a vulnerable deployment. Each is stated with the failure it prevents.

1. **Throttle per account and per source.** No more than 100 consecutive failures per account (the NIST cap); 3–5 per window is a saner operating point. Add per-source caps for distributed guessing, and a cool-down that tells users how long to wait, per NIST's usability guidance. *Prevents:* Tables 1–2 collapsing; lockout DoS.
2. **Consume codes on every attempt.** Successful or failed — and reject any previously seen code. This is the restart rule expressed in state, and it is what makes captured codes worthless and the squaring of §4.3 true. *Prevents:* layer carry-over; replay.
3. **Uniform failure messages.** Every failure restarts at the username with the same generic error. No oracle,

no enumeration. *Prevents:* layer-state oracle (§3.3); account enumeration.

4. **Protect the seeds like keys.** Decrypt seed material only at verification time and re-encrypt immediately; HSM or hardware-backed storage where the deployment can afford it. Never log token values. *Prevents:* bulk seed theft, which converts every account into a live oracle forever (§7).
5. **Bound the drift window.** At most one time step, recorded per seed rather than widened permanently. *Prevents:* silent multiplication of the attacker's odds (§4.4).
6. **Hash at rest — and know what it buys.** Hashing OTPs does not defeat a database thief (a 27-bit code space falls instantly offline), but it limits the blast radius of a briefly exposed live window and is basic hygiene per OWASP's OTP guidance. *Prevents:* nothing magical; that is the point of stating it.
7. **Monitor and notify.** Alert users on failed cycles with time, source, and browser; every failure the user does not recognize is reconnaissance against them. *Prevents:* silent pre-attack mapping; gives users agency.

Each requirement is testable without attacking production, and the state machine at the center of the design — consume-on-every-attempt, restart-on-anything — is small enough to assert in a unit test. The reference behavior, in twelve lines:

```
# the restart rule, as a state machine — the invariant
seen = set()                          # codes consumed
def attempt(tok_a, tok_b, window):
    if tok_a in seen or tok_b in seen:
        return RESTART                # replay = resta
    seen ⊨ {tok_a, tok_b}             # consume on EVE
    ok = verify(seed_a, tok_a, window) \
         and verify(seed_b, tok_b, window)
    return GRANT if ok else RESTART   # any failure —

# assertions the deployment must keep green:
#   correct-a/wrong-b  → RESTART, and token_a is dead
#   wrong-a            → RESTART, token_b never evalu
#   both correct twice → GRANT then RESTART (single u
```

7 Threats to validity

The uniformity assumption. The model assumes codes are uniform on the code space. HMAC-SHA1/256 truncation holds this to within negligible bias for honest implementations; a biased RNG at *enrollment* time, however, would shrink the effective space below 10^8 and every table with it. The 160-bit seed, generated by a CSPRNG, is the object that makes the uniformity assumption safe — seed-generation review is part of the claim.

The verifier is assumed honest and correct. Every number is conditional on the checklist of §6 actually being enforced: a verifier that leaks layer state, carries forward passed layers, or widens drift under user pressure voids the corresponding section. This study analyzes the design; it does not audit any particular deployment, and the gap between the two is where TOTP deployments historically fail.

No user study, no field data. Friction claims (§3.6, §5.5) are engineering judgment, not measurement. Login-time overhead, mis-keying rates at 8 digits, cool-down resentment, and lockout frequency under real traffic are unknowns that a pilot should measure before an estate-wide rollout; the scheme's history-adjacent cousins (single-TOTP) suggest the friction is survivable but not free.

Online scope. The math is exclusively online guessing. It does not model seed database theft (which converts guessing into living off a stolen oracle), client-side malware reading the authenticator, enrollment-channel attacks, or the recovery path — each is addressed qualitatively in §5 and each is a real attack that the expected-time tables do not price. The seed-theft case deserves its bluntest statement: both seeds live in the same verifier database by design, so a database compromise defeats both layers at once regardless of device separation — encryption of seed material (checklist item 4) raises the bar from a file copy to a live-key compromise, and nothing in this scheme survives an attacker who holds decrypted seeds. That is not a weakness relative to password+TOTP (which stores one seed and a password hash behind the same database); it is the reason the scheme's claim is scoped to the online model and the reason HSM-backed storage appears in the checklist rather than in a footnote.

Falsifiability. What evidence would move these conclusions? A field pilot measuring lockout rates and login times above the friction budget would weaken the usability claims of §3.6 and §5.5; a demonstrated layer-carry-over in a real verifier would void §4.3's table at that deployment; a phishing campaign outperforming the relay-cost estimate would revise §5.3. The math itself — code space, attempts, window — is not empirical and stands or falls with arithmetic that any reader can re-run (Availability).

Assurance accounting. Under SP 800-63B the scheme is single-factor possession authentication, however many seeds it uses. Deployments that need multi-factor assurance for compliance, as distinct from security against guessing, need a second factor *category* — this scheme does not provide one.

8 Open problems

- **The 8-digit ecosystem gap.** Can mainstream authenticators be moved to configurable digit counts, or should the ecosystem standardize on 8? RFC 6238's test vectors are already 8-digit; the gap is purely client-side rendering (§5.4).
- **Separation attestation.** Is there a deployable way for a verifier to *require* that two seeds be provisioned to distinct hardware — attested at enrollment — rather than trusting policy? Without it, factor separation remains an honor system.
- **Relay telemetry.** Real-time phishing relays must complete both codes inside one window (§5.3). What anomaly signal does that timing create — two valid codes from one IP followed by session use from another geography seconds later — and can it be operationalized into detection rather than post-hoc forensics?
- **Recovery without passwords.** Every passwordless scheme inherits a recovery path that looks suspiciously like a password reset (§5.2). What does re-enrollment look like when it must be as strong as the login it recovers — identity verification, delay, notarized identity — and what does it cost in support load?
- **A formal carry-over bound.** The squaring argument of §4.3 is elementary but informal. A short proof that any partial-success carry-over reduces online security to the carried layer, under arbitrary verifier policies, would make the restart rule's status precise and give auditors a checklist item instead of a paragraph.

9 Conclusion

Feasible — against the stated threat, unambiguously. Online brute forcing of this login flow is not merely impractical; at strict throttling it is measured in geological time, and the restart rule — fail anywhere, start over — is what makes the second layer worth having. The design needs no new client platform, no new vendor, and no ceremony beyond what TOTP estates already run; its novelty is a discipline, and the arithmetic of §4 shows exactly what the discipline purchases.

As a password replacement the honest framing is narrower. Double-TOTP eliminates the long-lived typed secret that fuels credential stuffing, spraying, and reuse — but it is possession-only authentication under NIST's accounting; its seeds can fall to a single device compromise unless deliberately separated; its recovery path inherits every weakness the password reset used to have; and its codes remain relayable by real-time phishing. The pragmatic reading for an organization considering it: a strong, deployable step toward passwordless login for estates that cannot roll FIDO2 everywhere — with passkeys as the destination, not the rival. And whichever factor combination an estate deploys, the architecture lesson generalizes: bounded attempts, single-use consumption, and a flow that restarts from zero on any failure are what turn "two codes" into "two layers."

AVAILABILITY

Every number in Tables 1–3 reproduces offline from one calculation — code space ÷ attempts, times 30 seconds per window, squared for the double layer — with no network, no API key, and no dependencies:

```
python3 - <<'PY'
YR = 31_557_600          # Julian year, seconds
N  = 10**8               # 8-digit code space
for a in (100, 10, 3):   # attempts per layer per
    single = N/a * 30 / YR
    double = (N/a)**2 * 30 / YR
    print(f"a={a:>3}  single={single:,.0f} yr"
          f"  double={double:,.0f} yr")
# drift allowance (+/-1 step): three codes live a
print("drift:", f"{(N/100)**2*30/YR/3:,.0f} yr")
PY
```

References

1. J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. *IEEE Symposium on Security and Privacy*, 2012.
2. W3C. Web Authentication: An API for accessing Public Key Credentials (Level 2). W3C Recommendation, 2021.
3. D. M'Raihi, S. Machani, M. Pei, and J. Rydell. HOTP: An HMAC-based one-time password algorithm. RFC 4226, IETF, 2005.
4. D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-based one-time password algorithm. RFC 6238, IETF, 2011.
5. P. A. Grassi, M. E. Garcia, and J. L. Fenton (eds.). Digital Identity Guidelines: Authentication and Authenticator Management. NIST SP 800-63B, 2017; revised as SP 800-63B-4, 2024.

6. A. Weinert. Your pa$$word doesn’t matter. Microsoft Security Blog, 2019.
7. OWASP Foundation. Authentication Cheat Sheet. OWASP Cheat Sheet Series.
8. OWASP Foundation. Multifactor Authentication Cheat Sheet. OWASP Cheat Sheet Series.

Colophon. Typeset from the study’s own calculation source; every table in this report reproduces offline from the command listed under Availability. Companion web edition: cybersec.org.za/article-passwordless-double-totp.html. Negative results are kept in: the six-digit rows of Table 3 and the single-layer margin of Table 1 are included because they are the numbers against which the 8-digit choice should be judged. Threats to validity are in §7, not the footnotes.