The PEACE Protocol[1]

A protocol for transferable encryption rights.

Logical Mechanism LLC[2]

October 30, 2025

[1]This project was funded in Fund 14 of Project Catalyst.
[2]Contact: support@logicalmechanism.io

Contents

1 Abstract

In this report, we introduce the PEACE protocol, an ECIES-based, multi-hop, bidirectional proxy re-encryption scheme for Cardano. PEACE solves the encrypted-NFT problem by providing a decentralized, open-source protocol for transferable encryption rights, enabling creators, collectors, and developers to manage encrypted NFTs without relying on centralized decryption services. This work fills a significant gap in secure, private access to NFTs on Cardano. The PEACE protocol was funded in round 14 of Project Catalyst[1].

2 Introduction

The encrypted NFT problem is one of the most significant issues with current NFT standards on Cardano. Either the data is not encrypted, available to everyone who views the nft, or the data encryption requires some form of centralization, some company doing the encryption on behalf of users. Current solutions [1] claim to offer decentralized encrypted assets (DEA), but lack a publicly available, verifiable cryptographic protocol or an open-source implementation. Most, if not all, of the mechanics behind current DEA solutions remain undisclosed. This report aims to fill that knowledge gap by providing an open-source implementation of a decentralized encryption protocol for encrypted assets on Cardano.

The encryption protocol must allow tradability of both the NFT itself and the right to decrypt the NFT data, implying that the solution must involve smart contracts and a form of encryption that allows data to be re-encrypted for another user without revealing the encrypted content in the process. The contract side of the protocol should be reasonably straightforward. It needs a way to price a token, to hold the encrypted data, and allow other users to purchase the token. To ensure tradability, the tokens may need to be soulbound. On the other side of the protocol is the encryption required to make this all work. Luckily, this type of encryption has been in cryptography research for quite some time [2] [3] [4]. There are even patented cloud-based solutions already in existence [5]. There is no open-source, fully on-chain, decentralized encryption protocol for encrypting NFT data on Cardano. The PEACE protocol aims to solve this problem.

The solution the PEACE protocol will implement is an ambitious yet well-defined bidirectional multi-hop proxy re-encryption scheme that uses ECIES [6] and AES [7]. Bidirectionality means that Alice may re-encrypt for Bob, and sequentially Bob may re-encrypt back to Alice. Bidirectionality is important for tradability, as there should be no restriction on who can purchase the NFT. Multi-hop means that the flow of encrypted data from Alice to Bob to Carol, and so on, does not end, in the sense that it cannot be re-encrypted for a new user. Multi-hopping is important for tradability, as a single tradable asset does not fit many use cases. An asset should always be tradable if the user wants to trade it. The encryption mechanisms used in the protocol are considered industry standards at the time of this report.

The remainder of this report is as follows. Section 4 discusses the preliminaries and background required for this project. Section 5 will be a brief overview of the required cryptographic primitives. Section 6 will be a detailed description of the protocol. Sections 7, 8, and 9 will dive into the security and threat analysis, and limitations of the protocol, respectively. The goal of this report will be to serve as a go-to reference and description of the PEACE protocol.

[1] https://projectcatalyst.io/funds/14/cardano-use-cases-concepts/decentralized-on-chain-data-encryption

3 Background And Preliminaries

Understanding the protocol will require some technical knowledge of modern cryptographic methods, a small amount of the arithmetic of elliptic curves, and just a pinch of knowledge about smart contracts on Cardano. Anyone comfortable with these topics will find this report very useful and easy to follow. The report will attempt to use research standards for terminology and notation. The elliptic curve used in this protocol will be BLS12-381 [8]. Aiken [9] is used to write all required smart contracts for the protocol.

Table 1: Symbol Description [10]

Symbol	Description
p	A prime number
$\mathbb{F}_p$	The finite field of characteristic p
$E(\mathbb{F}_p)$	An elliptic curve E defined over $\mathbb{F}_p$
E'	A twisted elliptic curve of E
$\#E(\mathbb{F}_p)$	The order of $E(\mathbb{F}_p)$ (also denoted n)
r	A prime number dividing $\#E(\mathbb{F}_p)$
δ	A non-zero integer in $\mathbb{Z}_n$
$\mathcal{O}$	The point at infinity of an elliptic curve E
$\mathbb{G}_1$	A subgroup of order r of $E(\mathbb{F}_p)$
$\mathbb{G}_2$	A subgroup of order r of the twist $E'(\mathbb{F}_{p^2})$
$\mathbb{G}_T$	The multiplicative target group of the pairing: $\mu_r \subset \mathbb{F}_{p^{12}}$
$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$	A type-3 bilinear pairing
R	A random oracle for the Fiat-Shamir transform
H_κ	A hash to group function for $\mathbb{G}_\kappa$

The protocol, both the on-chain and off-chain components, will make heavy use of the `Register` type. The `Register` stores a generator, $g \in \mathbb{G}_1$ and the corresponding public value $u = [\delta]g$ where $\delta \in \mathbb{Z}_n$ is a secret. We shall assume that the hardness of ECDLP and CDH in $\mathbb{G}_1$ and $\mathbb{G}_2$ will result in the inability to recover the secret $\delta \in \mathbb{Z}_n$. When using a pairing, we additionally rely on the standard bilinear Diffie-Hellman assumptions over ($\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$). We will represent the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with additive notation and $\mathbb{G}_T$ with multiplicative notation.

The `Register` type in Aiken:

```
pub type Register {
  // the generator, #<Bls12_381, G1>
  generator: ByteArray,
  // the public value, #<Bls12_381, G1>
  public_value: ByteArray,
}
```

Where required, we will verify Ed25519 signatures [11] as a cost-minimization approach; relying solely on pure BLS12-381 for simple signatures becomes costly on-chain. There will be instances where the Fiat-Shamir transform [12] will be applied to a Σ-protocols for non-interactive purposes. In theses cases, the hash function will be the Blake2b-256 hash function [13].

4 Cryptographic Primitives Overview

This section provides brief explanations of the cryptographic primitives required by the protocol. Where applicable, an algorithm describing the primitives will be in its respective algorithm segment. The `Register` type will be a tuple, $(\ g, u\ )$, for simplicity inside the algorithms. We shall assume that the decompression of the $\mathbb{G}_1$ points is a given. Proofs for many algorithms are in Appendix A.

There may be instances where we need to create a new `Register` from an existing `Register` [14] via a re-randomization. The random integer δ' is considered toxic waste. Randomization allows a public register to remain stealthy, which can be beneficial for data privacy and ownership.

Algorithm 1: Re-randomization of the Register type

Input: $(\ g, u\ )$ where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$
Output: $(\ g', u'\ )$

select a random $\delta' \in \mathbb{Z}_n$
compute $g' = [\delta']g$ and $u' = [\delta']u$
output $(\ g', u'\ )$

The protocol may require proving knowledge of a user's secret using a Schnorr Σ-protocol [15] [16]. This algorithm is perfectly complete and zero-knowledge, which is precisely what we need in this context. We can use simple Ed25519 signatures for spendability, and then use the Schnorr Σ-protocol for knowledge proofs for encryption. We will make the protocol non-interacting via the Fiat-Shamir transform.

Algorithm 2: Non-interactive Schnorr's Σ-protocol for the discrete logarithm relation

Input: $(\ g, u\ )$ where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$
Output: bool

select a random $\delta' \in \mathbb{Z}_n$
compute $a = [\delta']g$
calculate $c = R(g, u, a)$
compute $z = \delta * c + \delta'$
output $[z]g = a + [c]u$

There will be times when the protocol requires proving some equality using a pairing. In these cases, we can use something akin to the BLS signature, allowing only someone with the knowledge of the secret to prove the pairing equivalence. BLS signatures are a straightforward but important signature scheme for the protocol, as it allows public confirmation of knowledge of a complex relationship beyond the limits of Schnorr's Σ-protocol. BLS signatures work because of the bilinearity [17] of the pairing.

Algorithm 3: Boneh-Lynn-Shacham (BLS) signature method

Input: $(\ g, u, c, w, m\ )$ where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$, $c = H_2(m) \in \mathbb{G}_2$, $w = [\delta]c \in \mathbb{G}_2$, and $m \in \{0, 1\}^*$
Output: bool

$e(u, c) = e(g, w)$
$e(q^\delta, c) = e(q, c^\delta) = e(q, c)^\delta$

4.1 ECIES + AES-GCM

The Elliptic Curve Integrated Encryption Scheme (ECIES) is a hybrid protocol involving asymmetric cryptography with symmetric ciphers. The encryption used in ECIES is the Advanced Encryption Standard (AES). ECIES and AES combined with a key derivation function (KDF) like Argon2 [18] create a complete encryption system.

Algorithm 4: Encryption using ECIES + AES

Input: (g, u) where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$, m as the message
Output: (r, c, h)
select a random $\delta' \in \mathbb{Z}_n$
compute $r = [\delta']g$
compute $s = [\delta']u$
generate $k = KDF(s|r)$
encrypt $c = AES(m, k)$
compute $h = BLAKE2b(m)$
output (r, c, h)

Algorithm 5: Decryption using ECIES + AES

Input: (g, u) where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$, (r, c, h) as the cypher text
Output: (m,bool)
compute $s' = [\delta]r$
generate $k' = KDF(s'|r)$
compute $m' = AES(c, k')$
compute $h' = H(m')$
output (m', h' = h)

4.2 Proxy Re-Encryption

5 Protocol Overview

5.1 Design Goals And Requirements

5.2 On-Chain And Off-Chain Architecture

5.3 Key Management And Identity

5.4 Protocol Specification

6 Security Model

6.1 Trust Model

6.1.1 Assumptions

7 Threat Analysis

7.1 Metadata Leakage

8 Limitations And Risks

8.1 Performance And On-Chain Cost

9 Conclusion

A Appendix A - Proofs

Lemma A.1. *Algorithm 1 re-randomizes a* `Register`*.*

Proof. We start with $(\ g, u\ )$ where $g \in \mathbb{G}_1$ and $u = [\delta]g \in \mathbb{G}_1$ and we are given $(\ g', u'\ )$. Let us assume that $k \in \mathbb{Z}_n$ is a random integer.

Let's assume $g' = [k]g$ and $u' = [k]u$. We know $u = [\delta]g$ so $u' = [k][\delta]g = [\delta][k]g = [\delta]g'$.

Thus the discrete-logrithm relation that binds $(\ g, u\ )$, $u = [\delta]g$, is the same relationship between $(\ g', u'\ )$, $u' = [\delta]g'$.

□

Lemma A.2. *Correctness for Algorithm 2, a non-interactive Schnorr's Σ-protocol for the discrete logarithm relation.*

Proof. We start with $(\ g, u, a, z\ )$ where $g \in \mathbb{G}_1$, $u = [\delta]g \in \mathbb{G}_1$, $a \in \mathbb{G}_1$, and $z \in \mathbb{Z}_n$. Let us assume that $z = r + c * \delta$ and $a = [r]g$.

Use the Fiat-Shamir transform to generate a challenge value $c = R(g, u, a)$.

$[z]g = [r + c * x]g$

$[z]g = [r]g + [c][x]g$

$[z]g = a + [c]u$

An honest `Register` can produce an a and z that will satisfy $[z]g = a + [c]u$ proving knowledge of the secret $/delta$.

□

Bibliography

[1] J. Stone, “Stuff.io whitepaper 1.0.” Accessed: Oct. 25, 2025. [Online]. Available: https://book-io.medium.com/stuff-io-whitepaper-1-0-9529db7cdeaf

[2] M. Mambo and E. Okamoto, “Proxy cryptosystems: Delegation of the power to decrypt ciphertexts,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E80–A, no. 1, pp. 54–63, 1997, Available: https://search.ieice.org/bin/summary.php?id=e80-a_1_54

[3] M. Blaze, G. Bleumer, and M. Strauss, “Divertible protocols and atomic proxy cryptography,” in *EUROCRYPT 1998*, Springer, 1998. doi: 10.1007/BFb0054122.

[4] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage,” in *NDSS 2005*, 2005. Available: https://www.ndss-symposium.org/ndss2005/improved-proxy-re-encryption-schemes-applications-secure-distributed-storage/

[5] IronCore Labs, *Recrypt (rust): Transform/proxy re-encryption library*. Accessed: Oct. 25, 2025. [Online]. Available: https://github.com/IronCoreLabs/recrypt-rs

[6] *IEEE standard specifications for public-key cryptography—amendment 1: Additional techniques*. IEEE, 2004.

[7] *Advanced encryption standard (AES)*. NIST, 2001.

[8] S. Bowe, “BLS12-381: New zk-SNARK elliptic curve construction.” Accessed: Oct. 25, 2025. [Online]. Available: https://electriccoin.co/blog/new-snark-curve/

[9] Aiken Language Contributors, “Aiken: The modern smart contract platform for cardano.” Accessed: Oct. 26, 2025. [Online]. Available: https://aiken-lang.org/

[10] N. El Mrabet and M. Joye, Eds., *Guide to pairing-based cryptography*. in Chapman & hall/CRC cryptography and network security. New York: Chapman; Hall/CRC, 2017, p. 420. doi: 10.1201/9781315370170.

[11] S. Josefsson and I. Liusvaara, *Edwards-curve digital signature algorithm (EdDSA)*. IETF, 2017. Available: https://www.rfc-editor.org/rfc/rfc8032

[12] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Advances in cryptology – CRYPTO ’86*, in Lecture notes in computer science, vol. 263. Springer, 1986, pp. 186–194. doi: 10.1007/3-540-47721-7_12.

[13] J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, *The BLAKE2 cryptographic hash and message authentication code (MAC)*. IETF, 2015. Available: https://www.rfc-editor.org/rfc/rfc7693

[14] N. A. Given, “Sigmajoin: Outsourceable zerojoin.” Accessed: Oct. 25, 2025. [Online]. Available: https://github.com/ergoplatform/ergo-jde/blob/main/kiosk/src/test/scala/kiosk/mixer/doc/main.pdf

[15] J. Thaler, “Proofs, arguments, and zero-knowledge,” *Foundations and Trends in Privacy and Security*, vol. 4, no. 2–4, pp. 117–660, 2022, doi: 10.1561/3300000030.

[16] C.-P. Schnorr, “Efficient signature generation by smart cards,” *Journal of Cryptology*, vol. 4, pp. 161–174, 1991, doi: 10.1007/BF00196725.

[17] A. J. Menezes, *Elliptic curve public key cryptosystems*, vol. 234. in The springer international series in engineering and computer science, vol. 234. Boston, MA: Kluwer Academic Publishers, 1993. doi: 10.1007/978-1-4615-3198-2.

[18] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, *Argon2 memory-hard function for password hashing and proof-of-work applications*. IRTF CFRG, 2021. Available: https://www.rfc-editor.org/rfc/rfc9106