Fine penetration tests for fine websites

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

Pentest-Report Silence Laboratories sl-verifiable-enc Library 06.2025

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi, J. Ginesin

Index

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

Introduction

"This library provides a generic implementation of verifiable RSA encryption. It allows for the encryption of scalar values from various elliptic curves while providing proofs of correct encryption. This is particularly useful in cryptographic protocols where you need to prove that an encrypted value corresponds to a public key without revealing the private key."

From https://github.com/silence-laborator.../sl-verifiable-enc/README.md

This report describes the results of a security assessment of the Silence Laboratories *sl-verifiable-enc* library. The project, which included a review of the component's cryptography and a dedicated source code audit, was conducted by Cure53 in June 2025.

This security examination, registered as *SIL-05*, was requested by Silence Laboratories Pte. Ltd. in May 2025 and then scheduled to start shortly thereafter in June of the same year. The project belongs to the preexisting cooperation between Cure53 and Silence Laboratories.

In terms of the exact timeline and specific resources allocated to *SIL-05*, the Cure53 team has completed their research in CW24. In order to achieve the expected coverage for this task, a total of five days were invested. A team consisting of three senior testers was formed and assigned to the preparation, execution, documentation, and delivery of this project. Given the tasks envisioned in the frames of *SIL-05*, the assessment was contained into a single work package (WP):

- **WP1:** Cryptography reviews & audits against Silence Laboratories *sl-verifiable-enc* library

The white-box methodology was used for this *SIL-05* project. All sources corresponding to the test targets were shared to ensure that the project could be executed in accordance with the agreed framework. All further means of access needed to complete the project were also supplied to Cure53 by Silence Laboratories.

The project was completed without any major issues. To facilitate a smooth transition into the testing phase, all preparations were completed in CW23. Throughout the engagement, communications were conducted through a private, dedicated, and shared Slack channel. Stakeholders - including Cure53 testers and the internal staff from Silence Laboratories - were able to participate in discussions in this space.

Cure53 did not need to ask many questions, and the quality of all project-related interactions was consistently excellent. The testers offered frequent status updates on the examination and emerging findings. Moreover, live-reporting was used during this project, specifically being executed via the Slack channel mentioned.

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

Continuous communication contributed positively to the overall results of this project. Significant roadblocks were avoided thanks to clear and careful preparation of the scope, as well as through subsequent support.

The Cure53 team achieved good coverage of the WP1 objectives, identifying seven security-affecting flaws on the delineated scope. Six of the spotted issues were classified as security vulnerabilities and one is a general weakness without much damaging potential at present.

Generally, Cure53 was able to identify several *High*-scored problems within the Silence Laboratories *sl-verifiable-enc* library. One of the identified patterns concerned side-channel vulnerabilities, which seem to be primarily due to the inconsistent prioritization of constant-time operations during development. Further, a substantive concern is the use of PKCS#1 v1.5 for RSA encryption (see SIL-05-003). The standard in use is known for significant cryptographic weaknesses. The library exhibits widespread timing vulnerabilities and allows *nonce* reuse in its proof generation mechanism, which could severely undermine the protocol's security guarantees. A misleading error message was also noted in terms of hardening recommendations.

Ultimately, while the number of findings is not excessive, the actual issues are somewhat severe and highlight areas where the implementation requires further improvements and attention. The corrective action should make sure that the library no longer deviates from best practices and incorporates protections against the mentioned vulnerability classes.

The following sections first describe the scope and key test parameters, as well as how the work package was structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. The problems are then discussed chronologically within each category. In addition to technical descriptions, PoC and mitigation advice is provided where applicable. The report ends with general conclusions relevant to this June 2025 project. Based on the test team's observations and the evidence collected, Cure53 elaborates on the overall impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Silence Laboratories *sl-verifiable-enc* library.

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

Scope

- **Cryptography reviews & code audits against Silence Laboratories *sl-verifiable-enc* library**
  - **WP1:** Cryptography reviews & audits against Silence Laboratories *sl-verifiable-enc* library
    - **Source code:**
      - **URL:**
        - https://github.com/silence-laboratories/sl-crypto/blob/main/crates/sl-verifiable-enc/README.md
      - **Branch:**
        - main
      - **Commit:**
        - *6cc7f6ac6e0a6ed9a7a585e8fde00e2e2b03db85*
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

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

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *SIL-05-001*) to facilitate any follow-up correspondence.

SIL-05-001 WP1: Reusing nonces across proof rounds weakens security *(High)*

In the *sl-verifiable-enc* crate, the prover is meant to commit independently to fresh random scalars *r.*

Consequently, the verifier's Fiat-Shamir challenge forces the prover into $2^{-SECURITY\_PARAMETER}$ soundness error.

However, the verifier does not check that the proof values are all unique. Therefore, when the same random value *r* is used by a prover as the basis for all its proof generation, either maliciously or due to error, the commitments collapse into a single triple *(gr, Enc(r), Enc(x+r))*, repeated *SECURITY_PARAMETER* times.

This does not directly reveal or bypass the hash-based challenge, but it does let a malicious prover amortize the cut-and-choose game into one global search problem, cutting practical soundness in a potentially significant fashion. In edge cases where an attacker can craft a single commitment that satisfies both halves of the verifier's conditional check (e.g., via malleability or weak group rules), soundness collapses entirely. Concretely, instead of needing to be able to find multiple malicious *r* values, an attacker needs only to find one *r* value that works for their purposes.

Affected file:
lib.rs

Affected code:

```
pub fn verify(
    &self,
    q_point: &G,
    rsa_pubkey: &RsaPublicKey,
    label: &[u8],
) -> Result<(), RsaError> {
    let challenge = Self::challenge(q_point, label, &self.proofs);
    for i in 0..self.security_param {
        let proof = &self.proofs[i];
        // No check for proof uniqueness.
[...]     } }
```

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

To improve the current design, it is recommended to:

1. **Enforce proof uniqueness check on the verifier's end** in order to detect scenarios where proofs are repeated.
2. **Include round index in randomness generation**. Derive each *r* using a *PRF* with the round index.
3. **Consider using a verifiable random function** to ensure that each *r* was generated correctly.

SIL-05-002 WP1: Proof validation failure exposes timing side-channel *(Low)*

In the *verify* function, when the verifier iterates through the array of proofs to check whether the prover's given response to each of the challenges holds, an error will be raised by the verifier on the first failing proof, as opposed to after the iteration has completed in its entirety. This will leak the index of the first failing proof check via a timing side-channel.

Affected code:

```
pub fn verify(
	&self,
	q_point: &G,
	rsa_pubkey: &RsaPublicKey,
	label: &[u8],
) -> Result<(), RsaError> {
	let challenge = Self::challenge(q_point, label, &self.proofs);
	for i in 0..self.security_param {
		let proof = &self.proofs[i];
		let open_scalar = &self.open_scalars[i];
		let scalar_expo = G::generator() * open_scalar;
		let choice_bit = challenge.extract_bit(i);
		let enc_open_scalar = rsa_encrypt_with_label(
		open_scalar.to_repr(),
			label,
			rsa_pubkey,
			self.seed,
		)?;

		let g_r_option = G::from_bytes(&proof.g_r);
		let g_r = if g_r_option.is_some().unwrap_u8() == 1 {
			g_r_option.unwrap()
		} else {
			return Err(RsaError::VerificationFailed);
		};

		// If choice bit is 0
		let cond_a = {
			let cond1 = g_r.ct_eq(&scalar_expo);
```

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

```
                        let cond2 = proof.enc_r.ct_eq(&enc_open_scalar);
                        cond1 & cond2
                };
                // If choice bit is 1
                let cond_b = {
                        let calc_scalar_expo = *q_point + g_r;
                        let cond1 = calc_scalar_expo.ct_eq(&scalar_expo);
                        let cond2 = proof.enc_x_r.ct_eq(&enc_open_scalar);
                        cond1 & cond2
                };

                let verified =
                        Choice::conditional_select(&cond_a, &cond_b,
        choice_bit)
                                .unwrap_u8();
                if verified != 1 {
                        return Err(RsaError::VerificationFailed);
                }
        }
        Ok(())
}
```

To mitigate this issue, it is advised to modify the implementation of the *verify* function to ensure an error is only returned after the loop is concluded, thus preventing the index of the proof failure from being leaked.

SIL-05-003 WP1: Vulnerable RSA standard in use *(High)*

Throughout the implementation, PKCS#1 v1.5 is employed as the scheme used for RSA encryption. PKCS#1 v1.5 is well-known to be vulnerable to chosen adaptive plaintext attacks (e.g., the Bleichenbacher attack). A malicious eavesdropper with knowledge of the verifier's and prover's agreed upon label may choose a target ciphertext and use the verifier's decrypt function as an oracle for the Bleichenbacher attack, eventually recovering original plaintexts.

The malicious eavesdropper may target *enc_x_r* and *enc_r* for some *g_r* from the *ProofData* struct, which are encrypted using PKCS#1 v1.5. These could be leveraged to break confidentiality and learn the secret material *x* via *(x + r) - r = x*. Because *ProofData* is passed from the prover to the verifier as the basis of the protocol, it is reasonable to assume an attacker may observe it in transit.

Affected code:

```
fn rsa_encrypt_with_label(
        m: impl AsRef<[u8]>,
        label: &[u8],
        rsa_pubkey: &RsaPublicKey,
        seed: [u8; 32],
) -> Result<Vec<u8>, RsaError> {
```

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

```
let mut rng = ChaCha20Rng::from_seed(seed);
let m_int = BigUint::from_bytes_be(m.as_ref());
let label_int = label_int_from_bytes(label);
let plaintext = (m_int * label_int) % rsa_pubkey.n();
rsa_pubkey
.encrypt(&mut rng, Pkcs1v15Encrypt, &plaintext.to_bytes_be())
.map_err(|_| RsaError::EncError)
```

Replacing PKCS#1 v1.5 with a more secure RSA scheme is recommended. For example, *RSA-OAEP* could be used to ensure the confidentiality of *enc_x_r* and *enc_r,* therefore guaranteeing secrecy of *x.*

SIL-05-005 WP1: Deriving security parameter from hash function *(Low)*

Testing confirmed that the verifiable RSA encryption implementation uses a hardcoded security parameter of 128 bits rather than deriving it from the underlying hash function's output size. The current implementation assumes SHA-256 usage but does not enforce or validate that the security parameter matches the hash function's security properties.

This design could lead to security degradation if the hash function is changed to one with different security properties without corresponding updates to the security parameter. While the current hard-coded value is appropriate for SHA-256, it in fact does not exploit the hash function's full potential, instead limiting the parameter to 128 bits unless it is manually specified by the user. In the latter case, it is checked against an upper bound of 256 bits. Future modifications using hash functions with smaller output sizes could compromise the scheme's security guarantees without proper validation.

To mitigate this issue, Cure53 advises implementing the following changes:

- Derive the security parameter from the hash function type at compile time.
- Add validation to ensure the derived parameter is never less than 256 bits.
- Consider making the security parameter generic over the hash function type.

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

SIL-05-006 WP1: Scalar multiplication exposes timing side-channel *(High)*

Throughout the implementation, scalar multiplication over a group using the *Group* library is employed. In particular, both the *encrypt_with_proof* function and the *decrypt* function multiply secret material with other values. However, multiplication is not constant-time by default[1].

It can be seen in the *Group* library's underpinning implementation that scalar multiplication is not implemented to be constant-time[2], due to both branching and table lookups depending on secret material. Therefore, both the *encrypt_with_proof* and *decrypt* leak information via a timing side-channel on the secret values *x*, potentially making it possible for an adversary to recover them.

Affected code:

```
pub fn encrypt_with_proof<R: CryptoRngCore>(
    x: &G::Scalar,
    rsa_pubkey: &RsaPublicKey,
    label: &[u8],
    security_param: Option<usize>,
    rng: &mut R,
) -> Result<Self, RsaError> {
    let seed = rng.gen::<[u8; 32]>();
    let security_param = security_param.unwrap_or(SECURITY_PARAM);
    // Security parameter must be at least 128 and at most 256
    if !(SECURITY_PARAM..=256).contains(&security_param) {
        return Err(RsaError::InvalidSizeParam);
    }
    let mut proofs = Vec::with_capacity(security_param);
    let q_point = G::generator() * x;
    let mut r_list = Vec::with_capacity(security_param);
    let mut x_plus_r_list = Vec::with_capacity(security_param);
```

To mitigate this issue, a constant-time scalar multiplication algorithm should be deployed throughout the implementation of the verifiable encryption library, but especially when multiplying secret material by other values. Among other solutions, NaCL's implementation of constant-time scalar multiplication is hereby endorsed and suggested as a project that could be adapted for the tested library's use-case[3].

[1] https://soatok.blog/2020/08/27/soatoks-guide-to-side-channel-attacks/
[2] https://github.com/zkcrypto/group/blob/1d3[...]4ca/src/wnaf.rs#L149-L175
[3] https://nacl.cr.yp.to/

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

SIL-05-007 WP1: Modular inverse exposes timing side-channel *(High)*

In the *sl-verifiable-enc* crate, within the *rsa_decrypt_with_label* function, *rsa_privkey* is passed through the *mod_inverse* function before being passed into the *label_int_from_bytes* function. The *mod_inverse* function, which originates from the external *num_bigint_dig* library[4], is not constant-time due to its dependence on a variable-time extended GCD implementation from the same library.[5] This, in turn, potentially means that an attacker could derive *rsa_privkey* via a timing side-channel.

Affected code:

```
fn rsa_decrypt_with_label(
       ciphertext: &[u8],
       label: &[u8],
       rsa_privkey: &RsaPrivateKey,
) -> Result<Vec<u8>, RsaError> {
       let plaintext = rsa_privkey
       .decrypt(Pkcs1v15Encrypt, ciphertext)
       .map_err(|_| RsaError::DecError)?;

       let n = rsa_privkey.n();
       let label_inv = label_int_from_bytes(label)
       .mod_inverse(n)
       .and_then(|num| num.to_biguint())
       .ok_or(RsaError::InvalidLabel)?;
```

To mitigate this issue, a constant-time modular inverse algorithm should be used in lieu of the current variable-time modular inverse algorithm. These should be done consistently for all operations pertaining to secret material. To do so, one may reference Bernstein and Yang, who describe and provide sample implementations regarding how to implement modular inverse in constant-time[6].

[4] https://docs.rs/num-bigint-dig/latest/src/num_bigint_dig/algorithms/mod_inverse.rs.html#11-25
[5] https://docs.rs/num-bigint-dig/latest/src/num_bigint_dig/algorithms/gcd.rs.html#239
[6] https://gcd.cr.yp.to/

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

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.

SIL-05-004 WP1: Misleading error type during security parameter check *(Info)*

The *encrypt_with_proof* function, the *security_param* dictates how many proofs the prover and verifier will employ in the verifiable encryption protocol. This function checks for this value to be between at least 128 and at most 256. However, when *SECURITY_PARAM is found* to be outside of the specified range, the error *InvalidSizeParam* is raised, as opposed to the expected *InvalidSecurityParam* error.

Affected file:
sl-verifiable-enc/src/lib.rs

Affected code:

```
pub fn encrypt_with_proof<R: CryptoRngCore>(
    x: &G::Scalar,
    rsa_pubkey: &RsaPublicKey,
    label: &[u8],
    security_param: Option<usize>,
    rng: &mut R,
) -> Result<Self, RsaError> {
    let seed = rng.gen::<[u8; 32]>();
    let security_param = security_param.unwrap_or(SECURITY_PARAM);
    // Security parameter must be at least 128 and at most 256
    if !(SECURITY_PARAM..=256).contains(&security_param) {
       return Err(RsaError::InvalidSizeParam); // should be
InvalidSecurityParam
    }
```

It is recommended to replace the error currently returned by the *SECURITY_PARAM* check, *InvalidSizeParam*, with the correct error, as specified by the *RsaError* object, pointing to *InvalidSecurityParam.*

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

Conclusions

This *SIL-05* project was completed by Cure53 as a focused review of the Silence Laboratories *sl-verifiable-enc* library. Five days were spent by a three-tester team on the scope in June 2025, ultimately revealing the presence of seven security issues. Not only were some problems marked as carrying *High*-level risks, but the test also exposed a pattern of side-channel vulnerabilities throughout the implementation, suggesting that constant-time operations were not prioritized during development.

The testing methodology employed both high-level protocol analysis and low-level implementation review, with particular focus on verifying that the security guarantees, made by Silence Laboratories in the context of the library, were in fact translated correctly into code. The team successfully achieved comprehensive coverage of the core cryptographic operations, the Fiat-Shamir transformation, and the interaction between RSA encryption and group operations. This thorough approach was instrumental in identifying the systemic nature of certain issues, specifically the side-channel vulnerabilities.

To offer more details on the discovered flaw, one of the more severe findings stems from the choice to use PKCS#1 v1.5 for RSA encryption (SIL-05-003). This standard has been known to be vulnerable to adaptive chosen-ciphertext attacks (e.g., the Bleichenbacher attack) for over two decades. The aforementioned design decision indicates either a lack of awareness regarding modern cryptographic standards or technical debt from earlier development phases. The fact that this vulnerability exists in a newly developed cryptographic library is particularly troubling and suggests the need for more rigorous cryptographic review during the design phase. The concentration of timing vulnerabilities across different components (SIL-05-002, SIL-05-006, SIL-05-007) reveals that side-channel resistance was not systematically considered during implementation. The library relies on external dependencies (*Group library, num_bigint_dig*) that do not provide constant-time guarantees, yet these are used directly for operations on secret material.

Beyond timing attacks, the proof generation mechanism allows for *nonce* reuse across rounds (SIL-05-001). This significantly weakens the security guarantees of the cut-and-choose protocol. In particular, the implementation oversight transforms what should be a *2^SECURITY_PARAMETER* soundness error into a potentially much weaker security bound. The issue demonstrates that while the underlying cryptographic protocol may be sound, its implementation requires careful attention to ensure preservation of security properties.

An additional design concern involves the hardcoded security parameter implementation (SIL-05-005), which assumes SHA-256 usage but does not derive parameters from the hash function's properties. While not immediately exploitable, this rigid design could lead to security degradation if hash functions are changed without corresponding parameter updates. From a meta-level perspective, this suggests a lack of cryptographic agility in the current architecture.

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

Despite these issues, it is worth noting that the scope was well-prepared and communication with the Silence Laboratories team was smooth throughout the engagement. The comprehensive documentation and clean code organization facilitated efficient testing within the limited timeframe. The library's modular structure allowed for thorough analysis, and the team was responsive to queries. It is hoped that the findings of this *SIL-05* assessment can inform improvements in the *sl-verifiable-enc* library's security posture.

Cure53 would like to thank Jay Prakash and Sushi from the SILENCE LABORATORIES PTE. LTD. team for their excellent project coordination, support and assistance, both before and during this assignment.