Iris Protocol

Draft 0.0.1

A decentralized data exchange protocol

Tony Riemer
driemworks@idealabs.network
Ideal Labs, 2022

Abstract. This work describes the Iris protocol, a **decentralized data exchange protocol**. The protocol enables a secret sharing mechanism using *threshold proxy re-encryption* and *elliptic curve Diffie-Hellman* to allow a participant in the system to encrypt their data once and, by delegating re-encryption rights to a proxy, to non-interactively grant decryption rights to other participants in the future. The protocol readily lends itself to integration with a distributed ledger system, such as a blockchain, which allows for participants to exchange decryption rights for tokens within that blockchain system.

1 Introduction

In today's digital age, data is a valuable asset that is often stored and managed by centralized authorities, such as social media platforms and cloud storage providers. These centralized systems often have strict controls over the access and use of data, and can make it difficult for data owners and consumers to interact without the need for intermediaries. As such, the creators of content online generally do not reap the full benefits of the value created by their data. Further, this centralized model exposes user data to exposure through data breaches or other malicious activities. If the central data repository experiences a technical failure or goes offline, all of the data stored within it may become inaccessible. Additionally, centralized data management can also lead to a lack of transparency and accountability, as the organization or individual controlling the central repository has full control over the data and may be able to manipulate or misuse it without being held accountable.

To address these challenges, we propose a new encryption protocol that uses a combination of threshold proxy reencryption and ECDH (Elliptic Curve Diffie-Hellman) to enable a non-interactive, asynchronous, and asymmetric decentralized secret data sharing mechanism. Our protocol relies on a **proxy** to reencrypt secret keys. By integrating this protocol into a decentralized network, such as a blockchain, we can create unique ownership and access models for data that allow data owners and consumers to interact directly, without the need for centralized intermediaries.

In this whitepaper, we will provide a brief overview of the cryptographic protocols and primitives in use by Iris. Then, we describe the details of our proposed protocol and explain how it can be integrated into a decentralized network. We will also discuss the potential benefits and limitations of this approach, and outline how it can be used to create new and innovative solutions for data ownership and access in the decentralized ecosystem.

2 Background

In this section we provide a brief overview of the cryptographic mechanisms that are used in our protocol.

2.1 Cryptography Primer

We provide a brief overview of the cryptographic systems that are in use by the protocol. We will not go into depth on the specifics of each protocol, and instead we defer to the source material for an in-depth understanding of how each encryption mechanism functions. There are two protocols in use by Iris, firstly Elliptic curve Diffie-Hellman, which is used to distribute certain secret data to validators as part of an intermediate step, and proxy reencryption.

2.1.1 Diffie-Hellman and ECDH

The Diffie-Hellman key exchange can be described as a hybrid cryptosystem which combines the ideas of public-key cryptography along with a symmetric cryptosystem to transmit secret messages [1]. Diffie-Hellman is formulated thanks to the discrete logarithm problem (where Z represents the integers):

Definition 1: *Let* $x,\ z\ \in (Z/pZ)^*$ *where z belongs to the cyclic subgroup generated by x. Find an integer y such that:*

$$xy\ =\ z\ mod\ p$$

The idea behind this is that it is really *difficult* to find such a number over a finite field (in an infinite field, this is not the case). With this, two actors, say Alice and Bob, can share a secret over an insecure channel without risking its exposure. We leave a full treatment of how this works to [1]. Elliptic curve cryptography builds on this same idea, however, the discrete logarithm problem is formulated to function on an elliptic curve rather than the integers. First, an elliptic curve is defined as:

Definition 2: *An elliptic curve E over a field K is a cubic curve that consists of the points (x, y) satisfying the equation:*

$$y^2\ =\ x^3\ +\ ax\ +\ b$$

together with an element 0 *called "the point at infinity".*

With this definition, we can reformulate definition 1 to be defined over an elliptic curve, E, rather than over the field $(Z/pZ)^*$.

Definition 3: *Let G denote the abelian group formed by the points on the elliptic curve E with a cross product function defined,* $\oplus$*, and let E be an elliptic curve defined over a finite field. If* $P \in G$ *and R = kP is a multiple of P where k is a scalar, we define the discrete logarithm problem on E as follows. Find an integer k such that kP = R where P is a generator point on E and R belongs to the cyclic subgroup generated by P.*

As before, two actors Alice and Bob can now generate a keypair, which consists of a public key and a secret key, by choosing points on an elliptic curve rather than over a finite field. We will leave the full treatment of how actors communicate and generate cryptographic primitives to the source material.

2.1.2 Threshold Proxy Re-Encryption

Threshold Proxy Reencryption is an encryption technique based on Proxy Reencryption. In proxy reencryption, first defined in Shamir's secret sharing mechanism [2], two parties trust an intermediary, the proxy, to be able to reencrypt data. Specifically, our protocol leverages the approach by Nuñez [3]. We leave the full treatment to [3] as the literature is somewhat out of scope for this paper. At a high level, proxy reencryption functions by adding third party, called the proxy (let's name him Charlie), who is given a 'reencryption key' by Alice. Later on, another node Bob can ask Charlie to reencrypt the key so that Bob can use his secret key to decrypt the data.

An analogy for this type of secret sharing is like having a five word sentence, and you distribute two words from the sentence to five participants. For example, your sentence could be "This is not a bike", which is split into sets [{"this" , "is"}, {"this", "bike"}, {"not", "is"}, {"is", "a"}, {"not", "bike"}] and distributed to five individuals. Each set of words by itself is meaningless, and so each participant has only a slice of knowledge of the secret. Later, Bob collects words from only three participants, and is able to reconstruct the message: "This is not a bike".

2.2 Terminology

- We use the terminology $enc_A^{\ B}(plaintext)$ to indicate that the plaintext has been encrypted using A's public key and B's private key. In the case where B is an ephemeral private key, we omit it complete and simply write $enc_A\ (plaintext)$. In general, we assume this structure contains the ciphertext, public key, and nonce of the encrypted data.
- We use the terms *network, blockchain,* and *distributed ledger* interchangeably.

3 Secret Key Exchange

In the following, we construct a mechanism, leveraging $\{m,\ n\}$-threshold proxy reencryption, X25519 elliptic curve Diffie-Hellman function and the XSalsa20Poly1305 authenticated encryption cipher, to allow for the delegation of decryption rights to a to-be-determined recipient.

Assumptions

Assume that each participant in the system can be uniquely identified by an X25519 keypair. Suppose Olive owns some data, d, and wants to provide access to her data to another entity in the future whose identity she does not know yet. We also make the assumption that there is some external channel or automated control

mechanism that parties can interact with asymmetrically in order to approve decryption delegation. For example, if in a blockchain, this could look like a suite of smart contracts.

Encryption

In the encryption phase, Olive encrypts her plaintext d and delegates decryption rights to a trusted entity, Paul. So assume that Olive chooses some participant, Paul, to act as a proxy. That is, Olive delegates reencryption rights to Paul and Paul accepts the responsibility of generating key fragments when Olive grants access to new participants.

1. Olive generates a new keypair, (PK_d, SK_d).
2. Olive encrypts her plaintext, d, using the public key PK_d. This creates ciphertext C_P and capsule CAP_P.
3. Olive encrypts the secret key SK_d using Paul's public key PK_P, resulting in ciphertext $enc_P\ (SK_d)$.

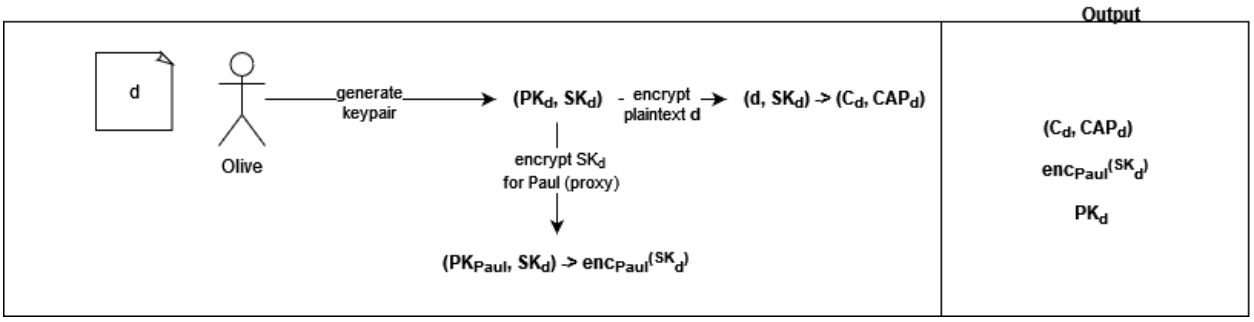

Fig. 1

Key Fragment Generation

When Olive grants access to Charlie via the external control mechanism, Paul, as the proxy, receives Paul's public key, generates key fragments, encrypts them, and distributes them to other available entities. Once complete, Herbert will have successfully delegated decryption rights to Charlie. For now, assume that Alice has given Bob authorization and Bob is requesting decryption rights from Herbert.

1. Charlie creates an ephemeral x25519 keypair, $(PK_{ephem, C}, SK_{ephem, C})$ and shares $PK_{ephem, C}$ publicly.
2. Paul decrypts the ciphertext $enc_P\ (SK_d)$ to recover the secret key SK_d.
3. Paul generates a new secp256k1 keypair (PK_{ephem}, SK_{ephem}) and a signer, with a verifying key PK_{signer}.
4. Paul generates m key fragments, $\{k_{i, d, C}\}_{i=1}^{m}$, using PK_{ephem}.
5. Paul chooses m other participants who are willing to receive a key fragment, $\{v_1, \ldots, v_m\}$.
6. For each v_i, Paul generates an ephemeral keypair $(PK_{ephem, i}, SK_{ephem, i})$. Using PK_{u_i} and $SK_{ephem, i}$, Paul encrypts the key fragment assigned to v_i, generating $enc_{v_i}(k_{i, d, C})$, which Paul shares publicly.

7. Paul encrypts the secret key SK_{ephem} using $PK_{ephem, C}$ and publishes it publicly.

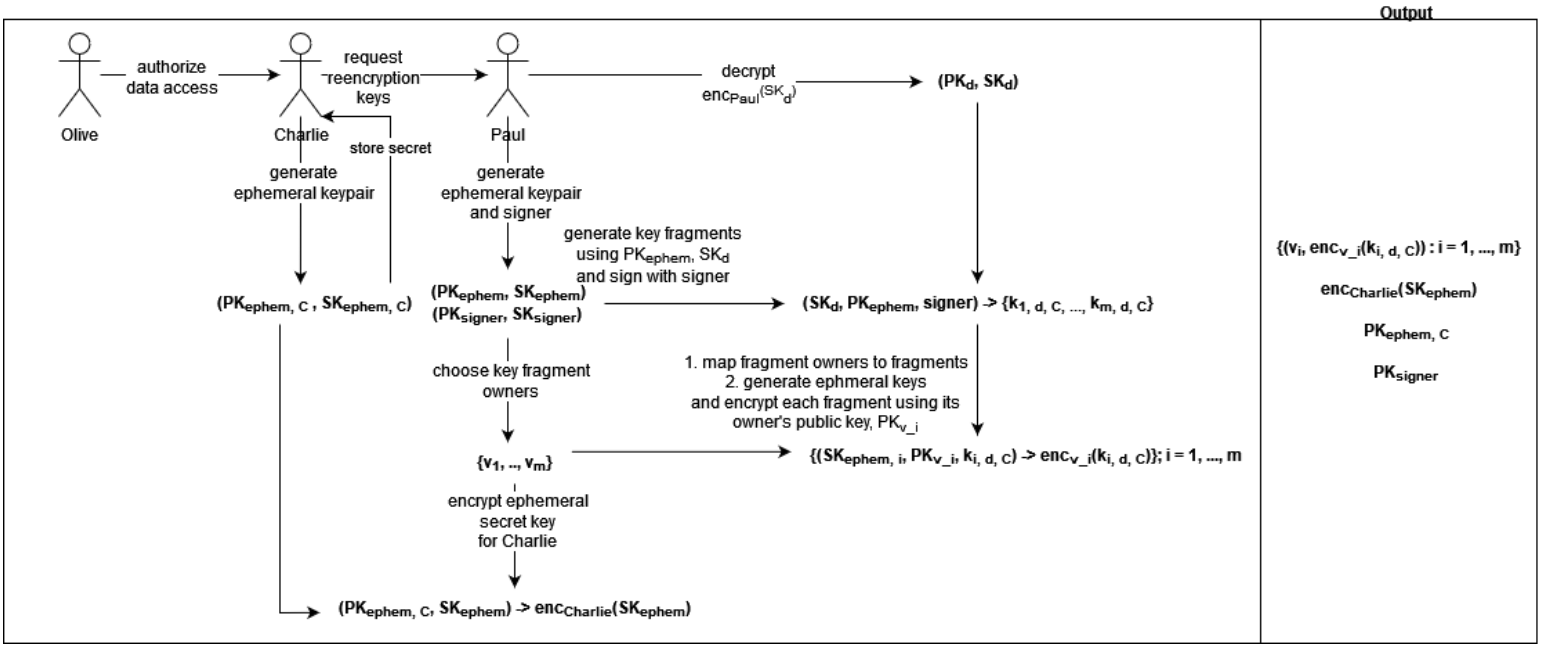

Fig. 2

To summarize, Paul has selected m nodes to hold encrypted key fragments. By generating key fragments using a public key that only Charlie knows, Paul has delegated decryption rights of the encrypted data to Charlie, without giving Charlie access to Olive's secret key or required reencryption of the data. We will see in the next phase how Charlie can decrypt the data.

Capsule Fragment Generation

In the next step, each key fragment 'owner' creates a capsule fragment using the key fragments from the previous phase.

Let $j \in [n, m]$ and choose j fragment holders $\{v_i: i \in [1, ..., j]\}$ from the participants chosen in the previous step. Each chosen fragment holder v_i then:

1. decrypts the encrypted key fragment $enc_{v_i}(k_{i, d, C})$ to recover $k_{i, d, C}$ using the ephemeral public key created by Paul, $PK_{ephem, i}$, and v_i's secret key SK_{v_i}.
2. verifies the key fragment is properly signed using the capsule key fragment $k_{i, d, C}$, the public key used when encrypting the data, PK_d, and the ephemeral public key that was created by Paul, $PK_{ephem, C}$, and the public key of the signer account, PK_{signer}, to create a verified key fragment $vk_{i, d, C}$.

3. creates a capsule fragment $C_{i,\,d,\,C}$ by reencrypting the key fragment using the verified key fragment v $vk_{i,\,d,\,C}$ and the capsule CAP_d.
4. encrypts the capsule fragment using Charlie's public key, $PK_{ephem,\,C}$, resulting in $enc_{Charlie}{}^{B}(c_{i,\,d,\,C})$.

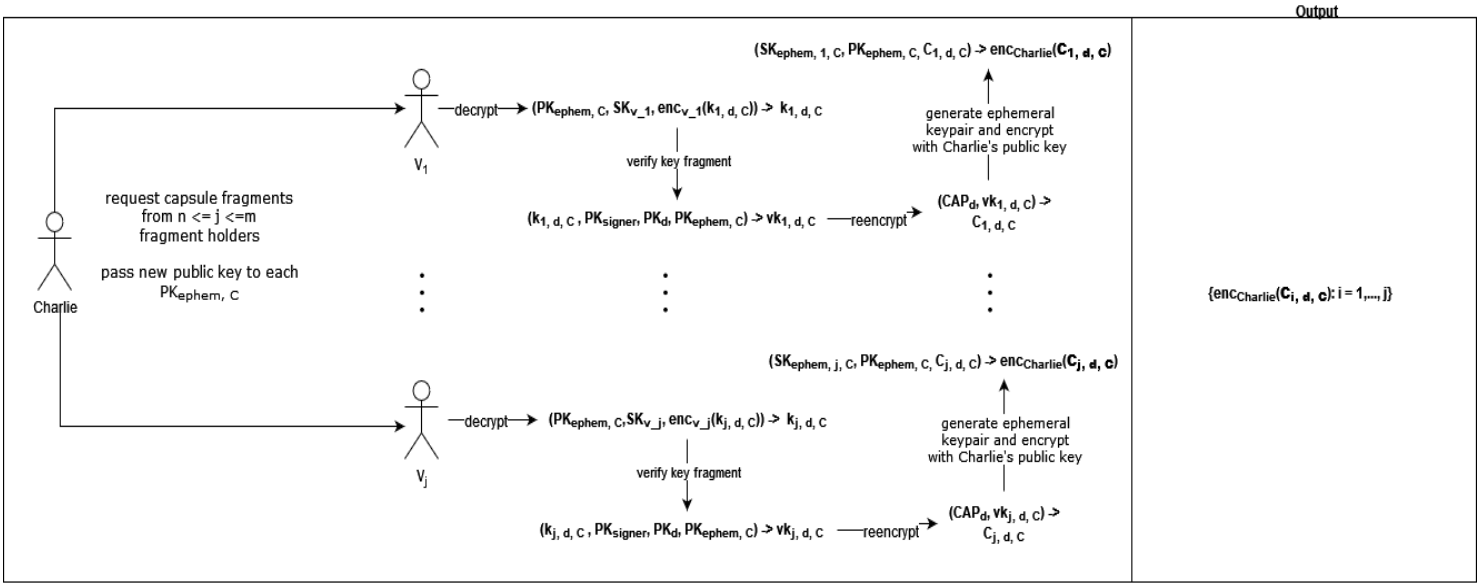

Fig. 3

Plaintext Recovery

1. Charlie decrypts each of the encrypted capsule fragments using $SK_{ephem,\,C}$, resulting in capsule fragments $\{C_{i,\,d,\,C}\}_{i=1}{}^{n}$.
2. Charlie uses the ciphertext and capsule, C_d and CAP_d, his own secret key $SK_{ephem,\,C}$, and the capsule fragments$\{C_{i,\,d,\,C}\}_{i=1}{}^{n}$, to recover the plaintext of the data d.

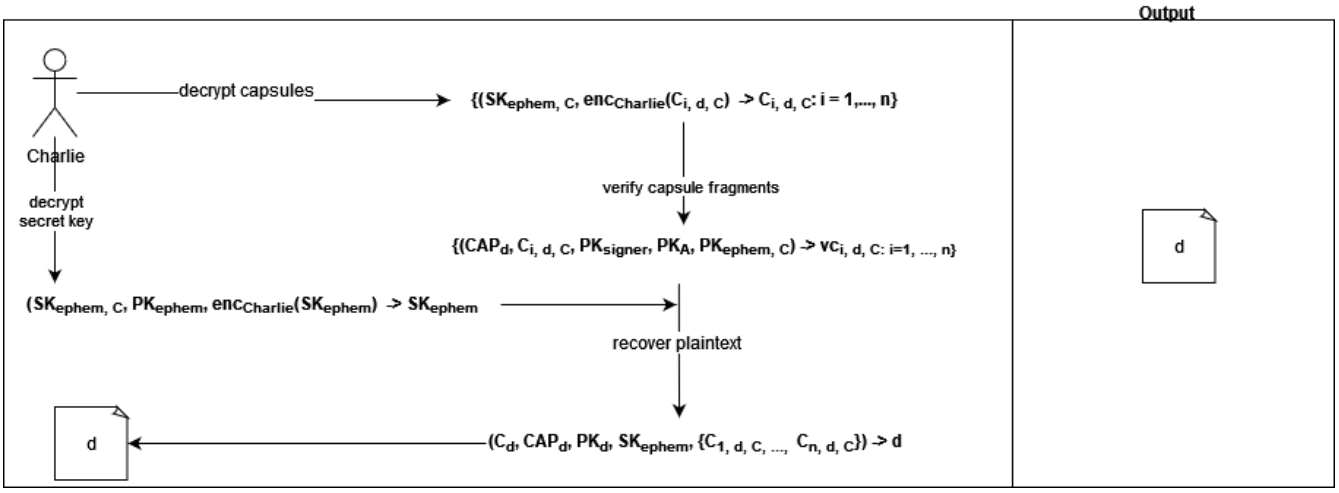

Fig. 4

4 Analysis and Blockchain Integration

In this section, we propose a potential integration of the protocol outlined in section 3 into a decentralized network, such as a blockchain. We will proceed to analyze the feasibility and security of such a system. To proceed, we will follow the same 4 major phases that are outlined in the protocol: encryption, key fragment generation, capsule fragment generation, and decryption.

The Network

First, we construct the network on which our solution will be proposed. We will use a blockchain where nodes are identified by unique keys generated on curve 25519. In our implementation, we leverage the Substrate framework to construct a node. As such, assume that each node exposes two RPC endpoints, **'encrypt'** and **'decrypt'**. Also assume that there is a subset of nodes, called **validator** nodes and denoted by V, that are responsible for finalizing blocks within the chosen consensus mechanism. At this time, we do not endorse or impose any specific form of consensus. In our construction, validator nodes will be responsible for *reencrypting key fragments*.

Encryption

The catalyst for initiating the protocol is when some participant, a 'data owner', passes some plaintext to the 'encrypt' RPC endpoint. We will assume that the caller has chosen a node to act as a proxy.. As in the protocol construction, let d represent the data.After execution of this RPC endpoint, the protocol will have produced the public key PK_d, the capsule fragment CAP_d, the ciphertext C_d, and the encrypted secret $enc_{proxy}(SK_d)$.

The output is put into runtime storage as:

$$PK_d \; -> \; (CAP_d, \; proxy)$$

$$PK_d \; -> \; proxy \; -> \; enc_{proxy}(SK_d)$$

Once encryption is successful, an *unsigned* transaction is submitted to encode these items within a block and the endpoint returns the ciphertext to the caller. Once the ciphertext has been made available through some offchain client, the data owner can then call into the blockchain's runtime to request that their ownership of the ciphertext is tracked (for example, by identifying the ciphertext's CID with a new asset class).

Key Fragment Generation

In this phase of the protocol, a proxy generates new key fragments and distributes them to validators. We will make an assumption about our network: we assume that it exposes some type of functionality for a node to authorize another to access data. In the most simple approach, assume that a data owner has the ability to authorize any other node to access their encrypted data. Then in this phase, assume that a data owner has authorized a 'data consumer' to access their data.

The 'data consumer' node generates a new keypair offchain and stores the secret key somewhere safe. In our runtime, the consumer uses the asset id to find the proxy, and requests reencryption keys from it, passing the newly created public key to it. The proxy generates new key fragments and randomly selects online validators to own a key fragment, where by 'own' we imply the fragment will be encrypted with the validator's public key (using ECDH). Thus, as output we get the following map, where id is the identity of the consumer node, C.

$$PK_d \; -> \; id \; -> \; (PK_{signer}, \; enc_{PK_{ephem,C}}(SK_{ephem,C}), \; \{(v_i, \; enc_{v_i}(k_{i,d,C})\}_{i=1}^{n})$$

Capsule Fragment Generation

The next step of the protocol requires that validators who hold a key fragment generate a capsule fragment for the consumer, C. Each validator generate the capsule fragment and encrypts it using the consumer's public key, encoding it in runtime storage as the map:

$$PK_d \; -> \; id \; -> \; \{enc_C(C_{i,d,C})\}_{i=1}^{n}$$

Decryption

Once sufficiently many capsule fragments have been encoded onchain, the data consumer can finally collect the minimum required number of them, decrypt them, recover the secret key, and finally decrypt the data. And we are done!

5 Conclusion

To summarize, the protocol we propose uses a multi-party system to enable a decentralized and non-interactive secret data exchange. We elaborated on the details of each phase of the protocol as well as discussed how the protocol may be integrated into a blockchain in order to build a decentralized data exchange network.

6 References

[1] https://math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf

[2] Adi Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.

https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf

[3] https://github.com/nucypher/umbral-doc/blob/master/umbral-doc.pdf