Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Audit-Report Passbolt Oracle Attack Scenario 12.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi, Dr. D. Bleichenbacher

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Passbolt is an open source credential platform for modern teams. A versatile, battle-tested solution to manage and collaborate on passwords, accesses, and secrets. All in one.”

From https://www.passbolt.com/

This PBL-12 report presents the findings accrued during crypto reviews and feasibility checks covering the plausibility of Passbolt oracle attacks.

The audit was conducted by three senior Cure53 pentesters at the behest of Passbolt SA, who specified the assessment’s aims during discussions held in November 2024. The test initiatives were actioned in CW50 December 2024 with a six day allocation.

A single work package (WP) was created for this exercise, entitled *WP1: Crypto reviews & feasibility checks covering possible oracle attacks*. The Passbolt maintainers granted access to sources, papers, and other assorted data, in accordance with the preferred white-box pentesting methodology. Preparatory measures were carried out prior to the active review phase in CW49 December 2024, helping to establish an optimized and hindrance-free evaluation environment.

The two sets of personnel from both organizations were invited to join a dedicated and shared Slack channel, which hosted real-time discussions about the ongoing project status and progress. The collaborations were seamless and minimal questions were needed, since the wider objectives and scope were clear from the start. Live reporting was deemed unnecessary for this testing iteration, though Cure53 ensured to relay frequent updates regarding the examinations and pertinent findings when required.

Despite achieving satisfactory coverage over the targeted features, the Cure53 consultants were only able to identify one security-relevant finding, pertaining to a merely *Informational* recommendation with minimal risk. The avoidance of severe security pitfalls reflects positively on the scrutinized implementation. Moreover, the main issue discovered entails a timing attack affecting an underlying library, which should be relatively easy to fix. Nevertheless, the use of RSA PKCS#1 for encryption within OpenPGP is discouraged due to the plausible risk of padding attacks via timing leaks, for instance.

Transitioning away from RSA PKCS#1 is hindered by the slow adoption of new ciphers within OpenPGP. To address this, the report asserts that the cryptographic library should remain responsible for mitigating timing leaks, rather than the application. Furthermore, the OpenPGP library developers should be contacted to address the padding flaw and improve security.

Lastly, Cure53 recommends researching protocols that may allow ciphertext modifications leading to significant timing differences. By prioritizing security within the cryptographic library and fostering collaboration with developers, the security of OpenPGP can be significantly enhanced.

The report will now provide insights into the *Scope* and testing setup, as well as display a comprehensive breakdown of all available materials in bullet point form. Next, the *Methodology* section clarifies the evaluation techniques applied by Cure53 and all interesting subsequent observations. This section hopefully verifies the extent of the test team's endeavors, despite the low yield of findings.

Subsequently, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* (of which none were located here) and followed by the *Miscellaneous Issues*. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will appraise the general security posture of the elements in focus, offering high-level hardening advice and next steps for the internal team.

Scope

- **Cryptography reviews & feasibility check covering possible Passbolt oracle attacks**
  - **WP1:** Crypto reviews & feasibility checks covering possible oracle attacks
    - **Primary focus:**
      - This review analyzed the feasibility of RSA PKCS #1 oracle attacks against the Passbolt implementation, as claimed in Duan, Wang, and Fu's *Security Analysis of Master-Password-Protected Password Management Protocols* paper.
      - The review covered code developed by Passbolt and corresponding protocols.
      - To review the feasibility of oracle attacks, Cure53 evaluated the relevant code of the underlying gopenpgp implementation by ProtonMail.
      - The definition of expectations was considered a small but crucial aspect of the review. Specifically, Cure53 believes that the OpenPGP implementation is secure against chosen ciphertext and timing attacks.
    - **Passbolt implementation:**
      - https://github.com/passbolt/mobile-passbolt-android/blob/8244f0a01a6d67148fe2a004e3abcb78d245d448/
    - **Underlying OpenPGP implementation source code:**
      - https://github.com/ProtonMail/gopenpgp/pull/318/commits/1ebd390de488f0f412ad024a1681d1c5fee78f21
    - **Paper outlining alleged attack:**
      - https://www.researchgate.net/publication/385420109_Security_Analysis_of_Master-Password-Protected_Password_Management_Protocols

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Methodology

This section documents the testing methodology applied by Cure53 during this project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, especially in the absence of significant security vulnerabilities detected.

RSA PKCS #1 v1.5 oracles

RSA PKCS #1 v1.5 is affected by a weakness that allows decrypting RSA-encrypted messages via a chosen ciphertext attack[1]. This breach strategy requires the attacker to send modified ciphertexts to the RSA private key holder, as well as glean specific information regarding the decrypted message.

Duan, Wang, and Fu's paper describes their analysis of password management protocols[2], claiming that OpenPGP-based authentication employed by Passbolt contains an RSA PKCS #1 oracle, which can be leveraged by a malicious attacker to recover encrypted key material.

The feasibility and severity of the attack depends on both the viability of the adopted ciphertext attack and the type of information gained by the attacker. As such, this audit's core objective was to determine these factors.

Types of PKCS #1 oracles

An attacker can retrieve insights via a number of practicable methods, as outlined below.

Information leakage via error message return
This approach depends on the correctness of the encrypted PKCS #1 padding. If the error messages contain information concerning the type or position of the incorrect padding, the attack can be significantly simplified.

Information leakage via timing side channel
To exemplify this approach, an analysis of NimbusJose was able to distinguish between correct and incorrect paddings via decryption time measurement[3]. Here, the timing difference was evoked by throwing and catching an exception in the event of incorrect padding. In Java, exception throwing is computationally expensive and can lead to observable timing discrepancies.

[1] https://archiv.infsec.ethz.ch/education/fs08/secsem/bleichenbacher98.pdf
[2] https://www.researchgate.net/publication/385420109_Security_Analysis_[...]_Protocols
[3] https://github.com/C2SP/wycheproof/blob/master/java/com/google/security/[...]/[...]/Nimbus[...]Test.java

In contemporary times, timing side channels are still oftentimes culpable for security vulnerabilities. For instance, CVE-2024-2467, CVE-2024-3296, CVE-2024-0914, CVE-2024-23218, CVE-2024-21484, and CVE-2024-0553 pertain to timing leaks across various implementations documented in 2024.

Number of chosen messages

This heavily depends on the nature of the padding oracle. Two significant factors here are the probability of a message holding a valid padding RSP and the frequency of distinguishable behaviors. An implementation that integrates additional checks (e.g., comparing version numbers or checking the length of the encrypted message) and subsequently throws an exception is typically significantly more difficult to compromise than a counterpart that offers verbose information regarding the padding error.

Böck, Somorovsky, and Young estimate that between 10,000 and 18,000,000 chosen messages are required to break different TLS implementations, depending on the information leaked by the respective oracle and partial knowledge of the RSA-encrypted message[4] [5].

The ability to glean partial information regarding the message to decrypt ultimately reduces the number of required chosen messages, since the insights can help to generate ciphertexts that are increasingly likely to conform with PKCS #1. Partial knowledge about the padding of the encrypted message (including the fact that the message is encrypted using PKCS #1). Cure53 assumes that the ramifications of partial knowledge have not been exhaustively researched; thus, any published attack can potentially be enhanced by exploiting this information.

Cure53 conducted a concise examination here, which assumed that a timing analysis can distinguish chosen messages with both correct PKCS #1 padding and correct size of the session key from other ciphertexts. With this oracle, one should be able to decrypt an RSA PKCS #1 decrypted message with less than 20,000,000 chosen messages. The test team could not ascertain any method by which to significantly simplify the attack via leveraging known information regarding the message targeted for decryption. Nevertheless, the aforementioned value should simply be considered an upper bound.

[4] https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-bock.pdf
[5] https://link.springer.com/chapter/10.1007/978-3-642-32009-5_36

OpenPGP

Asymmetric encryption in OpenPGP employs a hybrid approach whereby a symmetric key is encrypted with a public key algorithm, and the message is encrypted with the symmetric key. The symmetric part of the encryption contains an integrity check serving to detect modifications.[6]

If RSA PKCS #1 v1.5 is utilized for the asymmetric part of the encryption, the decryption must be implemented in a careful manner to sufficiently block oracles. In particular, it is important to ensure that decryption does not permit distinguishing between various error types, as outlined in RFC 4880 section 14[7]:

> *…The simplest solution is to report a single error code for all variants of decryption errors so as not to leak information to an attacker. …*

However, the recommendations offered in this section are considered insufficient, since the RFC does not include timing differences. A significant timing difference can occur if the symmetric encryption is either bypassed or performed, depending on the validity of the PKCS #1 padding. These differences are observable even if the decryption of the modified ciphertexts fails. This situation may be exacerbated as the attacker can attempt to increase the timing difference by increasing the length of the ciphertext's symmetric part. Consequently, successful decryptions of the symmetric ciphertext and iterations that abort early are easier to distinguish.

Maury, Reinhard, Levillain, and Gilbert outline the implications of oracles affecting the OpenPGP message format in their 2015 paper, which focuses on the symmetric part of the encryption[8].

Code Review

Passbolt decrypts OpenPGP-encrypted messages using ProtonMail's gopenpgp library. Precise implementation of RSA decryption in gopenpgp is critical. As such, Cure53's code review placed particular emphasis on vetting the OpenPGP implementation. The library's developers are aware of padding oracles and describe potential attack vectors in their ElGamal implementation. However, sufficient timing attack countermeasures are not documented, presumably due to the assumption that the use of OpenPGP for emails is not subject to timing measurement. Cure53 must reiterate that timing differences in Passbolt's protocol may lead to information leakage.

[6] https://openpgp.dev/book/zoom/encryption.html
[7] https://datatracker.ietf.org/doc/html/rfc4880#section-14
[8] https://cyber.gouv.fr/sites/default/files/2015/05/format-Oracles-on-OpenPGP.pdf

Crucially, the code that handles RSA keys leverages the go/rsa library. Cure53 noted that the library authors have provided certain functionalities for RSA PKCS #1 encrypted messages that can be adopted for padding oracle attack prevention, such as *rsa#DecryptPKCS1v15*[9] and *rsa#DecryptPKCS1v15SessionKey*[10].

The second function decrypts an RSA PKCS #1 encrypted ciphertext, expecting a SessionKey of a given size. If either the PKCS #1 padding or length of the encrypted SessionKey is incorrect, a random byte string of the correct length is returned. The motivation behind this function is that the caller of the function will always receive a session key (either the encrypted version or a random key) if the ciphertext is incorrect. Using a random session key will result in a failed decryption of the ciphertext's symmetric part. The advantage of this method is that the time required for ciphertext decryption does not depend on whether the PKCS #1 padding is valid, while the time associated with message decryption does not leak information regarding the padding validity.

If *DecryptPKCS1v15* is utilized for decryption and an error is immediately returned upon failure, decryption of the ciphertext's symmetric part will only be performed if the padding is correct and contains a usable session key, which leads to an observable timing difference. Notably, an attacker can increase the timing difference between correct and incorrect paddings by increasing the size of the ciphertext. Conversely, if the OpenPGP decryption uses *DecryptPKCS1v15SessionKey* with a randomly generated symmetric key as input, the decryption of the ciphertext's symmetric part will always be attempted, regardless of whether the padding of the decrypted RSA ciphertext is correct. In this case, a modified ciphertext should always fail since OpenPGP messages leverage an integrity check. If the decryption is indeed constant time (as claimed in the documentation), the attacker should not be able to distinguish between valid and invalid RSA paddings.

Affected file:
https://go.dev/src/crypto/rsa/rsa.go

Affected code:
```
//
// Decrypt decrypts ciphertext with priv. If opts is nil or of type
// *[PKCS1v15DecryptOptions] then PKCS #1 v1.5 decryption is performed.
Otherwise
// opts must have type *[OAEPOptions] and OAEP decryption is done.

func (priv *PrivateKey) Decrypt(rand io.Reader, ciphertext []byte, opts
crypto.DecrypterOpts) (plaintext []byte, err error) {
	if opts == nil {
		return DecryptPKCS1v15(rand, priv, ciphertext)
	}
```

[9] https://pkg.go.dev/crypto/rsa#DecryptPKCS1v15
[10] https://pkg.go.dev/crypto/rsa#DecryptPKCS1v15SessionKey

Countermeasures

Padding oracles can be avoided in the long term by deprecating the use of RSA PKCS #1 as the underlying asymmetric encryption algorithm. In fact, RFC 9580 no longer recommends RSA keys, integrating X25519 and X448 as new options for asymmetric encryption.[11] Even though support for these encryption modes remains unclear at present, X25519 offers multiple advantages such as robust security and smaller ciphertexts, while X25519 offers faster key generation over RSA.

The Passbolt team can also consider alternative encryption modes. RSA-OAEP would be beneficial in this context since the decryption of the ciphertext's asymmetric part already prevents chosen message attacks against RSA-encrypted messages, whereas RSA PKCS #1 v1.5 requires a careful implementation of symmetric and asymmetric decryption. Another viable approach would be to adopt RSA-KEM (RFC 5990).

[11] https://www.rfc-editor.org/rfc/rfc9580.html#name-algorithm-specific-fields-for-

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

PBL-12-001 WP1: Lack of RegExp for authentication token validation *(Info)*

Cure53 noted that the code responsible for validating user authentication tokens relies on a series of conditional checks (i.e., string splitting and version/UUID checks), which can introduce maintenance overheads if data formats evolve over time. As such, the implementation can be enhanced by utilizing a single, well-tested regular expression that enforces the same structural requirements. By replacing manual checks with an optimally constructed RegExp, one can reduce the risk of overlooking edge cases and simplify future maintenance.

Affected file:
gpgAuthToken.js

Affected code:
```
  /**
   * Validate authentication token fields individually.
   *
   * @param {string} field The name of the field to validate
   * @param {string} value The value of the field to validate
   * @return {*} True or Error
   */
  validate(field, value) {
    let sections = [];
    switch (field) {
      case 'token' :
        if (typeof value === 'undefined' || value === '') {
          return new Error('The user authentication token cannot be
              empty');
        }
        sections = value.split('|');
        if (sections.length !== 4) {
          return new Error('The user authentication token is not in the
              right format');
        }
        if (sections[0] !== sections[3] && sections[0] !== 'gpgauthv1.3.0')
{
          return new Error('Passbolt does not support this GPGAuth
              version');
        }
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
        if (sections[1] !== '36') {
          return new Error(`Passbolt does not support GPGAuth token nonce
              longer than 36 characters: ${sections[2]}`);
        }
        if (!Validator.isUUID(sections[2])) {
          return new Error('Passbolt does not support GPGAuth token nonce
              that are not UUIDs');
        }
        return true;
      default :
        return new Error(`No validation defined for field: ${field}`);
    }
  }
```

The code above splits the token into four sections and relies on several explicit checks for each section. By scattering checks across multiple *if* statements, an edge case may be easily missed or future regressions introduced when updating the format or adding new rules.

To mitigate this issue, Cure53 suggests adopting a single, comprehensive regular expression as a replacement for the currently implemented manual checks. The RegExp should assert that the token is non-empty and composed of four pipe-delimited sections, the conditional token version rules, a nonce length check, and the valid UUID format.

The example below pertains to a single RegExp that encodes all of the aforementioned rules. Pertinently, this may require modifying to align with Passbolt's distinct requirements:

```
^gpgauthv1\.3\.0\|36\|[0-9A-Fa-f]{8}-[0-9A-Fa-f]{4}-[0-9A-Fa-f]{4}-[0-9A-
Fa-f]{4}-[0-9A-Fa-f]{12}\|gpgauthv1\.3\.0$
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

The overarching impressions of this winter 2024 Passbolt-Cure53 joint venture will now be discussed. In short, the confirmation can be made that the components under scrutiny have garnered a favorable verdict, considering the scant number of observed security drawbacks.

Cure53 conducted crypto reviews and feasibility checks covering possible Passbolt oracle attacks. One key observation here is that oracle attacks should be handled by the underlying OpenPGP library. The current OpenPGP implementation is affected by certain weaknesses that raise the probability of exploitation, for which improvement recommendations are proposed.

Passbolt's authentication protocols leverage a hybrid encryption scheme to send challenges from the server to the client. A successful decryption ensures that the receiver is in possession of the corresponding RSA private key.

The hybrid encryption used by Passbolt is based on OpenPGP, which adopts RSA PKCS #1 v.1.5 encryption and an authenticated symmetric cipher. Duan, Wang, and Fu's 2024 paper stipulates that the lack of chosen ciphertext security for RSA PKCS #1 v.1.5 allows chosen ciphertext attacks against the encryption mode. This compromise would facilitate decrypting authentication challenges and keys encrypted with the same RSA key. The attack will only be successful if the challenge receiver leaks information regarding the padding check during challenge decryption.

The analysis of the protocol used by Passbolt concludes that the code implementing the OpenPGP decryption is fundamental. The cryptographic library (i.e., OpenPGP) is typically considered responsible for preventing cryptographic attacks, as opposed to the application that employs the library. With this in mind, applications should apply cryptographic primitives with robust implementations.

Ciphertexts generated by OpenPGP consist of two elements: an RSA-encrypted part and a part encrypted by a symmetric cipher. The latter contains an integrity check, since it either uses an authenticated encryption mode such as EAX or OCB or includes modification detection code (MDC).

Any information leakage that occurs prior to the symmetric encryption integrity check can potentially be utilized to perform a chosen ciphertext attack against any RSA PKCS #1 encrypted ciphertext. This encryption mode must be implemented correctly to prevent padding oracle attacks, which is achievable by restricting information leaks via error messages and timing differentiations.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Accordingly, optimal implementation of the OpenPGP primitives is essential towards preventing padding oracle attacks. Cure53's analysis of the underlying OpenPGP library confirmed that the construct can be improved. The developers adopt the *rsa#DecryptPKCS1v15* method to decrypt RSA-encrypted messages. *rsa#DecryptPKCS1v15SessionKey* is preferable in this context since a random symmetric key will be used upon incorrect RSA padding, thus obfuscating timing information from a potential attacker.

Passbolt incorporates certain countermeasures to the protocol using OpenPGP that are advantageous and elevate security resilience, including hiding error messages returned by the PGP implementation from potential attackers. Albeit, these do not represent primary safeguards against attacks of this nature.

Generally speaking, Cure53 recommends deprecating RSA PKCS #1 for encryption. Unfortunately, OpenPGP is significantly slow at adding new ciphers, thus transitions are increasingly difficult to achieve.

To finalize, Cure53 recommends raising the padding issue with the developer of the OpenPGP library. Fixing timing leaks by using appropriate methods from the RSA library should be prioritized. The root cause responsibility lies with the affected cryptographic library, which should attempt to negate as many attack strategies as possible and avoid burdening the adopting application with these requirements.

Cure53 is also open to discussing these findings with the paper's authors, which could help to strengthen additional implementations. For supplemental research, the in-house team could investigate the availability of protocols that allow ciphertext modifications leading to significant timing differences.

Cure53 would like to thank Remy Bertot and Cedric Alfonsi from the Passbolt SA team for their excellent project coordination, support, and assistance, both before and during this assignment.