# Encryption with Poseidon

Dmitry Khovratovich

Ethereum Foundation and Dusk Network

December 26, 2019

## 1 Introduction

We show how to securely encrypt with Poseidon in the context of EC-based Diffie-Hellman (ECDH).

Throughout the text, we denote $i$-th element of a $n$-element tuple $X$ by $X[i], 0 \leq i \leq (n-1)$.

## 2 Scenario

Let $B \in \mathbb{F}_q^2$ be a curve point of prime order $p$. Let $(k, [k]B)$ be a keypair with $k < p$ as a private key.

We consider the following scenario:

1. Alice approaches Bob, who has public key $K$.
2. Alice selects message $M$ of $l$ $\mathbb{F}_q$ elements long.
3. Alice selects nonce $N$ (can be a timestamp).
4. Alice generates a shared secret key $k_S$ using $K$.
5. Alice encrypts $M$ using $N, k_S$ with an authenticated encryption scheme $\mathcal{E}$ and gets ciphertext $C$.
6. Alice sends $C$ and additional information $\mathcal{A}$, needed to decrypt $C$, to Bob.
7. Bob decrypts $C$ and processes $M$. The ciphertext integrity property of $\mathcal{E}$ ensures it has not been modified in between.

## 3 Design

We suggest using Poseidon permutation[1] $\mathcal{P}$ of width 4, which maps $\mathbb{F}_q^4$ to itself, in the DuplexSponge mode.

### 3.1 Shared key

To generate a key shared with Bob whose public key is $K = [k]B$, Alice proceeds as follows:

1. Generate random $r < p$;
2. Generate expanded encryption key

$$k_S \leftarrow [r]K = (k_S[0], k_S[1]) \in \mathbb{F}_q^2.$$

[1]The S-box type and the number of rounds depend on $q$. For example the S-box $x^5$ is recommended for $q$ being the prime subgroup size of curve BN254.

## 3.2 Encryption

In order to encrypt message $M \in \mathbb{F}_p^l$ of length $l$ with nonce $N < 2^{128}$ using shared key $k_S$, Alice proceeds:

1. Create Poseidon input state $S = (0, k_S[0], k_S[1], N + l * 2^{128}) \in \mathbb{F}_q^4$.
2. Repeat for $0 \leq i\lceil l/3 \rceil$:
   (a) Iterate Poseidon on $S$:
   $$S \leftarrow \mathcal{P}(S);$$
   (b) Absorb three elements of message (in the last iteration the missing elements are set to 0):
   $$S[1] \leftarrow S[1] + M[3i]; \quad S[2] \leftarrow S[2] + M[3i+1]; \quad S[3] \leftarrow S[3] + M[3i+2].$$
   (c) Release three elements of ciphertext:
   $$C[3i] \leftarrow S[1]; \quad C[3i+1] \leftarrow S[2]; \quad C[3i+2] \leftarrow S[3].$$
3. Iterate Poseidon on $S$ last time:
   $$S \leftarrow \mathcal{P}(S);$$
4. Release last ciphertext element:
   $$C.add(S[1]).$$

### 3.3 Transmission

Alice sends $(C, \mathcal{A} = ([r]B, N, l))$ to Bob.

### 3.4 Decryption

Bob obtains $(C, A', N, l)$ and decrypts as follows:

(a) Generate $k_S \leftarrow [k]A'$.
(b) Create Poseidon input state $S = (0, k_S[0], k_S[1], N + l * 2^{128})$.
(c) Repeat for $0 \leq i\lceil l/3 \rceil$:
   i. Iterate Poseidon on $S$:
   $$S \leftarrow \mathcal{P}(S);$$
   ii. Release three elements of message:
   $$M[3i] \leftarrow S[1] + C[3i]; \quad M[3i+1] \leftarrow S[2] + C[3i+1]; \quad M[3i+2] \leftarrow S[3] + C[3i+2].$$
   iii. Modify state:
   $$S[1] \leftarrow C[3i]; \quad S[2] \leftarrow C[3i+1]; \quad S[3] \leftarrow C[3i+2].$$
   iv. If 3 does not divide $l$ then check that the last $3 - (l \bmod 3)$ elements of $M$ are 0. If not, reject the ciphertext.
(d) Iterate Poseidon on $S$ last time:
   $$S \leftarrow \mathcal{P}(S);$$
(e) Check last ciphertext element:
   $$C.last = S[1].$$