Yelling Secrets into a Crowd: Private Document Sharing on a Public Blockchain

Ajay Shridhar Joshi, Benita Kathleen Britto, Michael Yanagisawa
joshi28@wisc.edu, bbritto@wisc.edu, yanagisawa@wisc.edu

***Abstract*—Preserving privacy on a public blockchain is challenging – the principle of decentralization is fundamentally at odds with the desire for confidentiality. In this paper, we explore how a blockchain developer could securely share private documents on a public blockchain, a technique that could have real-world applications to fields such as digital communication and electronic medical records. We present different privacy-preserving techniques in blockchain, including current approaches and potential solutions. We also present a proof-of-concept document sharing tool based on a blockchain with built-in privacy and anonymity guarantees. We then assess our proof-of-concept, including assumptions and components necessary for scaling up the prototype. We then conclude with key takeaways and insights from the project, highlighting the difficulty of building privacy into a public blockchain.**

The implementation is available on Github.

I. Introduction

In order for the blockchain to be more widely adopted in other industries, the blockchain must provide some way of keeping a user's data private and confidential. However, this notion of privacy seems at odds with the public nature of the blockchain.

For example, one proposed use case for blockchains is to store electronic medical records. In this use case, is it possible to keep a patient's healthcare data completely private and confidential? Also, is it possible for the user to selectively share the blockchain data with certain users (e.g. different doctors)? Put another way: is there access control for the blockchain?

In all of these scenarios, we assume a case where the blockchain users do not trust a central authority to manage the data (hence, the need for a decentralized blockchain). The above use cases motivate a couple other desires in a blockchain: anonymity and control of the data stored on-chain.

In our project, we set out to examine how (and if!) privacy can be preserved on a public blockchain. This project is a mix of a review of privacy techniques in blockchain along with a small proof-of-concept document sharing tool based on what we have learned. This paper is broken down into a few sections:

1) **Background**: we first start out with some cryptography concepts that help motivate the privacy-preserving techniques we see later.
2) Current privacy-preserving approaches: what coins use privacy today, and what techniques do they use?
3) **Implementation**: we outline our proof-of-concept document sharing tool on a blockchain that has built-in privacy and anonymity guarantees. Because this is a prototype, we also include some of our assumptions, as well as some of the components that we envision would be needed to scale the prototype up.
4) **Conclusions**: we wrap up with future scope and our big takeaways from the project as well.

We had a great (and difficult) time learning about how hard it can be to build privacy into a public blockchain. We hope you enjoy and learn a thing or two, too.

II. Background

A. Preliminaries

Below enlists terminology that we use throughout the paper.

1) pk: Public key
2) sk: Secret key/ private key
3) pk_i: Public key of i^{th} person
4) sk_i: Secret key of i^{th} person
5) G: Elliptic curve generator
6) l: Elliptic curve constant
7) $pk = sk * G$
8) m: Message
9) rk: Re-encryption key

B. Trusted Execution Environments

As covered in class, TEEs are "hardware enclaves" where private information can only be decrypted by the TEE and the CPU.

C. Zero Knowledge Proofs

Zero-knowledge proofs provide a way to demonstrate the truth of a statement probabilistically, without revealing any information about the statement itself. This is achieved through a protocol where a prover can prove something to a verifier without divulging any information other than the fact that the statement is true. The concept of zero-knowledge proofs can be difficult to grasp; even Vitalik Buterin has acknowledged their complexity [1], [4], [11], [12].

A helpful analogy to understand the process is the case of two billiard balls, one red and one green, which appear identical to a color-blind friend. The prover can prove to the friend that the balls are distinguishable without revealing which is which. The friend holds a ball in each hand, and the prover watches as the friend swaps the balls or leaves them as is behind their back. The prover then has to guess whether the balls were swapped or not. By repeating this

process multiple times, the friend becomes convinced that the balls are differently colored, without learning which is which.

While zero-knowledge proofs typically require interaction, researchers have proposed non-interactive versions by using a shared key or string. In this case, randomness is the key ingredient, and the shared key takes the place of the requisite randomness of the interactive case. The workflow involves the prover passing secret information through a special algorithm to generate a zero-knowledge proof, which is then checked by the verifier.

While zero-knowledge proofs are not without their challenges, they have been proposed as a means to improve security and privacy in blockchain protocols. For those interested in a more in-depth look at the math behind zero-knowledge proofs, Vitalik Buterin's blog is a good resource. Other libraries, such as PySnark [5], are also helpful.

D. Diffie Hellman Key Exchange

The Diffie-Hellman key exchange is a cryptographic protocol used to establish a shared secret key between two parties over an insecure communication channel [17]. The protocol works by each party generating a private and public key pair. The private key is kept secret, while the public key can be freely shared. Using these keys, both parties can independently generate a shared secret key without ever transmitting it over the insecure communication channel. The strength of the protocol lies in the fact that even if an attacker intercepts the public keys during the exchange, they cannot derive the shared secret key without the private keys.

E. Homomorphic Encryption

Homomorphic encryption is a type of encryption that allows computation to be performed directly on encrypted data without the need to decrypt it first [9]. This means that computations can be carried out on sensitive data without revealing the actual data itself.

In traditional encryption schemes, data is encrypted and then decrypted before any computation can be performed. With homomorphic encryption, computations can be performed on the encrypted data, which remains encrypted throughout the computation. The result of the computation is also encrypted, and can be decrypted to obtain the output.

F. Threshold Proxy Re-encryption

In the implementation portion of this project, we use a type of homomorphic encryption called Umbral threshold proxy re-encryption to allow blockchain users to securely share documents with one another [9]. We'll slowly build our way up to what the full concept of threshold proxy re-encryption is.

1) Proxy Re-encryption: The core use case of proxy re-encryption is that we can delegate access to decrypt encrypted data from one user to another. For example, Alice encrypts a document such that she is the only person that can decrypt. If she wants Bob to have access, Alice can re-encrypt the encrypted document and share a re-encryption key with the proxy to allow Bob to decrypt the document. [13], [20], [24]

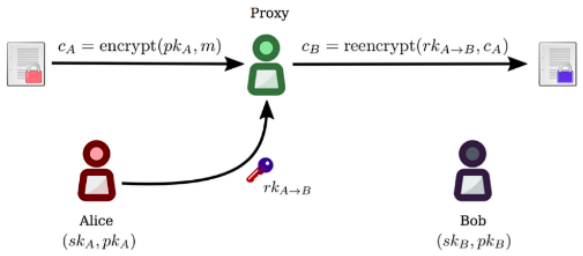

Fig. 1. Proxy Re-encryption scheme explained

How does a simple case of proxy re-encryption work? Alice has a message she wants to encrypt. She encodes the message m with her public key pk_A, resulting in a ciphertext:

$$c_A = \text{encrypt}(pk_A, m)$$

Note that Alice can still decrypt the message with her secret key sk_A. She then decides she wants to allow Bob to have access to read the message. To do so, she creates a re-encryption key $rk_{A->B}$ using:

$$rk_{A->B} = \text{rekey}(sk_A, pk_B)$$

She then re-transform the message:

$$c_B = \text{reencrypt}(rk_{A->B}, c_A)$$

Notice that the proxy does not need access to secret keys or the original plaintext message (which means that even if it wanted to, the proxy could not reveal the original message). Given c_B, Bob can decrypt the message with his secret key sk_B.

Overall, in order for Alice to delegate access to Bob, all Alice has to do is generate a re-encryption key between A and B (and leave the proxy to re-compute the ciphertext so that Bob can decrypt it).

2) Threshold Proxy Re-encryption: Threshold proxy re-encryption tackles the same problem as above: Alice wants to delegate decryption rights to Bob. However, in the "threshold" case, instead of one proxy server, there are multiple semi-trusted proxy servers. Because not all servers can be trusted, Alice breaks the re-encryption key into many pieces, giving each proxy server just one of the re-encryption key "fragments". Bob must receive a pre-defined threshold of key fragments in order to unlock the message. For example, if there are 20 proxies, Bob might only need 10 re-encryption key fragments to unlock the message (where 10 is the threshold set by the owner of the message). [18]

The process involves several steps, including encapsulation, encryption, re-encryption key fragment generation, re-encapsulation, decapsulate fragments, and decryption.

- **Encapsulation**: Alice's public key is used to encapsulate a symmetric key K. This key, K, is used to encrypt

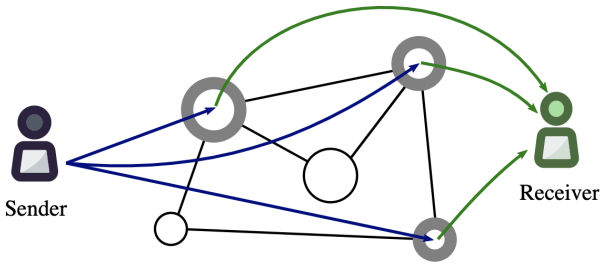

Fig. 2. Threshold Proxy Re-encryption

Alice's documents. Capsules are resulting objects that are an encapsulation of the symmetric key K.

- **Encryption**: Encapsulation is done within encryption:
  $(capsule, ciphertext) = \text{encrypt}(m, pkA)$
  $(capsule, ciphertext) = \text{encrypt}(m, \text{encapsulate}(pkA))$
  $(capsule, ciphertext) = \text{encrypt}(m, (K, capsule))$
  $(capsule, ciphertext) = \text{encrypt}(m, K)$

  Alice's public key is used to generate a symmetric key K which is used to encrypt the message m. The result of the encryption is a $capsule$ and a $ciphertext$. The capsule is an encapsulation of the symmetric key K while the $ciphertext$ is an encrypted message. The capsule can be re-transformed so that the message can be shared across users.
- **Re-Encryption key fragments**: Say, Alice wants to share m with Bob. She generates key fragments (kfrags) which is a result of Alice's secret key and Bob's public key divided into N fragments given to N proxies of which t fragments (threshold) are required by Bob to decrypt the message. Kfrags will be used to transform the capsule. Note, Alice does not require Bob's secret key to perform this operation. In addition, the proxy does not receive any secret key. This ensures the encrypted message is not compromised.

```
List[kfrags] = generate_kfrags(sk_A,
    ↪ pk_B, threshold=t, total=N)

for i, proxy in proxies:
   send_to_proxy(capsule, kfrags[i])
```

- **Re-Encapsulation**: The key fragments (kfrags) along with the capsule outputs capsule fragments (cfrags). The proxy performs this operation when it receives the kfrags and capsule from Alice. Recall that one proxy receives only one kfrag.
  `cfrag = reencrypt(capsule, kfrag)`
- **Decapsulate Fragments**: Bob contacts the proxy to receive t out of N cfrags. These cfrags along with Bob's private key will be used to decapsulate the key fragment to receive the symmetric key K.
- **Decryption**: Bob can decrypt the ciphertext with the symmetric key K.
- **Decapsule**: Alice is still able to retrieve the symmetric key K by using the capsule and her private key and decrypt the ciphertext that she had encrypted.

If you are hungry for a more thorough and formal write-up of how threshold proxy re-encryption works, you can find it in [19].

III. Current Privacy-preserving techniques

Over the years, Bitcoin has gotten a reputation of being an anonymous platform, but as Bitcoin admits, it is anything but. It is "probably the most transparent payment network in the world" [3]; all of the data is stored on a public blockchain.

What happens when you want to keep some of your transactions private (whether for nefarious reasons or not)? (For example, Richard Whitt argues that users have a "fundamental right to control access to their personal data" [27].) Bitcoin -- which claims it is built for *transparency* -– offers some rudimentary solutions: don't reuse Bitcoin addresses. Don't publish Bitcoin addresses on social media. Watch out for your IP addresses. In other words, they provide a human-centered approach to privacy.

Over the past decade, other blockchain developers have posited that there are better, more formal ways that we can handle privacy. We'll take a look at three different coins that have taken slightly different approaches to preserving privacy:

1) Monero ($2.8B market cap, 27th largest)
2) ZCash ($360M market cap, 119th largest)
3) Oasis ($340M market cap, 124th largest)

All data as of April 20, 2023 [6]–[8].

A. Monero

Monero is a cryptocurrency built with anonymity and privacy guarantees built into its foundation. It anonymizes the following pieces of each transaction:

- Sender: anonymized using ring signatures
- Receiver: anonymized using stealth addresses
- Amount: anonymized using Ring Signature Confidential Transactions (ringCT)
- Transaction broadcast (e.g. IP address): anonymized using Kovri

1) Anonymizing the Sender through Ring Signature: To authorize a transaction, a ring signature obscures the true sender by using a ring of public keys to sign the transaction. The list of public keys used to sign the message is available in the transaction, but it is not possible to determine with absolute certainty which public key corresponds to the actual sender [14].

To generate a ring signature, the sender of the transaction follows these steps:

- Assume that there are n slots in the ring, one for each public key
- Generate a random value α
- Choose the sender's position in the ring, say i
- Generate $(n-1)$ secret keys for the public keys in the ring, i.e., sk_j for all $j \neq i$ and j in the range $[1, n]$

- Generate a challenge for the sender: $c_{i+1} = Hash(List[pk], m, \alpha * G)$
- Generate challenges for the rest of the public keys, wrapping around from $c_{n+1} \rightarrow c_1$: $c_j + 1 = Hash(List[pk], m, sk_j * G + c_j * pk_j)$
- Ensure that alpha satisfies the equation to form the ring: $\alpha = (sk_i + c_i * pk_i)\%l$
- Form the signature as $\sigma(m) = [c_1, sk_1..sk_n, List[pk]]$.

To verify a ring signature,

- Start by calculating $c_2 = Hash(List[pk], m, sk_1 * G + c_1 * pk_1)$
- Next, calculate c_3 and repeat this process until reaching c_1 (i.e., $c_{n+1} \rightarrow c_1$).
- If the calculated c_1 matches the c_1 from the signature, the ring signature is valid

2) Anonymizing the Receiver through Stealth Address: The sender generates a one-time address for the receiver by combining the receiver's public address and the sender's random secret value [26].

Here are more details on the process:

- The sender has the receiver's public key, pk_r.
- The sender generates a random secret value, r.
- The sender obscures the receiver's public key by computing $Hash(r * pk_r)$.
- The sender also includes the public key to the shared secret in the transaction, $R = r * G$.
- When the receiver sees the transaction, it checks if the receiver field in the transaction matches with its private key by computing $Hash(R{*}sk_r) = Hash(r{*}G{*}sk_r) = Hash(r * (G * sk_r)) = Hash(r * pk_r)$.

3) Hiding the amount in the transaction using RingCT: This feature was included in Monero 2.0. In a standard Monero transaction (Monero 1.0), the transaction amount is visible on the blockchain, which can be problematic for privacy. In contrast, in Monero Ring Confidential Transactions (RingCT), the transaction amount is encrypted, so it's not visible on the blockchain.

To achieve this, the sender of a RingCT transaction generates a random number, called a "blinding factor," and multiplies it by the transaction amount. This result is then encrypted using a type of homomorphic encryption called a Pedersen commitment [14]. The receiver of the transaction can verify that the transaction is valid without knowing the actual transaction amount. To do this, the receiver checks that the Pedersen commitment is valid, meaning that the sum of the encrypted inputs is equal to the sum of the encrypted outputs. Recall that Monero is cryptocurrency and so has UTXOs. The receiver can then compute the difference between the encrypted input and output amounts, which reveals the net transaction amount.

This technique is similar to the commit and reveal technique we learnt in class. Both Commit and Reveal and Pedersen commitments are methods used to hide information while still enabling verification of correctness, but they differ in their underlying approach and application. Commit and Reveal is a two-phase process where the value is first committed and then revealed, while Pedersen commitments use a fixed value and a random number to commit to a value and enable verification of correctness.

4) Hiding the location of the node using the Kovri service: Monero anonymizes the location of the node (IP address) by using the Kovri service, which is similar to Tor [16].

B. ZCash

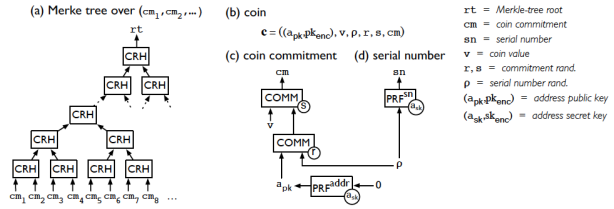

Fig. 3. ZCash coin commitment scheme

Zcash [23] is a cryptocurrency built on top of Bitcoin to provide anonymity for the sender, receiver, and amount in a transaction. Zcash uses commitments to hide coin information (owner, value of coin/note), which are organized as a Merkle tree for generating efficient proofs of membership. Zero-knowledge proofs are used in the form of zk-SNARKs to prove that the transaction is valid by showing that the owner has sufficient balance to pay the amount, and is not a double spend, among multiple other things.

Zero-knowledge proofs used here are non-interactive, i.e. don't need rounds of interaction between the prover and verifier for the proof, which helps the fact that any participant on the blockchain can verify the proof without any interaction with the owner/prover. Zcash requires a trusted setup phase for the parameters of the zk-SNARK, which if compromised can enable an attacker to generate 'fake' proofs which will be accepted by the verifier.

Although we could use zero-knowledge proofs for maintaining anonymity [15] [23] for document sharing, one of the main reasons we decided not to move forward with this approach is the complexity of zero-knowledge proofs(zk-SNARKs here), which we admittedly do not yet understand completely.

C. Oasis

Oasis responds to some of the implementation weaknesses of the above coins. While the Oasis Foundation agrees that the next generation of blockchain should focus on private computation, they see the need for a privacy approach that will scale: "These approaches [zero-knowledge proof systems and secure multiparty computation] have significant performance overhead and are only applicable to limited use cases with relatively simple computations." [22]

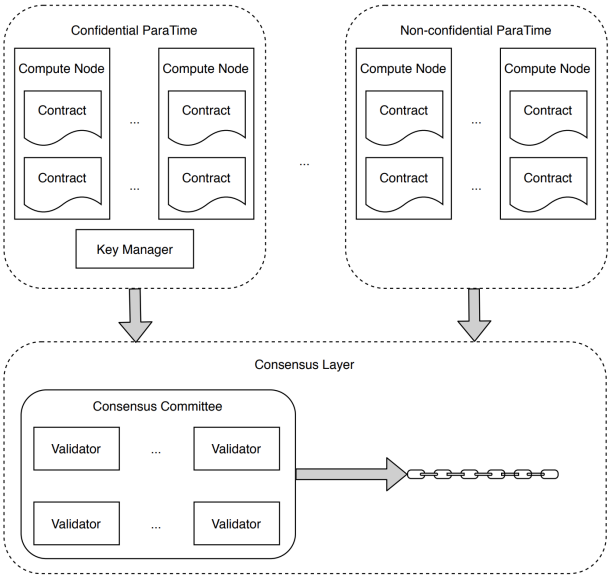

Fig. 4. Oasis Platform Architecture

Unlike the other coins we've looked at, though, most of what makes Oasis unique is not the privacy guarantees but the overall architecture. It's built to be modular, scalable, and fast. Computation (e.g. for smart contracts) is done in parallel runtimes (called "ParaTimes"), which is completely separate from the consensus layer, allowing the consensus layer to be as simple as possible. The architecture is modular: the consensus and execution layers are completely separate, and new ParaTimes can be added using whatever techniques suit it best. (For example, ParaTimes may implement any variety of confidential computing techniques – multiparty computation, fully homomorphic encryption, zero-knowledge proofs, etc. – allowing the developer to weigh the trade-offs of each technology [22] [21].)

In practice, Oasis releases a few pre-built ParaTimes types. In most cases, confidential computation is accomplished primarily with TEEs. For example, the Cipher ParaTime guarantees confidentiality with three mechanisms [10]: The contract is executed in a trusted execution environment, The contract's storage on the blockchain is encrypted, and The client's transactions and queries are end-to-end encrypted

Overall, our major takeaways from how Oasis implements privacy are:

- self-contained confidential compute nodes and
- a ParaTime architecture that is agnostic to the privacy-preserving mechanism (but that uses TEEs in released ParaTimes)

D. Other text-based privacy blockchain proposals

One healthcare blockchain proposed using zero-knowledge proofs and proxy re-encryption to store electronic medical records for India's national healthcare scheme [25].

IV. Design and Implementation

Our goal was to build a prototype of a simple blockchain that allows users to share documents on-chain privately. We liked the simplicity of the consortium blockchain from Project 2b, so we sought to build privacy-preserving mechanisms on top of that project.

The components we wanted to implement were:

- Proxy re-encryption: In our use case, this allows a user to share a document with any other users in the blockchain.
- Anonymization of the sender/receiver: To add a layer of anonymity over the document sharing, we want to make it so that an attacker perusing the blockchain does not know who sent and received the message. Our plan was to use Monero's anonymization techniques as a source of inspiration for our project.

Following enlists our requirements:

- A user can "upload" a document to the blockchain that no other user can read
- The user can then share the document to any other user, allowing only that other user to read the document

To do so, we implement:

- Proxy re-encryption for multiparty document sharing
- Hide the sender address using ring signatures
- Hide the receiver address using stealth addresses
- Hide linkage between transaction using symmetric key encryption

A. Design decisions and assumptions

Why did we go with Monero anonymization over ZCash and Oasis?
Monero building blocks are simpler and easier to understand than building blocks for Zcash. Oasis requires the use of trusted hardware(TEEs), which we don't assume, so using approaches from Monero made sense.

Why did we choose a push based share mechanism?
Our approach to sharing is based on pushing rather than pulling. With a push mechanism, the sender takes the initiative to start the sharing process by sharing a document. This is comparable to the "Share" feature in Google Docs, which allows the owner of a document to grant access to specific individuals. In contrast, a pull mechanism would require the recipient to request access from the document owner and wait for approval. This would require the document owner to be responsive, which we aim to avoid.

B. Our prototype

Our approach to revealing the prototype design is to do it in stages. Firstly, we outline a basic approach, followed by the incorporation of proxy re-encryption into Project p2b to enhance the initial approach. We then proceed to add anonymity to the prototype and detail two possible methods to achieve this. We describe the process required for Alice to store documents on the blockchain and share a document with Bob.

1) Design 1: Naive Approach: Simple key-exchange-based sharing: This design facilitates document sharing on a public blockchain without providing anonymity. The process involves generating a symmetric key between the sender and receiver using Diffie-Hellman. The document owner/sender then encrypts the document with this symmetric key and sends it to the receiver as a transaction. The receiver uses the symmetric key extracted from the transaction to decrypt the document.

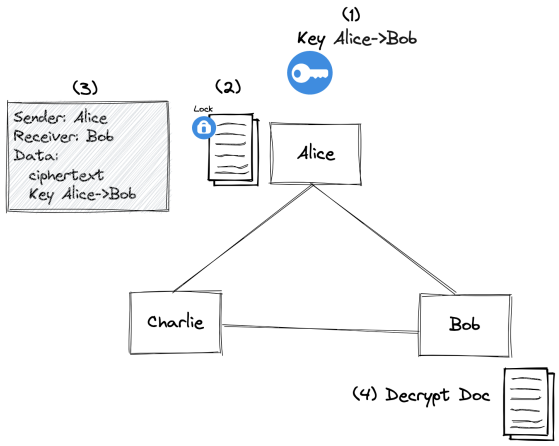

Fig. 5. Design for naive file sharing system

The workflow is as follows:

- Alice, the document owner, generates the symmetric key using Diffie Hellman.
- Alice encrypts the document with the symmetric key.
- Alice creates a transaction of the encrypted document and the symmetric key.
- Bob, the intended recipient, decrypts the document using the symmetric key. Charlie or any other unauthorized party cannot decrypt the document since the symmetric key ensures that only Bob has access to the decrypted content.

The strengths of this design:

- Provides a simple way to share private information on a public blockchain while keeping the information entirely on the chain

The weaknesses of this design:

- If Alice needs to share the document with another person, she must generate another symmetric key, encrypt the document again, and share it with the new recipient.
- This technique leads to redundancy in document sharing as the document needs to be encrypted multiple times, once for each recipient.

2) Design 2: Using Proxy Re-encryption: To eliminate the need for encrypting the document multiple times, we adopt a technique called proxy re-encryption, specifically using a threshold-based proxy re-encryption approach (Section II-F). In our implementation, we use a single proxy for simplicity, but it can easily be extended to use multiple proxies. We use pyUmbral, which is a python implementation of the threshold-based proxy re-encryption [12]. Note that this implementation provides privacy of the document but not anonymity.

The document sharing process is divided into two stages:

- Upload the document encrypted with the encapsulation of the owner's public key, resulting in a capsule and ciphertext that are added to the blockchain. (Section II-F)
- Share the document with a receiver (re-encryption): The owner generates a key fragment using their secret key and the receiver's re-encryption public key. The key is then split into a configurable number of fragments, and each fragment is sent to a proxy. A threshold is set on the number of fragments the receiver needs to collect from the proxies to decrypt the document. The sender also adds a reference to the upload transaction so the receiver can use the capsule and ciphertext from that transaction, along with the key fragments collected from the proxy, to decrypt the document. This mechanism results in encrypting the document only once.

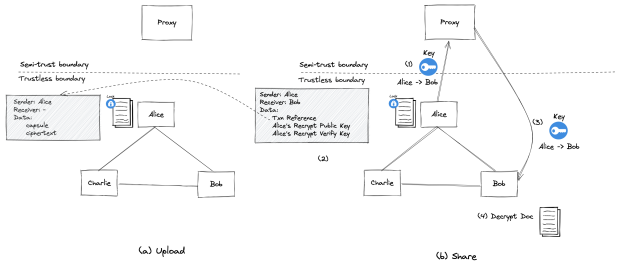

Fig. 6. Proxy re-encryption workflow

The workflow is as follows for the upload stage:

- Alice, the owner, uploads the document using her public key, resulting in a capsule and ciphertext added to the transaction and the blockchain. This is similar to the Encryption process under Section II-F.

The workflows is as follows for the share stage:

- Alice decides to share the document with Bob. She generates a single key fragment using her private key and Bob's public key and sends it to the proxy (Section II-F). The proxy is responsible for re-encrypting the capsule with the public keys of Alice and Bob.
- Alice then adds her re-encryption public key and verify key to the transaction, along with a reference to the transaction holding the capsule and ciphertext of the encrypted document. The public key will be used by Bob to retrieve the re-encryption key from the proxy while the verify key is used by Bob to check if the re-encryption key received from the proxy is valid.
- Bob extracts the capsule and ciphertext from the referenced transaction and contacts the proxy to receive the re-encryption key to decrypt the document. To contact the proxy, Bob will extract the sender's public key from the transaction and send a message to the proxy containing Alice's public key and his public key. The proxy

will respond with the re-encapsulated capsule fragments aka cfrags (Section II-F). When Bob receives the re-encryption key from the proxy, he verifies if the key is valid by using Alice's verify key, thus maintaining the semi-trusted property of the proxy. Bob uses his private key and the re-encryption key to decrypt the document.

The strengths of this design are:

- Support for sharing documents with more than one person
- Document is encrypted only once with the owner's public key

The weaknesses of this design are:

- Use of a semi-trusted proxy
- The re-encryption capability is offloaded to the proxy, and the proxy holds the re-encryption key, which is no longer part of the chain

3) Design 3: Our Approach for Anonymous Document Sharing: The previous design revealed sensitive details such as the sender, receiver, and transaction fields, which compromised user anonymity. To address this, we apply concepts from the Monero paper, such as ring signatures, stealth addresses, and symmetric key encryption in the share stage.

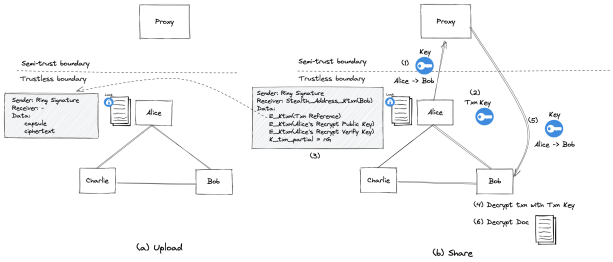

Fig. 7. Anonymous document sharing workflow

The process starts with the upload stage which consists the follows:

- Alice uploads the encrypted document using her public key, similar to Design 2. However, to hide her identity, she signs the transaction using a ring signature (Section III-A-1), which includes her public key among others, making it difficult to identify the actual sender. We use the pyring [2] implementation of ring signatures in our prototype.

Next, the share stage:

- Alice generates a key fragment (same as Design 2) to share with the proxy
- Alice also generates a transaction key using a random secret value (r). She shares the public key associated with this random secret value ($r * G$) in the transaction, which is used to generate a stealth address for Bob (Section III-A-2). Alice then encrypts the transaction reference, her public key, and verify key using the secret value r and Bob's public key. This obscures all the contents of the share transaction.
- When Bob receives the transaction, he checks if the stealth address matches his address (Section III-A-2)
- If it does, he decrypts the data fields of the transaction using the shared secret key $r * G$ and his private key. He then obtains the reference transaction, Alice's public key, and verify key.
- Finally, Bob requests the re-encryption key from the proxy, and the document decryption process is the same as in Design 2 (i.e. Steps 5 and 6 in the diagram are the same as Design 2 step 3 and 4).

The strengths are:

- This design ensures user anonymity by hiding sensitive information, thereby eliminating metadata leakage.

The weaknesses are:

- However, it still shares the same weaknesses as Design 2.

4) Design 4: Alternative approach by eliminating the proxy: This design presents an alternative to Design 3 by eliminating the need for a semi-trusted proxy and bringing all components on-chain. However, we have decided not to implement this approach due to its weaknesses. Despite this, we were interested in developing a design that would not depend on a semi-trusted party.

To protect user identity and ensure the security of their documents, both sender and receiver are required to use a distinct private/public key for each uploaded or received document. This practice is necessary because if the same address were reused, it would compromise the user's anonymity and also prevent the secure addition of the recryption key to the blockchain. Failure to use new keys for each document would result in the receiver gaining access to all of the owner's documents.

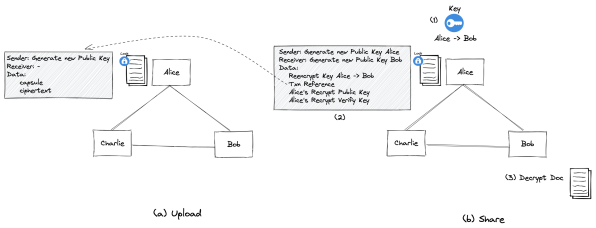

Fig. 8. Alternative anonymous document sharing workflow

The upload process is as follows:

- Alice uploads the document to the chain after encrypting it with her public key (same as Design 2)

The share process is as follows:

- Alice generates a re-encryption key for Bob
- Alice puts the re-encryption key in the share transaction (instead of sharing with proxy from Design 2) along with the transaction reference (same as Design 2)
- When Bob receives the share transaction, he gets the encrypted document using the transaction reference and uses the re-encryption key to re-encrypt the document.

- Bob then proceeds to decrypt the document with his own private key
- Note, this approach does not require a transaction key as seen in Design 3

The strengths of this design are:

- This model operates entirely on-chain, eliminating the need for semi-trusted proxies.
- By utilizing unique private/public keys for each document transaction, it ensures the anonymity of both the sender and the receiver. The per-document keys are never linked together, preventing any correlation between different transactions.

The weaknesses of this design are:

- Per-document keys generates a lot of keys to maintain carefully without leaking

C. Implementation Challenges

1) Understanding Cryptographic constructions: Understanding the cryptographic constructs used in our prototype was a challenging task. We had to delve into the details of various concepts, including stealth addresses, ring signatures, and proxy re-encryption, to build a secure and efficient system. These constructs are complex and require a deep understanding of mathematical concepts and their practical implementations. We spent a significant amount of time researching and learning these concepts to develop a comprehensive understanding of how they work and how they can be applied in our prototype. Despite the challenges, we were able to successfully integrate these constructs into our system, which has greatly enhanced the privacy and security of the network. Overall, the process of understanding and implementing these cryptographic constructs was a valuable learning experience.

2) Interoperability of cryptographic libraries: In our implementation, we rely on two libraries for performing cryptographic operations: pyUmbral [18] for proxy re-encryption and encryption of data fields, and pyring [2] to support ring signatures and stealth addresses. To use these libraries, we need separate key pairs. Therefore, we require a public-private key address for pyring and a re-encryption public-private key pair for pyUmbral. However, it would be more convenient if we could use a single key pair for all cryptographic operations.

3) Unknown unknowns: One of the biggest challenges that we faced while working to build a private and confidential blockchain is that none of us are cryptography nor blockchain experts – which makes it hard for us to know with confidence how easily a given scheme could be attacked. We readily admit that there are a lot of "unknown unknowns" in designing and implementing our project.

D. Existing weaknesses and potential solutions

1) Existence of proxy: In the current implementation (Section III-B-3), we use a proxy, which is commonly used by many other implementations of proxy re-encryption.

One idea we discussed doing (in theory, at least) was whether it was possible to remove the proxy and keep all the data (e.g. re-encryption keys) on-chain. One potential solution we explored was having one public key per document (Section III-B-4).

One benefit of this scheme would be that all the data (ciphertexts, re-encryption keys) would all be on-chain. However, the downsides would be that each user would have to maintain a wallet of document-to-key mapping. Nodes also wouldn't be able to offload ciphertext re-encryption to a proxy, meaning more compute for each individual node.

We're also worried that we're missing some very obvious attack to using one public key per document (in lieu of having the proxy server). We think that this is the mostly likely scenario.

2) Validating a "transaction": Currently, there is no good way to verify if a single transaction is "correct". If a user uploads a "bad" document, there's no way to validate that the transaction is invalid. Therefore, the blockchain will simply append an "invalid" transaction to the blockchain.

In the worst case, a malicious user could try to upload a lot of very small invalid documents. Because there is no good mechanism to validate each transaction, all of the transactions would be added to the blockchain.

In order to prevent this type of DDoS attack, we propose setting the gas price appropriately in order to make a DDoS attack sufficiently costly. For example, every transaction could have a fixed transaction cost along with a variable cost that depends on the document size.

3) Other (small) information leaks: Currently, we simply encrypt the document, meaning that the document size and encrypted document size are correlated. Revealing the size of the document may incidentally reveal information, too. (For example, if you knew that the document-sharing scheme was used to encrypt "yes" and "no", you could tell the two apart.)

V. Future Scope

A. Limiting document access and revoking access

Ideally, we'd want some way for a user to be able to revoke access to a document. For example:

- A document may be shared with another user accidentally
- The user may only want to share a document with a user for a limited period of time (e.g. just for one week)

Given that all of the information necessary to read the document is on the blockchain, we haven't been able to think of a good way to revoke access to a given document. The user who accidentally received the document would have everything they need to decrypt the document; there's no way (that we know of) to prevent the user from reading transaction history from the blockchain.

In order to deal with limited-time sharing, we think that some sort of expiration date could be encrypted into the transaction. One thing we're not sure about is whether there is a blockchain consensus of the current time (or if some other mechanism, like an oracle, might need to be used).

B. Updating a document

Likewise, updating a document might be difficult with the current iteration; there's no good way to say that an upload transaction is an update to a previously uploaded document.

One (lightly explored) proposal is to upload a document pointer to the blockchain (instead of the document contents itself) and use an oracle to access the actual contents of the document. While this approach would limit the amount of information that needs to be appended to the blockchain (and would make updates simpler on the chain), we would effectively offload the full document revision history to the oracle (which would make our blockchain a little less decentralized).

More research could be done to understand what other approaches could be explored to update documents on-chain in a scalable way.

VI. Conclusion

We have successfully created a document sharing prototype on project p2b that guarantees data privacy and anonymity. Our work on this project has exposed us to various cryptographic constructs such as proxy re-encryption for maintaining document privacy, ring signatures for obscuring the sender of a transaction, stealth addresses for obscuring the recipient, and Diffie Hellman key exchange based encryption for hiding the linkage between transactions. Although we acknowledge that there may be potential security loopholes in our design, getting to this point has been a challenging but insightful journey. We have learned that adding privacy to a public blockchain can be difficult and that integrating an application like document sharing into a blockchain may not meet the performance requirements of a large-scale file server. Nonetheless, our approach ensures that we can share documents in a decentralized manner in an almost trustless network. However, as we rely on semi-trusted proxies to hold re-encryption keys, we are not entirely operating in a trustless setting.

References

[1] 2010. Example of a good Zero Knowledge Proof. (2010). https://mathoverflow.net/questions/22624/example-of-a-good-zero-knowledge-proof

[2] bartvm. 2023. bartvm/pyring: Ring signature implementations in Python. (2023). https://github.com/bartvm/pyring

[3] Bitcoin. 2023. Protect your privacy - Bitcoin. (2023). https://bitcoin.org/en/protect-your-privacy

[4] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and Its Applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*. Association for Computing Machinery, New York, NY, USA, 103–112. DOI: http://dx.doi.org/10.1145/62212.62222

[5] Vitalik Buterin. An approximate introduction to how zk-SNARKs are possible. (????).

[6] CoinGecko. 2023a. Monero Price: XMR Live Price Chart News. (2023). https://www.coingecko.com/en/coins/monero

[7] CoinGecko. 2023b. Oasis Network Price: ROSE Live Price Chart News. (2023). https://www.coingecko.com/en/coins/oasis-network

[8] CoinGecko. 2023c. Zcash Price: ZEC Live Price Chart News. (2023). https://www.coingecko.com/en/coins/zcash

[9] Cem Dilmegani. 2023. What is Homomorphic Encryption? Benefits Challenges (2023). (2023). https://research.aimultiple.com/homomorphic-encryption/

[10] Oasis Network Documentation. 2023. Confidential Hello World in Cipher ParaTime. (2023). https://docs.oasis.io/dapp/cipher/confidential-smart-contract/

[11] Ethereum. 2023. Zero-knowledge proofs. (2023). https://ethereum.org/en/zero-knowledge-proofs/

[12] S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*. Association for Computing Machinery, New York, NY, USA, 291–304. DOI: http://dx.doi.org/10.1145/22145.22178

[13] Susan Hohenberger. 2007. Lecture 17: Re-encryption (Special Topics in Theoretical Cryptography). (2007). https://www.cs.jhu.edu/~susan/600.641/scribes/lecture17.pdf

[14] Koe1, Kurt Alonso, and Sarang Noether. 2020. Zero to Monero: Second Edition. (2020). https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf

[15] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *2013 IEEE Symposium on Security and Privacy* (2013), 397–411.

[16] Monero. 2023. Kovri (on Moneropedia). (2023). https://web.getmonero.org/resources/moneropedia/kovri.html

[17] Svetlin Nakov. 2023. Diffie–Hellman Key Exchange. (2023). https://cryptobook.nakov.com/key-exchange/diffie-hellman-key-exchange

[18] nucypher. 2023. nucypher/pyUmbral: NuCypher's reference implementation of Umbral (threshold proxy re-encryption) using OpenSSL and Cryptography.io. (2023). https://github.com/nucypher/pyUmbral

[19] nycyher. 2018. UMBRAL: A Threshold Proxy Re-encryption Scheme. (2018). https://github.com/nucypher/umbral-doc/blob/master/umbral-doc.pdf

[20] Blockchain Partner. 2019. Decentralized identity : granting privacy with proxy re-encryption. (2019). https://medium.com/@teamtech/decentralized-identity-granting-privacy-with-proxy-re-encryption-e0bf68ad465c

[21] Oasis Protocol Project. 2020. The Oasis Blockchain Platform. (2020).

[22] Oasis Protocol Project. 2023. An Implementation of Ekiden on the Oasis Network. (2023).

[23] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*. IEEE, 459–474.

[24] Sculptex. 2023. Sharing Secrets Securely – Proxy ReEncryption. (2023). https://medium.com/@sculptex/sharing-secrets-securely-proxy-re-encryption-58b2a315e086

[25] Bhavye Sharma, Raju Halder, and Jawar Singh. 2020. Blockchain-based Interoperable Healthcare using Zero-Knowledge Proofs and Proxy Re-Encryption. In *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*. 1–6. DOI: http://dx.doi.org/10.1109/COMSNETS48256.2020.9027413

[26] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. (2013). https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf

[27] Richard S. Whitt. 2020. Digital Stewardship: An Introductory White Paper. (2020). https://ssrn.com/abstract=3669911