HashCloak

Code Review and Security Assessment
For
Vana

Initial Delivery: October 30, 2025
Updated: November 5, 2025

Prepared For:
Tim Nunamaker | *Open Data Labs*
Maciej Witowski | *Open Data Labs*

Prepared by:
Alex Yu-Chen Liao | *HashCloak Inc.*
Manish Kumar | *HashCloak Inc.*

Table Of Contents

Executive Summary

Vana engaged *HashCloak Inc.* to perform a security audit of the ECIES component in the Vana SDK, a TypeScript SDK for building data-driven applications on the Vana Network. The security audit was done from October 20th to October 29th with 2 auditors.

For this engagement, we began by reviewing the functionality and overall design of the ECIES component. We then assessed the implementation against common vulnerabilities for elliptic-curve cryptography implementations and conducted threat modeling to identify potential risks. Finally, we analyzed the codebase for potential vulnerabilities, with particular emphasis on scenarios that could result in private data leakage or break message authentication.

Our review found that the source code is well-documented, with detailed comments that aid readability and comprehension. The codebase also includes extensive tests that cover most of the ECIES components, which improves overall reliability and security.

On November 5, we reviewed the following branch and associated commits to ensure that no new issues have been introduced into the codebase. All issues found in the audit have been resolved as of commits:

- Branch: https://github.com/vana-com/vana-sdk/tree/feat/ecies-improvements
- Commits:
    - https://github.com/vana-com/vana-sdk/pull/120/commits/936c8820f8a4da78355a08541cf8352da32982fb
    - https://github.com/vana-com/vana-sdk/pull/120/commits/3b70cb318964fb7686b7adc77092036035380ff7
    - https://github.com/vana-com/vana-sdk/pull/120/commits/fa76ab40e6a7fcb0cc7b4996e626caaf34ab3c07
    - https://github.com/vana-com/vana-sdk/pull/120/commits/328e084fc76778de049d108d958bdf220edba0cf
    - https://github.com/vana-com/vana-sdk/pull/120/commits/84de68ea165e13afb05898d025760d9aa8c73c99
    - https://github.com/vana-com/vana-sdk/pull/120/commits/261ff72e44ab6cb3708ea6ebde90e34369b96c56

Overall, we found the issues range from medium to informational:

Severity	**Number of Findings**
Critical	0
High	0
Medium	3
Low	0
Informational	3

Scope

For the audit, we considered the repository https://github.com/vana-com/vana-sdk/tree/fe13efc51eec4600b27e76b5fb61e4f2996dd252/packages/vana-sdk/src/crypto/ecies at commit fe13efc51eec4600b27e76b5fb61e4f2996dd252.
For the review, we looked at PR#120 of the same repository.

Overview

The ECIES component in the Vana SDK implements the elliptic curve integrated encryption scheme using secp256k1 curve. It includes an interface of ECIES that is compatible with the eccrypto library and also provides implementations in both Node.js as well as in the browser.

Methodology

The audit was conducted through manual code review. While the primary focus was the in-scope codebase, we also inspected dependency behavior when it was used. During manual code review, we first understand the functionality of each component and then consider how these components interact with each other in the codebase. Afterwards, we begin threat modeling, assessing the areas of concern and the potential attack surface.

Overview of Evaluated Components

ECIES

- Possible leakage of private data
- Non-constant time operation on sensitive data
- Correct usage of underlying dependencies
- Possible timing attacks
- Correct parameters for the underlying curve
- Key Handling & Validation
- Use of Randomness
- Serialization and Deseralization
- Error Handling
- Clearing of sensitive data

Findings

VNA-1: Sensitive Data not cleared after use

Type: Medium
Files affected: packages/vana-sdk/src/crypto/ecies/base.ts

Description: In the functions encrypt and decrypt, sensitive data such as ephemeralPrivateKey, sharedSecret and kdf are cleared after use. However, encryptionKey and macKey created from a slice of kdf are not cleared. In Javascript/Typescript, slicing a Uint8Array creates a new instance of Uint8Array in memory containing the copy of the selected elements from the original array i.e., it gives a completely new Uint8Array and not a reference to a portion of the original array. Therefore, clearing kdf does not clear the encryptionKey and macKey and hence must be cleared separately.

Impact: The encryptionKey and macKey may remain in memory after use which can expose the derived keys.

Suggestion: Clear encryptionKey and macKey data as well in both encrypt and decrypt function.

Status: The Vana team has added functions to clear encryptionKey and macKey after use. The fix can be found in this commit.

VNA-2: Missing odd/even prefix check while normalizing public keys

Type: Medium
Files affected: packages/vana-sdk/src/crypto/ecies/base.ts

Description: The normalizePublicKey function is used to normalize a public key into an uncompressed format. First, the function checks whether the public key is already in uncompressed form by verifying that its length equals UNCOMPRESSED_PUBLIC_KEY_LENGTH (65 bytes) and that the first byte matches

the UNCOMPRESSED prefix (0x04).

```
if (publicKey.length === CURVE.UNCOMPRESSED_PUBLIC_KEY_LENGTH) {
    if (publicKey[0] !== CURVE.PREFIX.UNCOMPRESSED) {
      .
      .
      .
   }
}
```

If this condition fails, the function checks the condition for the compressed key and checks whether its length equals COMPRESSED_PUBLIC_KEY_LENGTH (33 bytes).

```
if (publicKey.length === CURVE.COMPRESSED_PUBLIC_KEY_LENGTH)
```

However, it does not verify whether the prefix is one of the valid compressed prefixes COMPRESSED_EVEN (0x02) or COMPRESSED_ODD (0x03).

Impact: A malformed key could be incorrectly treated as a valid compressed public key.

Suggestion: In addition to the length check, check for offset should also be added i.e., the if condition should be changed to:

```
if (publicKey.length === CURVE.COMPRESSED_PUBLIC_KEY_LENGTH &&
(publicKey[0] === CURVE.PREFIX.COMPRESSED_EVEN || publicKey[0] ===
CURVE.PREFIX.COMPRESSED_ODD))
```

Status: The Vana team has added an odd/even prefix check while normalizing public keys in this commit.

VNA-3: Missing odd/even prefix check while determining ephemeral public key size based on prefix

Type: Medium
Files affected: packages/vana-sdk/src/crypto/ecies/interface.ts
Description: In function deserializeECIES, the ephemeral key size is calculated based on the prefix present

```
const ephemKeySize =
   bytes[FORMAT.EPHEMERAL_KEY_OFFSET] ===
CURVE.PREFIX.UNCOMPRESSED
     ? CURVE.UNCOMPRESSED_PUBLIC_KEY_LENGTH
     : CURVE.COMPRESSED_PUBLIC_KEY_LENGTH;
```

Here, the function checks at the key offset whether the prefix is equal to the UNCOMPRESSED (0x04) value. If it is not, it assumes by default that the key is in compressed format. However, it fails to verify that when the prefix is not equal to UNCOMPRESSED, it must be either COMPRESSED_EVEN (0x02) or COMPRESSED_ODD (0x03) before attempting deserialization.

Impact: It can deserialize malformed data which can cause unexpected errors or crashes.

Suggestion: Explicitly verify that the prefix byte is one of the COMPRESSED_EVEN or COMPRESSED_ODD for the case when the offset is not uncompressed.

Status: The Vana team has updated prefix checks against ephemeral public keys in deserializeECIES. The fix can be found in this commit.

VNA-4: Incomplete validation checks in VALIDATION constant object helper functions

Type: informational
Files affected: packages/vana-sdk/src/crypto/ecies/constants.ts

Description: In constant object VALIDATION, the helper functions isValidPublicKey, isCompressedPublicKey and is UncompressedPublicKey only check whether the key length and compressed/uncompressed prefixes are correct or not but fails to check whether the point is on the curve or not. We did not find their uses in the current scope of audit. In the current scope, the public keys validation is done using the validatePublicKey function with additional length and prefix checks and the functions in VALIDATION are not used. If the intention is to use it in the current codebase then the helper functions should be fixed with additional check of point being on the curve otherwise should be removed from the codebase to avoid any confusion.

Impact: A malformed key could be incorrectly treated as a valid public key.

Suggestion: If the functions are not in use in the current codebase then they must be removed. If the intention is to use them in the current codebase then the check of point being on the curve should be added in addition to the length and prefix check.

Status: The Vana team has deleted unused codes in this commit.

VNA-5: Ambiguity in deserializing ECIESEncrypted

Type: Informational
Files affected: packages/vana-sdk/src/crypto/ecies/interface.ts

Description: In function deserializeECIES, while deserializing ECIESEncrypted, both compressed and uncompressed ephemeral public key sizes are considered, however, the TypeScript interface ECIESEncrypted which represents the ECIES encrypted data considers 65 bytes of uncompressed Ephemeral public key as mentioned in the comment.

```
export interface ECIESEncrypted {
  /** Initialization vector (16 bytes) */
  iv: Uint8Array;
  /** Ephemeral public key (65 bytes uncompressed) */
  ephemPublicKey: Uint8Array;
```

```
  /** Encrypted data */
  ciphertext: Uint8Array;
  /** Message authentication code (32 bytes) */
  mac: Uint8Array;
}
```

This ambiguity can cause confusion and lacks uniformity across the codebase. If the intention is to use both compressed and uncompressed ephemeral public keys then the comment must be updated. If the intention is to only use the uncompressed public key then we must only take into account the uncompressed key size i.e. 65 bytes.

Impact: May deserialize unsupported ECIES encrypted data.

Suggestion: If the intended behavior is to allow both compressed and uncompressed ephemeral public keys then the code comment must be updated but if only uncompressed key is allowed then the function should be modified to take only uncompressed public keys with fixed key size of 65 bytes.

Status: The Vana team has updated deserializeECIES to only accept uncompressed keys in this commit.

VNA-6: Avoid using has( ) while reading from validated public keys cache

Type: informational

Files affected: packages/vana-sdk/src/crypto/ecies/base.ts

Description: In function normalizePublicKey, it is first checked whether the public key is present in the validated public key cache or not.

```
// Check cache first
   if (BaseECIESUint8.validatedKeys.has(publicKey)) {
```

```
    return publicKey;
  }
```

The check is done using .has() which only checks if the key exists in the map or not but does not check whether the value is true or false. While this does not cause any issue in the current codebase as only the valid keys are set in the map with value true. Using get() instead of has() is encouraged as it retrieves the value as well thus can prevent future updates that some of the keys can be false (i.e. in the case that some keys need to be revalidated), since the if statement will fail for either undefined or false.

Impact: If in future updates where some public keys can be changed to false in validatedKeys map, the check in the function could fail.

Suggestion: Use get() instead of has().

Status: The Vana team has updated the cache check to use get() instead of has() in this commit.

References

- https://www.secg.org/SEC1-Ver-1.0.pdf
- https://www.shoup.net/iso/std4.pdf

Appendix

Test for VNA-3

In file packages/vana-sdk/src/crypto/ecies/__tests__/compatibility.test.ts

```
it("AUDIT: should reject tampered ephemeral key", async () => {
    const privateKey = new Uint8Array(
      Buffer.from(
        "0000000000000000000000000000000000000000000000000000000000012345",
        "hex",
      ),
    );
    const publicKeyBrowser = browserProvider.createPublicKey(privateKey, false);
    const publicKeyNode = nodeProvider.createPublicKey(privateKey, false);

    // check that both providers generate the same public key
    expect(publicKeyBrowser).toEqual(publicKeyNode);

    const message = new TextEncoder().encode("Browser to Node test");

    // Happy path: Browser encrypts, Node decrypts
    const encrypted = await browserProvider.encrypt(publicKeyBrowser, message);
    // Serialize to raw format and deserialize to simulate data transfer
    const data = serializeECIES(encrypted);
    const ecies = deserializeECIES(data);

    const decrypted = await nodeProvider.decrypt(privateKey, ecies);
    expect(decrypted).toEqual(message);


    // Tamper with the ephemeral public key prefix byte
```

```
    const tamperedEphemKey = new Uint8Array(encrypted.ephemPublicKey);
    tamperedEphemKey[0] ^= 0xff; // Flip bits to corrupt

    const tamperedEncrypted: ECIESEncrypted = {
      ...encrypted,
      ephemPublicKey: tamperedEphemKey,
    };

    // Serialize and deserialize tampered data
    // ISSUE: No error is thrown during serialization/deserialization even when prefix is
tampered - only during decryption
    const serialized = serializeECIES(tamperedEncrypted);
    const deserialized = deserializeECIES(serialized);

    // Attempt decryption with tampered ephemeral key
    // Node provider should reject tampered ephemeral key
    await expect(
      nodeProvider.decrypt(privateKey, deserialized),
    ).rejects.toThrow(/Invalid ephemeral public key/i);

    // Browser provider should also reject tampered ephemeral key
    await expect(
      browserProvider.decrypt(privateKey, deserialized),
    ).rejects.toThrow(/Invalid ephemeral public key/i);

  });
```