On Conducting Systematic Security and Privacy Analyses of TOTP 2FA Apps

Conor Gilsenan
U.C. Berkeley

Noura Alomar
U.C. Berkeley

Serge Egelman
U.C. Berkeley / ICSI

Abstract

Two-factor authentication (2FA) has consistently proven to dramatically increase the security of online accounts, but the privacy implications of enabling different methods of 2FA are not well studied. The Time-based One-Time Passwords (TOTP) algorithm is one method of 2FA that is widely deployed throughout industry. RFC6238 defines how the client and server utilize a shared secret to generate and validate a deterministic one-time password (OTP) during authentication. However, there is no standard for how to back up the shared secret on the client, resulting in custom implementations across dozens of consumer TOTP apps that directly impact the security and privacy of the TOTP 2FA scheme. In this paper, we define an assessment methodology for conducting systematic security and privacy analyses of the backup and recovery functionality of TOTP apps. Using this workflow, we analyzed the Authy Android app and observed that it sends the plaintext usernames of third party accounts to Authy servers, uses an inadequate work factor when deriving keys with PBKDF2, and encrypts Base32 encoded TOTP secrets, which makes them vulnerable to offline attacks.

1 Introduction

The Time-based One-Time Passwords (TOTP) algorithm is one method of two-factor authentication (2FA) that is widely deployed throughout industry, including at some of the largest sites on the internet, such as facebook.com, google.com, and amazon.com. RFC6238 [8] defines the technical algorithm which allows the client and server to utilize a shared secret to generate and validate a deterministic one-time password (OTP) during authentication. However, it avoids specifying many of the practical steps required to implement TOTP 2FA as a security mechanism in the real world, such as transferring the shared secret from server to client.

A separate publication by Google [4] has become the de facto standard for transferring data from the server to the client during TOTP registration and defines several key terms, including: (1) the *issuer*, which is the name of the website on which the user is currently enabling TOTP 2FA; (2) the *name*, which is the username of the user's account on the website; and (3) the *TOTP secret*, which is the shared secret defined by RFC6238 [8]. The client requires these three fields to generate OTPs and differentiate between accounts in the user interface.

Many TOTP apps provide custom backup and recovery mechanisms to ensure that users do not lose access to this data when they lose their devices, buy new devices, or uninstall the app. These features are critical usability enhancements, but can also introduce security and privacy issues depending on how they are implemented. Studies evaluating the security and privacy of TOTP apps, and their back-up mechanisms specifically, are sparse within the literature. However, researchers have conducted such studies on password managers, which share the same security goal: securely backup a local secret.

Bhargavan and Delignat-Lavaud [10] analyzed several applications that leveraged client-side cryptography to protect data stored remotely and concluded that vulnerabilities in the client applications themselves can expose sensitive data. Li et. al [13] defined high level security goals and attack vectors to analyze five prevalent web-based password managers, but the paper lacks technical details required to replicate their work. Belenko and Sklyarov [9] recovered master passwords in 16 password management apps via brute force attacks. Overwhelmingly, researchers have concluded that most users are not able to choose passwords that resist offline attacks [11, 12, 15].

In this paper, we define a tool-agnostic assessment methodology for conducting systematic security and privacy analyses

Who Are You?! Adventures in Authentication (WAY) 2020.
August 7, 2020, Virtual Conference.

of the backup and recovery functionality of TOTP apps. We present a case study in which we employ our methodology to analyze the popular Authy Android app and propose solutions for three identified security and privacy issues. Our goal is for other researchers to utilize this assessment methodology to both replicate our findings and to analyze other prevalent TOTP apps.

2 Assessment Methodology

This section defines the assessment methodology and explains how it can be used to answer the following key questions:

1. What personal information, if any, is leaked to the company that develops the TOTP app?
2. What is the risk of an external attacker accessing the TOTP backups?
3. What is the risk of an attacker compromising the plaintext TOTP secrets from the TOTP backups?

We assume that the attacker does not have physical access to the user's device and that all communication between the TOTP app and remote servers is via HTTPS. Discussions of encryption throughout this paper refer to the TOTP app locally encrypting fields within the TOTP backup before sending it remotely.

Phase 1: Analyze Documentation. In addition to prior academic work, the researcher should gather and preserve copies of the TOTP app's privacy policy, company white papers, blog posts, conference talks, and previous analyses conducted in industry. This information can help to quickly establish an understanding of the architecture and cryptographic design of the backup and recovery functionality of the app, and any public disclosures of personal information collected by the app. However, the question remains whether said information accurately reflects how the TOTP app actually functions. Therefore, the next phase of the workflow shifts focus to reverse engineering the TOTP app itself.

Phase 2: Capture Network Traffic. To begin verifying the accuracy of the gathered documentation, the researcher should to review the data that the app sends to remote servers. The goals of this phase include:

1. Identify any plaintext or encrypted personal information in TOTP backups sent remotely; and
2. Obtain the ciphertext of backed up TOTP secrets.

The researcher should capture all network traffic while executing the following actions.

First, register a TOTP secret before enabling the TOTP app's backup functionality. It is likely a clear privacy issue if any TOTP secrets or other personal information is disclosed while backups are disabled.

Second, enable the backup functionality and analyze the request/response traffic. Specifically, determine whether the *issuer*, *name*, and *TOTP secret* fields are encrypted before being sent remotely. Plaintext content in the TOTP backup can be read by the backup servers, which is a clear security and/or privacy issue, particularly for those three fields.

Third, register a second TOTP secret and observe whether it is backed up automatically. Real-time backups are likely ubiquitous because they provide a desirable user experience, but they can also pose a privacy risk depending on which data in the TOTP backup is encrypted. For example, if the *issuer* field were included in plaintext, then the backup servers would learn the specific time that the user enabled 2FA on their account on the third party service.

Lastly, uninstall and reinstall the app to simulate losing the TOTP secret. Ideally, the app would be installed on an entirely separate device in case it utilizes fingerprinting to uniquely identify devices. Once reinstalled, exercise the entire recovery workflow to recover the plaintext TOTP secrets.

Phase 3: Analyze App Binary. Several of the fields in the TOTP backup likely contain encrypted content, which cannot be easily analyzed without decompiling and analyzing the app's binary. The goals of this phase include:

1. Identify the cryptographic algorithms and configurations used to generate TOTP backups; and
2. Determine how the app verifies that the user has entered the correct password during recovery.

If the app derives encryption keys from user provided password(s), then it is critical to document the key derivation function (e.g. PBKDF2) and its configuration, such as the work factor (e.g. rounds). Additionally, document the encryption cipher and mode used to encrypt the TOTP secret and other data in the TOTP backup. The researcher must prove that the ciphertext obtained from analyzing the network traffic was, in fact, generated using the cryptographic processes documented while analyzing the app's binary. One approach to accomplish this is to implement the app's decryption process in a separate script and show that decrypting the ciphertext yields the original plaintext TOTP secret.

Finally, the researcher must determine how the TOTP app verifies that the user entered the correct password during the recovery process. The cryptographic design of this functionality is critical to an attacker's ability to "crack" the ciphertext in an offline attack, which is discussed in the next section.

Phase 4: Attack Ciphertext Offline. The goal of this phase is to determine whether an attacker can extract plaintext TOTP secrets from TOTP backups.

If the TOTP app derives keys from a user provided password, then it is likely that the ciphertext can be "cracked" by leveraging techniques used to crack password hashes. An attacker conducting an offline attack against the ciphertext would iterate through password guesses and, for each, derive the encryption key and decrypt the ciphertext. The attacker would determine whether each guess is correct by leveraging the same verification mechanism that the TOTP app uses during the recovery process to notify the user whether they provided the correct backup password. For example, the app might use an authenticated encryption algorithm, such as AES-GCM; it might generate a MAC over the plaintext or ciphertext, which can be validated; or, it might use some other custom heuristic. Provided there is a reasonable method of distinguishing between success and failure for each password guess, modern open-source password cracking tools, such as Hashcat [2], can be adapted to crack the ciphertext.

If the TOTP app uses randomly generated keys, then an offline attack is likely computationally infeasible. In general, it is a meaningful barrier to force the attacker from a fully offline attack to an online attack because they must then face the third party service's online defenses, such as API rate-limiting.

Phase 5: Analyze Recovery Workflow. Unlike an internal attacker who has direct access to TOTP backups, an external attacker needs to obtain them via remote attack vectors. The researcher should exercise and document the entire recovery workflow with the goal of revealing any opportunities for:

1. an external attacker to access TOTP backups; or

2. the user to stop any malicious recovery attempts initiated by an attacker.

Authentication mechanisms on the backup servers which attempt to identify who is initiating the recovery request are a compelling target. The researcher should record all communications received during recovery, such as SMS messages, emails, and in-app notifications. It is also often extremely useful to document the recovery process as a workflow diagram so that it can be visually analyzed for patterns or weak points.

Phase 6: Recommend Improvements. To conclude the analysis, the researcher should summarize all security and privacy issues, and clearly outline potential solutions to each. There may be situations where drastic architectural changes are unavoidable, but researchers should strive to recommend realistic and pragmatic solutions because they are more likely to actually be implemented, which benefits the real world users of the app.

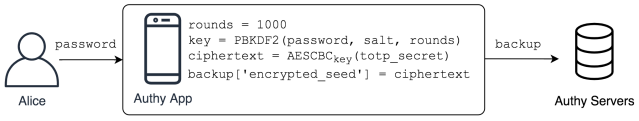

Figure 1: Overview of the Authy backup architecture.

3 Case Study: Authy 2FA

In this section, we present a case-study in which we employed our assessment methodology to conduct a security and privacy analysis of version 24.3.1 of the Authy 2FA Android App, which was published to the Google Play Store on April 1, 2020.

Phase 1: Analyze Documentation. In 2018, Authy published a blog post [3] detailing the backup and recovery architecture of the Authy 2FA app, which included enough technical detail to diagram the entire process (see Figure 1). The post states that the Authy app prompts the user to enter a password with a minimum length of 6 characters. Then, it stretches the password using a random salt and 1,000 rounds of Password-based Key Derivation Function 2 (PBKDF2) [6] to derive an encryption key. The resulting key is used to encrypt the TOTP secret using AES-CBC and the resulting ciphertext is included in the TOTP backup, which is sent to Authy servers.

Phase 2: Capture Network Traffic. We used a recent version of a closed source tool developed by Reardon et. al. [14] to capture the plaintext network traffic on the Android device before TLS encryption was applied. In addition to identifying the TOTP secret ciphertext and associated random salt value, we observed that the TOTP backup contained the *name* and *issuer* values in plaintext. This means that by simply enabling the backup functionality, users of the Authy app are unwittingly providing the names of the third party services they use and their account usernames on those services to Authy servers. For example, for the fictitious user Alice, Authy could learn that "Alice's username for her account on example.com is alice123."

We cannot determine any technical reason that the TOTP backups include the name and issuer fields in plaintext. Backups could be bound to the user's Authy account, which the user is forced to create upon installation by providing an email address and phone number. Additionally, there is no in-app disclosure to warn the user that Authy collects this personal information. The Authy Privacy Policy [1] does include a statement which may relate to this data collection: "We keep a record of your log-ins to accounts for which you use Authy for 2-factor authentication." However, this statement is vague and further research is required to determine whether users

understand that this data is collected.

Phase 3: Analyze App Binary. We downloaded the Android Package (APK) file for the Authy app from APK Mirror[1] and decompiled it using the open source tools dex2jar[2] and CFR.[3] After reviewing the non-obfuscated code of the decompiled binary, we were confident that all of the cryptographic details outlined in Authy's blog post [3], discussed in Phase 1, accurately reflected the functionality of the app.

However, the statically defined PBKDF2 work factor of 10^3 rounds is too low to meaningfully slow offline attacks given the capabilities of modern hardware. OWSAP recommends a minimum of 10^4 rounds and suggests up to 10^5 in higher security environments [5].

Also, the Authy app does not utilize a MAC, which is the common approach to provide message integrity since AES-CBC does not provide authentication natively. Instead, to determine if the user provided the correct password during recovery, the app uses a custom heuristic that relies on the de facto standard [4] Base32 encoding of the TOTP secret in the QR code during TOTP 2FA setup. The app derives the encryption key from the password, decrypts the ciphertext, and considers the process successful if the resulting plaintext is valid Base32 format. Unfortunately, encrypting the TOTP secret in Base32 format makes the resulting ciphertext vulnerable to offline attacks, which we discuss in the next section.

Phase 4: Attack Ciphertext Offline. To prove that we correctly identified the cryptography used to generate TOTP backups, we wrote a script in Go which accepted the following inputs: (1) the ciphertext of the encrypted TOTP secret; (2) the associated salt; and (3) the password that we chose when enabling backups. The script implements the same decryption logic as the Authy app and decrypts the input ciphertext to verify that the resulting plaintext matches the expected TOTP secret.

Unlike our analysis in the lab, an external attacker will not know the user's backup password. However, the fact that Authy encrypts TOTP secrets in Base32 format makes the ciphertexts vulnerable to offline "cracking" attacks. The probability that a single random password guess will result in plaintext that is valid Base32 is approximately 10^{-29} (see Appendix A for calculations), which means that any guess which *does* result in valid Base32 is very likely to be the user's real recovery password.

Considering that the Authy app allows backup passwords as short as 6 characters, the effective guessing techniques employed by modern password cracking tools alone are almost certainly enough to crack the ciphertext of TOTP secrets for normal users who rely on weak passwords. However, the fact that TOTP backups include the victim's account usernames in plaintext increases the efficacy of the attack. Prior work by Wang et. al. [16] used personal information about the victim to achieve "success rates over 73% against normal users and over 32% against security-savvy users" within only 100 attempts during online guessing attacks. Here, the attacker could use the victim's account usernames to harvest personal information from publicly available sources and make orders of magnitude more guesses in their offline attack.

Phase 5: Analyze Recovery Workflow. We analyzed Authy's recovery workflow for users who have lost their device, but still have access to their phone number. Other recovery scenarios will be reviewed in future work.

The attacker can enter the user's phone number on Authy's website to submit a recovery request. The user is immediately notified via SMS and email, and the recovery process only begins if they click a link in the email; otherwise, the attack fails. Even if the user accidentally confirms the malicious recovery request, they have ample opportunity to cancel the process within the 24 hour delay period [7] by clicking a cancel link in one of several confirmation and progress notifications sent via email. We believe that it would be infeasible for an attacker to obtain the ciphertext via this workflow unless they also compromised the victim's email account.

Phase 6: Recommend Improvements. Our analysis found three security and privacy issues that should be addressed. First, the TOTP backups include the *name* and *issuer* fields in plaintext, which informs Authy which third party services the user utilizes and their account usernames on those services. The simple remedy is to encrypt those fields before including them in the TOTP backup.

Second, encryption keys are derived from user provided passwords using PBKDF2 with only 10^3 rounds as a work factor, which is lower than current best practices. At a minimum, Authy should increase the statically defined work factor to at least 10^4, the minimum recommended by OWASP [5]. Switching to a memory-hardened and/or cpu-hardened key derivation function, such as scrypt, bcrypt, or Argon2, would allow the app to dynamically calculate the work factor based on the actual resources available, thus allowing higher end devices to employ better security.

Finally, the encrypted TOTP secrets are vulnerable to offline attack because they are encrypted in Base32 format. Offline attacks would be infeasible if the TOTP secrets were Base32 decoded before encrypting because plaintext TOTP secrets are random values and there would be no heuristic to differentiate one password guess from another. However, this would impact the usability of the app because it would not be possible to display a message during recovery if the user provides the wrong password. Future work is required to recommend a solution that balances security and usability.

[1] https://www.apkmirror.com
[2] https://github.com/DexPatcher/dex2jar
[3] https://github.com/leibnitz27/cfr

Responsible Disclosure to Authy. We shared a draft of this paper with Authy on July 6, 2020. Authy was receptive, had several internal teams review our paper, and started a dialogue to discuss our findings.

4 Conclusion and Future Work

In this paper, we have shown that the implementation details of the backup and recovery mechanisms of TOTP apps can introduce both security and privacy issues for end users. We plan to utilize the proposed assessment methodology to analyze other prevalent TOTP apps to determine whether these issues are pervasive. Our eventual goal is to propose a set of design requirements that all TOTP apps should follow to avoid introducing security and privacy issues via their backup and recovery mechanisms.

Acknowledgments

This research received funding from the Center for Long-Term Cybersecurity (CLTC) at U.C. Berkeley.

References

[1] Authy app privacy notice. Available: https://web.archive.org/web/20200616001529/https://www.twilio.com/legal/privacy/authy. [Online; accessed: 16-June-2020].

[2] Hashcat plugin development guide. Available: https://github.com/hashcat/hashcat/blob/master/docs/hashcat-plugin-development-guide.md. [Online; accessed: 22-June-2020].

[3] How Authy 2FA backups work. Available: https://web.archive.org/web/20200616101909/https://authy.com/blog/how-the-authy-two-factor-backups-work/. [Online; accessed: 16-June-2020].

[4] Key uri format. Available: https://github.com/google/google-authenticator/wiki/Key-Uri-Format. [Online; accessed: 12-May-2020].

[5] Password storage cheat sheet. Available: https://web.archive.org/web/20200617224006/https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2. [Online; accessed: 17-June-2020].

[6] PKCS #5: Password-based cryptography specification version 2.0. Available: https://www.ietf.org/rfc/rfc2898.txt. [Online; accessed: 19-June-2020].

[7] Restoring authy access on a new, lost, or inaccessible phone. Available: https://web.archive.org/web/20200607061849/https://support.authy.com/hc/en-us/articles/115012672088-Restoring-Authy-Access-on-a-New-Lost-or-Inaccessible-Phone. [Online; accessed: 07-June-2020].

[8] TOTP: Time-based one-time password algorithm. Available: https://tools.ietf.org/html/rfc6238. [Online; accessed: 02-Oct-2019].

[9] Andrey Belenko and Dmitry Sklyarov. "secure password managers" and "military-grade encryption" on smartphones: Oh, really? *Blackhat Europe*, page 56, 2012.

[10] Karthikeyan Bhargavan and Antoine Delignat-Lavaud. Web-based attacks on host-proof encrypted storage. In *WOOT*, pages 97–104, 2012.

[11] Joseph Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[12] Joseph Bonneau, Sören Preibusch, and Ross Anderson. A birthday present every eleven wallets? the security of customer-chosen banking pins. In *International Conference on Financial Cryptography and Data Security*, pages 25–40. Springer, 2012.

[13] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium*, pages 465–479, 2014.

[14] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *Proceedings of the 28th USENIX Security Symposium*, pages 603–620, 2019.

[15] Blase Ur, Sean M Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L Mazurek, William Melicher, and Richard Shay. Measuring real-world accuracies and biases in modeling password guessability. In *24th USENIX Security Symposium*, pages 463–481, 2015.

[16] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1242–1254, 2016.

A Evaluating the Efficacy of Authy's Base32 Heuristic

The following is the mathematical evaluation of the Base32 heuristic used during the recovery process to determine whether the user provided the correct password.

We must answer the question: *Given the ciphertext of the encrypted TOTP secret, what is the probability that a single password guess will generate a plaintext output that is valid Base32?* Recalling that an ASCII character has $2^8 = 256$ possible bit permutations and that Base32 allows 32 valid characters (A-Z and 2-7), the probability that the ASCII representation of an L-length string is valid Base32 is:

$$= P(\textit{single byte is valid Base}32)^L = (32/256)^L = 0.125^L$$

The probability that a single password guess for a 32 byte TOTP secret (L = 32), which is a common length used in industry, will generate valid Base32 is:

$$= 0.125^{32} \approx 1.26 * 10^{-29}$$

With a very high probability, this heuristic will accurately verify whether the user entered the correct recovery password because it is extremely unlikely that the decryption process will result in plaintext that is valid Base32 format if the encryption key is derived from an incorrect password. TOTP secrets are random values, which means that there would be no heuristic to differentiate one password guess from another if TOTP secrets were Base32 decoded before encryption.