TOTP FOR BEGINNERS

Santokh Singh Saggu

Time-Based One-Time Password (TOTP)

A Comprehensive Guide

Understanding, Implementing, and Securing

Two-Factor Authentication with TOTP

Santokh Singh Saggu

TOTP

Copyright 2025 , Santokh Singh Saggu , All rights Reserved

Santokh Singh Saggu

Table of Contents

1. Introduction to TOTP

Time-Based One-Time Password (TOTP) is a widely adopted algorithm for generating one-time passwords that are time-synchronized between a client and server. It represents one of the most popular implementations of two-factor authentication (2FA) and multi-factor authentication (MFA) used to secure user accounts across the internet.

TOTP was standardized in 2011 as RFC 6238, building upon the earlier HOTP (HMAC-Based One-Time Password) algorithm defined in RFC 4226. The key innovation of TOTP is its use of time as a moving factor, eliminating the need for synchronization counters between client and server.

1.1 Why TOTP Matters

In an era of increasing cybersecurity threats, passwords alone are insufficient protection. TOTP provides:

- Enhanced Security: Even if passwords are compromised, attackers cannot access accounts without the time-based token
- User-Friendly Authentication: Simple six to eight-digit codes that are easy to enter
- Offline Functionality: No internet connection required on the authenticator device
- Standardization: Interoperable across multiple platforms and services
- Cost-Effective: No specialized hardware required; works with smartphones

2. How TOTP Works

Understanding the technical foundation of TOTP is essential for proper implementation. The algorithm combines cryptographic hashing with time-based synchronization to generate secure, temporary codes.

2.1 The Core Algorithm

TOTP generates one-time passwords using the following formula:

$$\mathbf{TOTP = HOTP(K, T)}$$

Where:

- K = Shared secret key between client and server
- T = Time step calculated from current Unix time
- HOTP = HMAC-Based One-Time Password algorithm

2.2 Step-by-Step Process

Step 1: Calculate Time Step

The current Unix timestamp is divided by the time step period (typically 30 seconds):

```
T = floor(Current Unix Time / Time Step)
```

For example, if the current Unix timestamp is 1609459200 and the time step is 30 seconds:

```
T = floor(1609459200 / 30) = 53648640
```

Step 2: Generate HMAC

The time step T is converted to an 8-byte array (big-endian) and used with the shared secret key K to compute an HMAC-SHA1 hash:

```
HMAC = HMAC-SHA1(K, T)
```

This produces a 20-byte (160-bit) hash value that serves as the basis for code generation.

Step 3: Dynamic Truncation

To convert the 20-byte hash into a human-readable code:

1. Take the last 4 bits of the hash as an offset value (0-15)
2. Extract 4 bytes from the hash starting at that offset
3. Convert these 4 bytes to a 31-bit integer (masking the most significant bit)
4. Apply modulo 10^D where D is the desired number of digits (typically 6)

Step 4: Code Formatting

The resulting number is zero-padded to ensure it has exactly D digits. For a 6-digit code, values range from 000000 to 999999.

2.3 Key Parameters

TOTP implementations rely on several configurable parameters:

Parameter	Description
Secret Key (K)	Shared secret, typically 160 bits (20 bytes), encoded in Base32 for user convenience
Time Step (X)	Duration for which a code remains valid, default is 30 seconds
T0	Unix epoch start time (0), representing January 1, 1970 00:00:00 UTC
Digits (D)	Number of digits in generated code, typically 6 or 8
Algorithm	Hash function used, commonly SHA-1, SHA-256, or SHA-512

3. Implementation Guide

Implementing TOTP requires careful attention to both server-side generation and client-side validation. This chapter provides practical guidance for developers.

3.1 Server-Side Setup

Secret Key Generation

Generate a cryptographically secure random secret for each user:

```
import secrets
import base64

def generate_totp_secret():
    # Generate 20 random bytes (160 bits)
    secret_bytes = secrets.token_bytes(20)
    # Encode in Base32 for user-friendly display
    secret_base32 = base64.b32encode(secret_bytes).decode('utf-8')
    return secret_base32
```

QR Code Generation

Create a provisioning URI following the Key URI Format specification:

`otpauth://totp/{label}?secret={secret}&issuer={issuer}`

Example implementation:

```
from urllib.parse import quote
import qrcode

def generate_qr_code(secret, email, issuer):
    label = quote(f"{issuer}:{email}")
    uri = f"otpauth://totp/{label}?
secret={secret}&issuer={quote(issuer)}"

    qr = qrcode.QRCode(version=1, box_size=10, border=5)
    qr.add_data(uri)
    qr.make(fit=True)
    return qr.make_image(fill_color="black", back_color="white")
```

3.2 Code Verification

When validating user-provided codes, implement time window tolerance to account for clock drift:

```
import hmac
import hashlib
import time
import struct

def verify_totp(secret, user_code, window=1):
    '''
    Verify TOTP code with time window tolerance
    window: number of time steps to check (±1 = 3 total checks)
    '''
    secret_bytes = base64.b32decode(secret, casefold=True)

    for offset in range(-window, window + 1):
        time_step = int(time.time() / 30) + offset
        generated_code = generate_totp_code(secret_bytes, time_step)

        if generated_code == user_code:
            return True

    return False

def generate_totp_code(secret_bytes, time_step):
    # Convert time step to 8-byte big-endian
    time_bytes = struct.pack(">Q", time_step)

    # Generate HMAC-SHA1
    hmac_hash = hmac.new(secret_bytes, time_bytes, hashlib.sha1).digest()

    # Dynamic truncation
    offset = hmac_hash[-1] & 0x0F
    code = struct.unpack(">I", hmac_hash[offset:offset+4])[0]
    code = (code & 0x7FFFFFFF) % 1000000

    return f"{code:06d}"
```

3.3 Best Practices

5. Store secrets securely: Encrypt TOTP secrets in your database using application-level encryption
6. Implement rate limiting: Limit verification attempts to prevent brute force attacks (e.g., 5 attempts per 15 minutes)
7. Provide backup codes: Generate one-time recovery codes for account access if the TOTP device is lost
8. Use time window validation: Accept codes from ±1 time step (30 seconds) to handle clock drift

9. Prevent code reuse: Track and reject recently used codes within the same time window
10. Clear user communication: Provide clear setup instructions and troubleshooting guidance
11. Test thoroughly: Verify implementation against RFC 6238 test vectors

4. Security Considerations

While TOTP significantly enhances account security, proper implementation and user education are crucial to maintain its effectiveness.

4.1 Threat Model

Protections Provided

- Password compromise: Stolen or leaked passwords alone cannot grant access
- Phishing attacks: Time-limited codes reduce the window of opportunity for captured credentials
- Keylogging: Recorded passwords insufficient without current TOTP code
- Session hijacking: Attackers cannot establish new sessions without valid TOTP

Vulnerabilities and Limitations

- Man-in-the-middle attacks: Real-time interception can capture both password and TOTP code
- Malware on device: Compromised authenticator apps can leak secrets or codes
- SIM swapping: If TOTP is on the same device as SMS backup, both factors can be compromised
- Social engineering: Users can be tricked into providing current codes to attackers
- Physical theft: Stolen device with unlocked authenticator app grants access
- Backup vulnerabilities: Cloud backups of TOTP secrets may be targeted

4.2 Secret Key Management

Critical Security Practices:

12. Secure Generation: Use cryptographically secure random number generators (CSPRNG)
13. Encryption at Rest: Encrypt secrets in database with strong encryption (AES-256)
14. Secure Transmission: Only display secrets once during setup, preferably via QR code

15. Key Rotation: Provide mechanism for users to regenerate secrets if compromised
16. Access Control: Limit system access to encrypted secrets to authorized processes only
17. Audit Logging: Log all TOTP setup, verification, and reset activities

4.3 Time Synchronization

Accurate timekeeping is essential for TOTP functionality:

- Server Requirements: Use NTP (Network Time Protocol) to maintain accurate system time
- Tolerance Windows: Accept codes from ±1 time step to handle minor clock drift (±30 seconds)
- User Guidance: Advise users to enable automatic time settings on their devices
- Monitoring: Alert on repeated failures that might indicate time synchronization issues

4.4 Brute Force Prevention

Implement multiple layers of defense against brute force attempts:

18. Rate Limiting: Restrict verification attempts per user account (e.g., 5 attempts per 15 minutes)
19. Progressive Delays: Increase delay between allowed attempts after failures
20. Account Lockout: Temporarily lock accounts after excessive failed attempts
21. Monitoring and Alerting: Detect and alert on suspicious patterns of verification attempts
22. IP Reputation: Apply stricter limits to requests from suspicious IP addresses

5. User Experience and Adoption

Successful TOTP implementation requires balancing security with usability. Poor user experience leads to reduced adoption and workarounds that compromise security.

5.1 Setup and Onboarding

Recommended Setup Flow

23. Clear Introduction: Explain what TOTP is and why it enhances security
24. App Recommendations: Suggest authenticator apps (Google Authenticator, Microsoft Authenticator, Authy)
25. QR Code Display: Present scannable QR code with manual entry option as backup
26. Verification Test: Require user to enter a code immediately to confirm setup success
27. Backup Codes: Generate and display one-time recovery codes for emergency access
28. Save Confirmation: Require explicit confirmation that backup codes have been saved

Setup Instructions Template

Step-by-step guidance for users:

29. Download an authenticator app on your smartphone if you don't have one
30. Open the app and select "Add Account" or scan QR code
31. Scan the QR code shown on this screen
32. Enter the 6-digit code from your app to verify setup
33. Save your backup codes in a secure location

5.2 Troubleshooting Common Issues

Issue	Solution
Code doesn't work	Check device time synchronization, wait for new code, verify correct account selected
Lost phone	Use backup recovery codes, contact support for account recovery process
Can't scan QR code	Use manual entry option, provide

	secret key in readable format with spacing
New phone setup	Provide option to disable and re-enable TOTP with new device
Multiple devices	Same secret can be added to multiple devices; recommend cloud-backup authenticators

5.3 Account Recovery

Implement a secure account recovery process to handle device loss:

- Backup Codes: Provide 10-12 single-use recovery codes during TOTP setup
- Support Process: Establish verified identity process for TOTP reset (e.g., email verification + security questions)
- Delay Periods: Implement 24-48 hour waiting periods for account recovery to prevent social engineering
- Notification: Send alerts to registered email when TOTP is disabled or reset

6. Popular Authenticator Applications

Multiple authenticator apps support TOTP. Understanding their features helps users choose the right option for their needs.

6.1 Comparison of Major Authenticators

App	Platforms	Cloud Backup	Key Features
Google Authenticator	iOS, Android	Yes (Google account)	Simple, reliable, widely supported
Microsoft Authenticator	iOS, Android	Yes (Microsoft account)	Push notifications, password manager integration
Authy	iOS, Android, Desktop	Yes (encrypted)	Multi-device sync, desktop support
1Password	All platforms	Yes (subscription)	Integrated with password manager
Bitwarden	All platforms	Yes (free/premium)	Open source, password manager integration

6.2 Recommendations for Users

Basic Users:

- Google Authenticator or Microsoft Authenticator for simplicity and reliability
- Enable cloud backup to prevent lockout from device loss

Power Users:

- Authy for multi-device support and desktop access
- 1Password or Bitwarden for integrated password management

Security-Focused Users:

- Consider hardware security keys (YubiKey, Titan Security Key) as primary method
- Use TOTP as secondary backup authentication method

- Avoid cloud backup for maximum security

7. Advanced Topics

7.1 TOTP vs Other 2FA Methods

Method	Advantages	Disadvantages
TOTP	Works offline, no network required, standardized, free	Manual code entry, vulnerable to phishing, device dependency
SMS Codes	Simple, no app required, works on basic phones	SIM swapping risk, network dependent, delivery delays, interception
Push Notifications	Convenient, no code entry, can show login details	Network required, push fatigue vulnerability, platform-specific
Hardware Keys	Phishing-resistant, durable, no batteries, multi-protocol	Cost, physical possession required, loss risk, limited mobile support
Biometrics	Convenient, fast, difficult to steal	Privacy concerns, spoofing possible, irrevocable if compromised

7.2 TOTP Parameter Customization

While RFC 6238 specifies default parameters, organizations may customize them for specific security requirements:

Time Step Variations

- 30 seconds (default): Balances security and usability
- 60 seconds: Reduces user pressure, increases vulnerability window
- 15 seconds: Higher security, can frustrate users with entry delays

Code Length Considerations

- 6 digits (default): 1 in 1,000,000 codes, easy to enter, sufficient for most use cases
- 8 digits: 1 in 100,000,000 codes, higher security, slightly more cumbersome
- 4 digits: Not recommended due to reduced security (1 in 10,000)

Hash Algorithm Selection

- SHA-1 (default): Most compatible, sufficient security for TOTP use case
- SHA-256: Enhanced security, good compatibility
- SHA-512: Maximum security, limited authenticator support

7.3 Enterprise Deployment Considerations

Large-scale TOTP deployments require additional planning:

34. User Training: Develop comprehensive training materials and support documentation
35. Phased Rollout: Implement gradually, starting with IT staff and power users
36. Policy Enforcement: Define mandatory vs optional use cases
37. Support Infrastructure: Establish helpdesk procedures for TOTP issues
38. Compliance: Ensure implementation meets regulatory requirements (PCI-DSS, HIPAA, etc.)
39. Monitoring: Track adoption rates, failure rates, and support tickets
40. Integration: Coordinate with existing identity management systems (Active Directory, LDAP, SSO)

8. Testing and Validation

Thorough testing ensures TOTP implementation reliability and security.

8.1 RFC 6238 Test Vectors

The TOTP specification includes test vectors for validation. Using the secret key 12345678901234567890 (Base32: GEZDGNBVGY3TQOJQGEZDGNBVGY3TQOJQ):

Time (Unix)	Date/Time (UTC)	TOTP Code
59	1970-01-01 00:00:59	287082
1111111109	2005-03-18 01:58:29	081804
1111111111	2005-03-18 01:58:31	050471
1234567890	2009-02-13 23:31:30	005924
2000000000	2033-05-18 03:33:20	279037
20000000000	2603-10-11 11:33:20	353130

8.2 Testing Checklist

- Algorithm Verification: Validate against RFC 6238 test vectors
- Time Synchronization: Test with various time offsets (±30, ±60 seconds)
- Secret Handling: Verify encryption at rest and secure transmission
- QR Code Generation: Test with multiple authenticator apps
- Rate Limiting: Confirm brute force protections engage correctly
- Code Reuse Prevention: Verify same code cannot be used twice
- Backup Codes: Test recovery code generation and validation
- Edge Cases: Test time step boundaries, midnight rollover, leap seconds
- Cross-Platform: Verify functionality across browsers and devices
- User Interface: Test setup flow, error messages, and documentation clarity

9. Future Developments and Trends

Authentication technology continues to evolve. Understanding emerging trends helps organizations plan for the future.

9.1 WebAuthn and FIDO2

Web Authentication (WebAuthn) and FIDO2 represent the next generation of authentication standards:

- Phishing Resistance: Cryptographic authentication tied to specific domains prevents phishing
- Passwordless: Can replace passwords entirely, not just augment them
- Platform Integration: Native support in modern browsers and operating systems
- User Experience: Biometric authentication (fingerprint, face) provides seamless experience
- TOTP Coexistence: Many systems will maintain TOTP as fallback option during transition

9.2 Passkeys and Sync

Major technology companies are implementing synchronized passkeys:

- Apple, Google, and Microsoft now support passkey sync across devices
- End-to-end encrypted syncing prevents provider access to authentication credentials
- Eliminates device loss risk while maintaining security
- TOTP remains relevant for services not yet supporting passkeys

9.3 Quantum Computing Considerations

While TOTP's use of symmetric cryptography (HMAC-SHA1) is generally resistant to quantum computing attacks, organizations should consider:

- Transition to SHA-256 or SHA-512 for enhanced future-proofing
- Monitor NIST post-quantum cryptography standards
- Plan for eventual migration to quantum-resistant authentication methods

10. Conclusion

Time-Based One-Time Password (TOTP) has proven itself as a robust, reliable, and user-friendly method for implementing two-factor authentication. Its standardization through RFC 6238, wide support across platforms, and offline functionality make it an excellent choice for organizations seeking to enhance account security.

Key Takeaways:

- TOTP significantly improves security over password-only authentication
- Proper implementation requires attention to secret management, time synchronization, and rate limiting
- User experience is crucial for successful adoption
- TOTP complements emerging authentication technologies like WebAuthn
- Regular testing and monitoring ensure continued reliability

As authentication technology evolves toward passwordless solutions and biometric methods, TOTP will continue to serve as a valuable fallback mechanism and educational stepping stone. Organizations implementing TOTP today are building a foundation for more advanced authentication strategies tomorrow.

By following the guidance in this booklet, developers and security professionals can implement TOTP effectively, balancing security requirements with user experience to create authentication systems that protect users without creating unnecessary friction.

Appendix A: Quick Reference Guide

Essential TOTP information at a glance:

Item	Value
RFC Standard	RFC 6238 (TOTP), RFC 4226 (HOTP)
Default Time Step	30 seconds
Default Code Length	6 digits
Default Algorithm	HMAC-SHA1
Secret Key Size	160 bits (20 bytes), Base32 encoded
Time Window Tolerance	±1 step (±30 seconds)
URI Format	otpauth://totp/{label}? secret={secret}&issuer={issuer}

Appendix B: Troubleshooting Guide

Common problems and their solutions:

Problem: Codes Always Invalid

- Check server time synchronization (NTP)
- Verify device time is set to automatic
- Ensure correct secret was provided during setup
- Confirm time step parameter matches (default 30 seconds)

Problem: Intermittent Failures

- Increase time window tolerance to ±2 steps
- Check for clock drift between server and client
- Verify rate limiting is not rejecting valid attempts

Problem: Setup Failures

- Ensure QR code is correctly formatted
- Verify Base32 encoding of secret is valid
- Check for special characters in issuer or label
- Test with multiple authenticator apps

Appendix C: Resources and Further Reading

Additional resources for learning about TOTP and authentication:

Official Standards

- RFC 6238: TOTP: Time-Based One-Time Password Algorithm
- RFC 4226: HOTP: An HMAC-Based One-Time Password Algorithm
- RFC 4648: The Base16, Base32, and Base64 Data Encodings

Key URI Format

- Google Authenticator Key URI Format specification
- https://github.com/google/google-authenticator/wiki/Key-Uri-Format

Libraries and Tools

- PyOTP (Python): https://github.com/pyauth/pyotp
- OTPAuth (JavaScript): https://github.com/hectorm/otpauth
- ROTP (Ruby): https://github.com/mdp/rotp
- OTPSharp (C#): https://github.com/kspearrin/Otp.NET

Security Guidelines

- NIST Special Publication 800-63B: Digital Identity Guidelines
- OWASP Authentication Cheat Sheet
- CIS Controls for Two-Factor Authentication

About the Author

Santokh Singh Saggu is a founder of Riasys Technology and a technologist who works at the intersection of thinking and making. He builds software, applications, and electronic systems by blending the technology of the mind—design, psychology, and philosophy—with the technology of matter—computers, programming languages, and machines.

He enjoys reading and studying books across multidisciplinary fields and learning from people and nature. Through this, he seeks knowledge, wisdom, insights, and patterns, which he analyzes and synthesizes into writing, design, and practical systems.

He believes most challenges are not purely technical but stem from confusion and lack of clarity. By focusing on simplicity and structure, he aims to make work and learning more manageable.

His work supports entrepreneurs and professionals in solving real business problems, and helps learners move from uncertainty to understanding in areas such as programming and electronic device design.

Visit : https://www.riasys.co.in