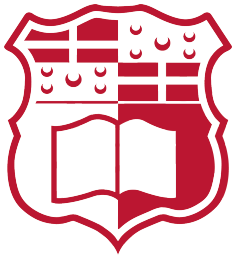

# Privacy-Preserving Solutions for Decentralized Permissionless Blockchains


**Alexander Zammit**

Supervised by Prof. Joshua Ellul

Co-supervised by Prof. Gordon Pace

Centre for Distributed Ledger Technologies

Information and Communication Technology

University of Malta


**September, 2022**

*A dissertation submitted in partial fulfilment of the requirements for the degree of M.Sc. in Blockchain and DLTs.*

L-Università ta' Malta



WWW.UM.EDU.MT

*First edition, Monday 12th September, 2022*

*To Stephanie and Nathan*

*who inspire and encourage me to ask for more from myself*

# Acknowledgements

This dissertation involved a long learning journey. The input of my supervisor Prof. Joshua Ellul and co-supervisor Prof. Gordon Pace was invaluable. They patiently guided me to significantly improve my work. The lessons learned will certainly continue to influence any future research I will be engaged in.

Credit also goes to all those involved in this multi-disciplinary Master's course, the administration, lecturers, and fellow students. All of them contributed to make this journey worthwhile. Knowing them has certainly helped mature my strong conviction that blockchains and DLT technologies are a huge opportunity, that is up to us to harness.

# Abstract

Permissioned blockchains have established themselves as the solution of choice in applications requiring some form of decentralized trust and privacy preservation. The old school of centralized solutions made permissioned access a natural choice in safeguarding privacy, and the same design models have been brought over to the blockchain context.

However, blockchains are a major disruptive innovation, requiring rethinking such solutions. Privacy technologies running on top of decentralized permissionless blockchains are attracting significant research and development. This dissertation investigates the key challenges, in implementing privacy preservation in this context.

This theme is explored by tackling a specific problem, for which a design based on permissioned blockchains is already available. The selected case study enables users to provide and withdraw data usage consent. The need for such functionality arises from the General Data Protection Regulation and has broad applicability. The dissertation redesigns the solution to satisfy the original case study requirements. However, it does so for the decentralized permissionless context by applying tokenized privacy and Zero Knowledge Proofs.

The new design and its partial implementation enabled investigating whether such solutions can be effectively implemented on a permissionless blockchain. Specifically, a measure of the resource levels required by ZKP-based applications, in terms of storage, processing time and transaction fees is presented.

Through this use-case it was demonstrated that mixers provide a good starting point for a wide range of solutions. Results showed the correlation between ZKP complexity, proving key sizes and proof generation times. Comparisons between PGHR13 and Groth16 showed how the ZKP scheme choice impacts applications and why Groth16 is considered an efficiency benchmark.

PGHR13 produced, proving keys up to 50% larger, a verification key 97% larger, and a verification cost rise of 220%. However, the two schemes only registered marginal differences in proof generation times.

The results for on-chain verification costs were especially interesting. ZKP verification was estimated to only consume 18% of the total cost. Groth16 ZKP verification on Ethereum for November 2021 was estimated to cost €104, whereas the total smart contract cost averaged at €590.

Finally, this exercise identified practical challenges that privacy-preserving solutions must solve for them to move to the permissionless context. These include, the challenges arising from representing real world assets using private tokens, financial feasibility, application security, the need for a holistic approach to privacy, the challenges of dealing with spam and malicious transactions, and the application of various decentralization vectors.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

# 1

# Introduction

Privacy is one of the most basic ingredients enabling us to engage socially and carry out business. Whether chatting with friends, shopping online, or doing business, privacy is a fundamental enabler. Conversely, its absence limits the type of interactions we are willing to engage in.

This is a fundamental challenge for public blockchains that allow anyone to read and verify transactions. In blockchains providing a smart contract platform, the data may include a broad range of both personal and business critical data. Thus, lack of privacy also constitutes a blockchain adoption hurdle.

This challenge has been evident for years, starting from the application of graph analysis to Bitcoin transactions (Ron and Shamir, 2013). This spawned the research and development of solutions in all directions. It included both refinements in transaction deanonymization, but also the development of better privacy preserving solutions.

## 1.1 | Motivation

Decentralized blockchains have been promising wide-spread disruption for years. Among others, decentralization promised user empowerment in direct democracy, health data handling and loyalty schemes. Yet many applications are opting for centralized permissioned blockchains. Instead Decentralized Finance (DeFi) and applications of questionable value continue to dominate the decentralized space.

In his November 2019 lecture, at the University of Malta, EY's Paul Brody explained how lack of privacy was blocking enterprise blockchain adoption. Their research showed that the future of blockchains is public. However, no enterprise is willing to put their data on a blockchain that gives unhindered access to competitors.

At the same time, the General Data Protection Regulation (GDPR) (European Commission, 2016) imposes strict requirements around user data handling. Regulatory compliance pushes service providers to retain data access control, making the permissioned setting more attractive than the permissionless alternative.

Looking at projects that identified permissioned blockchains as their preferred setting, allows studying why the decentralized alternative was shunned. Through this approach, this dissertation aims to contribute towards overcoming the challenges hindering decentralized blockchain adoption for privacy preserving solutions.

## 1.2 | Research Question

One requirement that arises from the GDPR is the need for educated consent. Broadly, this application class can be described as involving centralized service providers that must allow users to grant and withdraw data handling consent. This theme was chosen because of its general applicability.

The research focuses on the feasibility of such consent applications when implemented on a decentralized public blockchain in a privacy preserving manner. Specifically:

1. What resource levels do such applications require, in terms of storage, processing time and transaction fees?

   To investigate this question, a permissioned blockchain consent solution proposed in literature, namely Dwarna (Mamo et al., 2019), will be redesigned for a public blockchain. Focus will be put into understanding how such a solution, would differ in a permissionless environment. Thus, covering an additional research question:

   1.5 What would Dwarna look like in the permissionless context?

   Tackling this case study ensures that the dissertation is dealing with a problem for which a clear real-world need has already been established.

2. What kind of common challenges do such applications need to deal with?

## 1.3 | Overview of Work

Work stated with a review of privacy attack vectors and surveys covering privacy preserving technologies applicable to permissionless blockchains. This helped identifying tokenized privacy and Zero Knowledge Proofs (ZKPs) as the technologies of choice.

Looking at various ZKP projects, it was established that most relied on Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) schemes. Background topics, including finite field theory, elliptic curve cryptography and pairings, were covered. Next, a practical understanding of zkSNARK construction was attained.

Consulting with the tutors, Dwarna was identified to satisfy the need for a real-world problem with broad applicability. The core application requirements were identified, keeping an open mind for potential improvements.

Looking at ZKP support in permissionless blockchains, the focus was restricted to Ethereum as this provided three relatively mature tokenized privacy projects that could serve as an implementation starting point. Following a review of Zeth (Rondelet and Zajac, 2019), Nightfall (Konda et al., 2019) and the Aztec Protocol (Williamson, 2018), Zeth was chosen. This decision was greatly influenced by its high-quality protocol specifications document [1]. Next the focus shifted on obtaining an intimate understanding of Zeth, its open-source code, and the libsnark library [2].

The redesign of Dwarna started with an analysis of various decentralization vectors. In tandem, the privacy expectations of all participants were identified and mapped to privacy design patterns. Thus, the design was shaped.

A development plan was traced to help set implementation goals achievable within the available time. In consultancy with the tutors, development was restricted to the zkSNARK implementations. Five ZKPs of differing complexity were developed. To these five, testing also added Zeth's ZKP, which provided a benchmark. From here, the storage requirements, processing time, and transaction fees were measured.

Lastly, a list of challenges identified during the design process, having general relevance, was compiled. These were documented to provide a practical insight and a reference for other similar projects.

[1] Zeth Protocol Specification [Accessed Sep 2021] - `https://tinyurl.com/et8rdtd6`
[2] libsnark C++ library [Accessed Jul 2021] – `https://github.com/scipr-lab/libsnark`

## 1.4 | Aims and Objectives

This dissertation aims to contribute towards overcoming the challenges hindering the adoption of permissionless blockchains for privacy preserving solutions. Many projects deal with an ideal scenario where all assets can be fully managed on-chain. This dissertation is rather interested in real-world problems involving a mix of on-chain and off-chain elements, posing a bigger privacy challenge. To pursue these aims the following objectives were set:

1. Tackle a problem for which a real-world need has already been established, providing a detailed design for the permissionless blockchain setting.
2. Implement the core design elements.
3. Test the solution to obtain key application metrics, including storage requirements, processing time, and running costs.
4. Produce a list of design challenges having general relevance. Document these to provide a practical insight and a reference for other projects tackling privacy in the decentralized permissionless setting.

## 1.5 | Document Structure

The chapters that follow present the dissertation work incrementally. Chapter 2 discusses various background topics. It covers privacy attack vectors, privacy preserving technologies, Dwarna's dynamic consent solution, permissioned and permissionless blockchains, and blockchain solution layering. Chapter 3 describes an alternative design to Dwarna's dynamic consent solution. Chapter 4 discusses key implementation details. Chapter 5 presents application metrics obtained by testing the consent solution and Zeth. The same chapter also includes the list of design challenges relevant to similar projects. Chapter 6 discusses related work including alternative implementation approaches and a comparison against Dwarna's consent solution. Finally chapter 7 concludes with a result summary, a critique of the proposed consent solution and a discussion of future work.

# 2

# Background

This chapter presents the dissertation founding research. The research started with a focus on common deanonymization vectors, challenging smart contract privacy. Next, the prevalent blockchain privacy preserving technologies were identified.

## 2.1 | Attacking Privacy

In their paper, Béres et al. (2021) focus on deanonymizing Ethereum users. They highlight how account-based blockchains encourage address reuse, facilitating user tracking.

Their deanonymization exercise gathered information from different sources for building user profiles. These included Twitter, the Ethereum Name Service (ENS), HumanityDAO[1] and the Tornado Cash[2] mixer.

This process allowed homing in on a set of users, whose activity could be tracked further. Looking at the smart contracts they interacted with, the research exposed the type of services used, which included gambling and adult services.

Feng et al. (2019) surveyed the prevalent privacy technologies in blockchains. They also describe various information leak sources. Starting from the network level, IPs can be mapped to blockchain addresses. Graph analysis clusters related addresses. Transaction information exposes user time-zones and wealth levels. Capturing user data through different vectors further enriches user profiles.

Targeted attacks supplement proactive information gathering. DoS attacks can reduce user connectivity options enticing them to opt for less secure communication. Sybil attacks can reduce anonymity whenever privacy relies on hiding within a group of parties.

[1] Humanity DAO Post Mortem [Accessed Sep 2021] - `https://tinyurl.com/2e7w38mn`
[2] Tornado Cash [Accessed Sep 2021] - `https://tornado.cash/`

## 2.2 | Privacy Preserving Technologies

A primitive but effective approach to privacy involves replacing data with hashes. The raw data is kept off-chain, keeping only its fingerprint on-chain. Zyskind et al. (2015) use this approach. However, this is only applicable if the smart contract logic does not require access to the hash preimage.

Feng et al. (2019) categorize on-chain privacy preserving technologies, breaking these into identity and transaction privacy.

Identity privacy protects the sending and receiving party identities. It also obfuscates the link between them, making transaction graph construction harder and less effective. Ring signatures, Non-Interactive Zero Knowledge (NIZK) proofs and mixers are the prevalent technologies used in identity privacy.

Transaction privacy hides the on-chain application data. This includes a broad application specific data set and ideally also metadata such as transaction timestamps. Here, homomorphic cryptosystems and NIZK proofs are identified as the most popular options.

This dissertation implements NIZK proofs and employs mixer design patterns. Thus, these are discussed in detail in the sections and chapters that follow.

Ring Signatures involve bringing together multiple signatures such that to hide the true sender. The message is signed with the sender's private key and the public keys of other dummy participants. However, an attacker cannot tell these apart.

Homomorphic cryptosystems, allow performing operations on encrypted data. For example, one can have two encrypted integers and compute their encrypted sum. This is achieved without access to the unencrypted values.

This is certainly not a complete list of privacy technologies. Unterweger et al. (2018) present an Ethereum solution for customers to select energy tariffs from various offers. Here the customer energy consumption data and chosen tariff stay private. This solution uses quantized embeddings, transforming data into vectors. From these vectors, one calculates the distance between various offers, identifying the best tariff.

## 2.3 | Zero Knowledge Proofs

ZKPs allow provers to demonstrate having knowledge of some secret. Verifiers confirm that provers truly have the claimed knowledge without learning anything about their secret.

ZKP systems must satisfy the properties of completeness, soundness, and zero-knowledge. Completeness allows honest provers to convince honest verifiers whenever making true statements. Soundness ensures that verifiers reject false statements. Zero-knowledge ensures that verifiers learn nothing about the provers' secret knowledge.

NIZK proofs are a ZKP sub-category allowing provers and verifiers to fulfil their roles without direct interaction. Different NIZK proof implementation approaches are available. This dissertation only considers the most widely used option, zkSNARK. As Reitwiessner (2016) explains, the acronym highlights the key scheme properties:

**Succinctness** refers to the small proof size that such schemes produce. Proof generation is computationally intensive, requires several megabytes of storage and is done off-chain. Verification happens on-chain and is much less resource intensive, as it handles the succinct proof.

**Non-interactivity** highlights independent proof generation. Proofs can be passed as transaction arguments and deterministically verified by smart contracts.

**Arguments** refers to the system having computational soundness. The system is not expected to be secure against adversaries having unlimited computational power.

**Knowledge** requires that proofs can only be generated by provers having the claimed secret knowledge (witness).

This dissertation implements the Pinocchio PGHR13 (Parno et al., 2013) and Groth16 (Groth, 2016) zkSNARK schemes. Pinocchio is attributed to be the first scheme allowing blockchain zkSNARK implementation. Groth16 is the zkSNARK generating the smallest proofs and is considered an efficiency benchmark.

These schemes are composed of three operations: setup, proof generation and verification. The setup process generates a Common Reference String (CRS), composed of a proving key and a verification key. Understanding CRS generation helps appreciating how zkSNARKs work. Section 2.3.2 provides more details on how zkSNARKs are constructed.

### 2.3.1 | Proof of Correct Computation

ZKPs are often described to provide a proof of correct computation. A computation can be expressed in terms of relations. For example, the relation between a hash and its preimage can be expressed as a function that given the preimage returns a hash.
$h = HASH(preimage)$

To construct a ZKP for a specific computation, one starts by defining its relation programmatically. This definition is used by the setup process to produce the proving and verification keys. One can think of proof generation to be a process of recomputing the relation in the encrypted space by combining the relation inputs with the proving key. A proof is only generated if the inputs truly satisfy the relation. Verification is then largely limited in ensuring two facts. Firstly, it ensures that the presented proof is the product of the designated proof generator. Secondly, it ensures that the public inputs provided on proof generation are identical to those provided on verification.

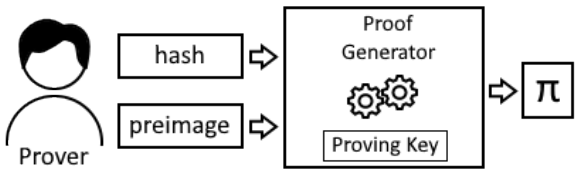


Figure 2.1: Proof Generation from Public and Secret Inputs

Figure 2.1 depicts proof generation for the hash/preimage relation. Here, the proof allows claiming preimage knowledge, without disclosing it. The generator requires the input of all relevant variables, the hash and preimage. Next, if the inputs truly satisfy the relation, a verifiable proof $\pi$ is provided. No information can be gleaned from this, beyond the fact that it results from an honest computation.

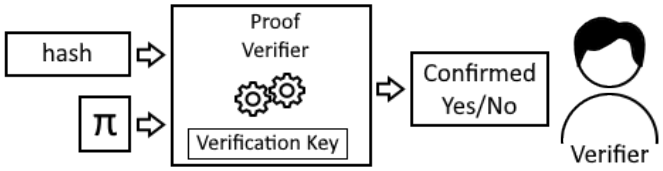


Figure 2.2: Proof Verification from Public Inputs and Proof

Given its strong obfuscation, a proof can only be interpreted by a specialized verifier depicted in figure 2.2. This verifier is armed with the verification key that twins with the generator's proving key. Combining this key with the proof and public inputs allows verifying honest proof computation. Here the verifier is only provided with the hash since the preimage must remain secret. Thus, a user proves knowledge of a secret preimage for a publicly known hash and hashing function.

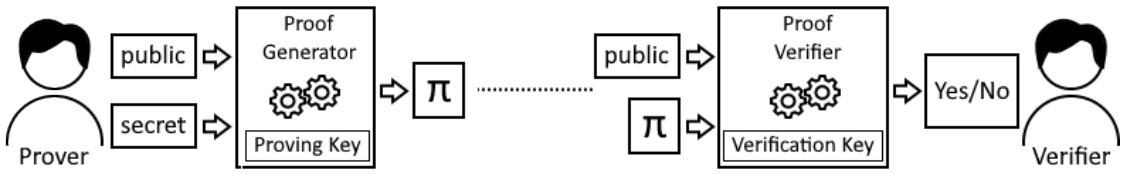


Figure 2.3: End-to-End Proof Generation and Verification

Generalizing the discussion, the computation could be described as a function that takes a set of secret inputs and public inputs as shown in figure 2.3. The secret inputs encapsulate the knowledge being proven, whilst the function and public inputs provide the necessary context that precludes ambiguity. Choosing between secret and public inputs, a balance is sought between privacy and a guarantee of unambiguous verification.

### 2.3.2 | Basic zkSNARK Construction

This section provides an informal description of proving and verification key generation in zkSNARK schemes. This is a multi-stage process, starting from the relation about which statements are to be proven.

The code for the hash preimage ZKP discussed in section 2.3.1, would quickly grow too large to track. Instead, this section considers the relation:

$x^3 + x + 5 == y$

Consider proving knowledge of the secret value $x$ that satisfies this relation for some public value $y$. This relation was used in an article series[3] explaining zkSNARKs by Vitalik Buterin. Here the same example is used.

The first stage involves representing this expression using an arithmetic circuit composed of additive and multiplicative gates. Each gate takes two inputs and returns one output. However, the relation must first be coded as a set of statements as follows:

```
sqr      = x * x
cubed    = sqr * x
cubed_px = cubed + x
out      = cubed_px + 5
```

Each statement has two inputs and one output, as required by the arithmetic gates. Intermediate variables were introduced as needed and the result $y$ was renamed to *out*.

[3]Quadratic Arithmetic Programs: from Zero to Hero [Accessed Aug 2022] - `https://tinyurl.com/2fkb7e9m`

Instead of looking at this code as a linear sequence of statements, each statement could be considered as a condition to be satisfied. If a prover provides values for all variables ($out, x, sqr, cubed, cubed\_px$) satisfying each condition, the required knowledge is still proven. Based on this intuition, the code is converted into a set of constraints, forming a Rank-1 Constraint System (R1CS). Here each statement has the format:

$left * right == output$

Thus, the example is converted as shown in table 2.1:

| **Statement** | **Constraint** | | | | |
|---|---|---|---|---|---|
| | $left$ | $*$ | $right$ | $==$ | $output$ |
| $sqr = x * x$ | $x$ | $*$ | $x$ | $==$ | $sqr$ |
| $cubed = sqr * x$ | $sqr$ | $*$ | $x$ | $==$ | $cubed$ |
| $cubed\_px = cubed + x$ | $(cubed + x)$ | $*$ | $one$ | $==$ | $cubed\_px$ |
| $out = cubed\_px + 5$ | $(cubed\_px + 5 * one)$ | $*$ | $one$ | $==$ | $out$ |

Table 2.1: Constraint System

Note the introduction of *one*, representing the unity constant.

This set of constraints is next converted into a Quadratic Arithmetic Program (QAP). The idea is to generate three polynomials, one for each of the left, right and output columns in table 2.1 and equate them as follows:

$p(s) = L(s) * R(s) - O(s)$

To construct these polynomials, the constraints are assigned a sequential index starting from 1. Next, given the constraint index, the evaluation of $L(s)$, $R(s)$ and $O(s)$ is required to reflect the constraint definition as shown in table 2.2.

| **Constraint** | **L(s)** | **R(s)** | **O(s)** |
|---|---|---|---|
| $x * x == sqr$ | $L(1) = x$ | $R(1) = x$ | $O(1) = sqr$ |
| $sqr * x == cubed$ | $L(2) = sqr$ | $R(2) = x$ | $O(2) = cubed$ |
| $(cubed + x) * one == cubed\_px$ | $L(3) = (cubed + x)$ | $R(3) = one$ | $O(3) = cubed\_px$ |
| $(cubed\_px + 5 * one) * one == out$ | $L(4) = (cubed\_px + 5 * one)$ | $R(4) = one$ | $O(4) = out$ |

Table 2.2: Quadratic Arithmetic Program Construction

Each of the left, right, and output polynomials are derived by summing up another set of polynomials, one for each of the variables ($one, out, x, sqr, cubed, cubed\_px$).

For example, $L(s)$ is composed of:

$$\begin{aligned} L(s) = \quad & 1 * L_{one}(s) + \\ & out * L_{out}(s) + \\ & x * L_x(s) + \\ & sqr * L_{sqr}(s) + \\ & cubed * L_{cubed}(s) + \\ & cubed\_px * L_{cubed\_px}(s) \end{aligned}$$

The per variable polynomials act like switches, including/excluding variables to reflect the constraint composition. Their roots (zeros) coincide with the constraint indices where the variable is not included. On the other hand, for constraint indices where the variable is included, the polynomial evaluates to the required constant multiple. Table 2.3 shows the expected evaluation of the polynomials composing $L(s)$.

| $s$ | $L(s)$ | $L_{one}(s)$ | $L_{out}(s)$ | $L_x(s)$ | $L_{sqr}(s)$ | $L_{cubed}(s)$ | $L_{cubed\_px}(s)$ |
|---|---|---|---|---|---|---|---|
| 1 | $x$ | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | $sqr$ | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | $(cubed + x)$ | 0 | 0 | 1 | 0 | 1 | 0 |
| 4 | $(cubed\_px + 5 * one)$ | 5 | 0 | 0 | 0 | 0 | 1 |

Table 2.3: Evaluation of per variable polynomials for L(s)

This construction ensures that $p(s)$ equates to 0 when evaluated at constraint indices. Thus, $p(s)$ can be expressed in terms of its roots and some other polynomial $h(s)$:
$p(s) = (s-1) * (s-2) * (s-3) \ldots (s-m) * h(s)$
$t(s) = (s-1) * (s-2) * (s-3) \ldots (s-m)$
$p(s) = t(s) * h(s)$

$t(s)$ is referred to as the target polynomial and is publicly known.

Now that the problem is expressed in terms of polynomials, provers can be challenged to evaluate $p(s)$ and $h(s)$ for some random point. The premise is that only a prover knowing the correct secret inputs would be able to provide a valid solution. Validation would then center around verifying this property:

$p(s)/h(s) == t(s)$

These computations happen in the encrypted space. The random evaluation point, together with many other constants, are generated and encrypted during the trusted setup. The unencrypted values must be destroyed as their disclosure would compromise the ZKP soundness. Indeed, calling the setup trusted, highlights the reliance on an

honest setup execution.

This discussion leaves out many details. However, enough progress was made to better define what proofs, proving keys and verification keys are made of. A proving key is a set of constants that when combined with the public and secret inputs, solve a set of polynomials about which knowledge is being proven. The proof itself is the solution resulting from such computations.

The verification key is composed of another set of constants. This is combined with a proof to test a property that (with very high probability) should only be satisfied if the proof was honestly computed.

The zero-knowledge property results from performing computations in the encrypted space and the inclusion of random values that obfuscate the proof without inhibiting verification.

This section provided more insight into the construction of zkSNARKs. For a more detailed explanation of this process, the walkthrough provided by Petkus (2019) is recommended.

## 2.4 | Private Tokens

Privacy in the decentralized setting has been attracting research and development for years. Zerocoin (Miers et al., 2013) and Zerocash (Ben Sasson et al., 2014) led to the launch of Zcash in 2016, a significant milestone for ZKP-based privacy. In the same year, the Baby ZoE project[4] explored the integration of Zerocash over Ethereum. Furthermore, the collaboration[5] between ZCash and the Ethereum Foundation led to the inclusion of precompiled contracts (Wood, 2014) facilitating ZKP support on Ethereum.

This research spilled over to other projects. Clearmatic's Zeth (Rondelet and Zajac, 2019) carried forward the idea of developing Zerocash over Ethereum. Nightfall (Konda et al., 2019) from Ernst & Young, and the Aztec Protocol (Williamson, 2018) also include similar design traits. This helped establishing a design pattern for privacy based on private tokens.

### 2.4.1 | Mixer Smart Contract

This dissertation heavily reuses Zeth for its implementation. Thus, this section describes decentralized mixers using Zeth as an example. Its core idea is the creation

[4]Baby ZoE project [Accessed Sep 2021] - `https://github.com/zcash-hackworks/babyzoe`

[5]An Update on Integrating Zcash on Ethereum (ZoE) [Accessed Sep 2021] - `https://tinyurl.com/y9dbmc4j`

of an anonymity set, where users work with set elements whilst external observers are unable to determine which element is being used. In Zeth, set elements could be a cryptocurrency or a fungible token.

This chapter describes examples of private tokens wrapping Ether (ETH). However, tokens could represent anything, possibly voting ballots, passports, or medical data.

One way to start using a mixer is through a minting transaction. This deposits ETH into the mixer and returns private tokens wrapping the deposited amount. A Zeth private token is known as a ZethNote.

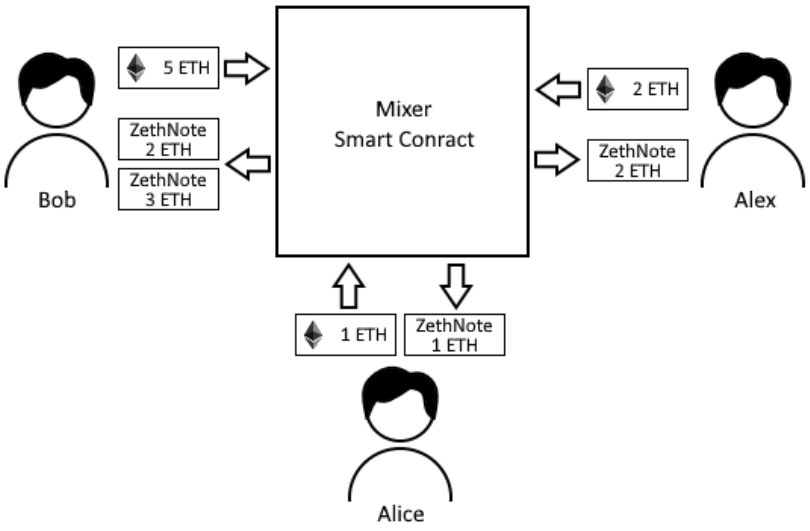


Figure 2.4: Mixer Minting - Depositing ETH to receive ZethNotes.

Figure 2.4 illustrates minting. Bob deposits 5ETH and receives two ZethNotes, wrapping 2ETH and 3ETH. Other users are also minting. The mixer is shown as a perfectly opaque box. Section 2.4.2 shows how this is not the case.

ZethNotes compose the anonymity set elements, hiding the amounts being wrapped. However, minting does not hide the deposited amounts. Thus, even though the ZethNote value is hidden, in case of Alice and Alex this can be inferred from the transaction. In comparison Bob is enjoying more privacy since his minting produces two ZethNotes. One can infer the total amount, but not how these are split.

Users are now able to transfer ZethNotes. A transaction fully spends the input ZethNotes whilst producing fresh output ZethNotes such that the total input and output amounts balance out.

The main benefit of transacting ZethNotes is the privacy enjoyed from exchanging anonymity set elements. Figure 2.5 illustrates a transfer. Bob privately picks his 3ETH

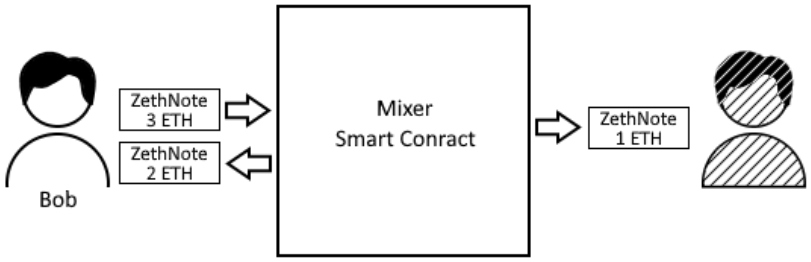


Figure 2.5: Mixer Transfer - Splitting a ZethNote, between sender and recipient.

ZethNote. He creates two output ZethNotes, one wrapping 1ETH for the intended recipient and another wrapping the change for himself. The recipient is shown unnamed since his/her identity is known only to Bob.

This transfer shows an alternative method for becoming a ZethNote owner. Bob might have introduced a new user to the mixer in full anonymity. This user will only disclose having a ZethNote on submitting his/her first mixer transaction. Meanwhile many other transactions could be completed, making it much harder to infer when the user's ZethNote was created.

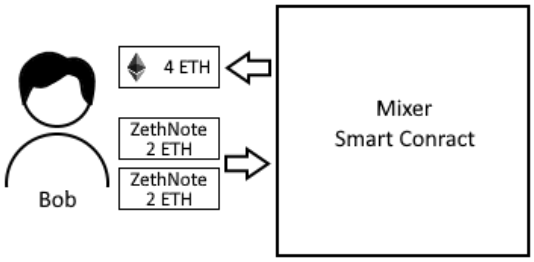


Figure 2.6: Mixer Burning - Spending ZethNotes to withdraw ETH.

Finally, users are also able to withdraw their ETH. This involves burning ZethNotes and withdrawing the wrapped amount from the smart contract. Figure 2.6 shows Bob withdrawing two ZethNotes. Withdrawal triggers a regular ETH transfer. Everyone can see 4ETH flowing from the mixer to Bob.

### 2.4.2 | Transaction Graphs

So far, the mixer was presented as an island of anonymity within a blockchain enjoying little anonymity. Ethereum only provides pseudonymity and as demonstrated by Béres

et al. (2021) its users are easy deanonymization targets.

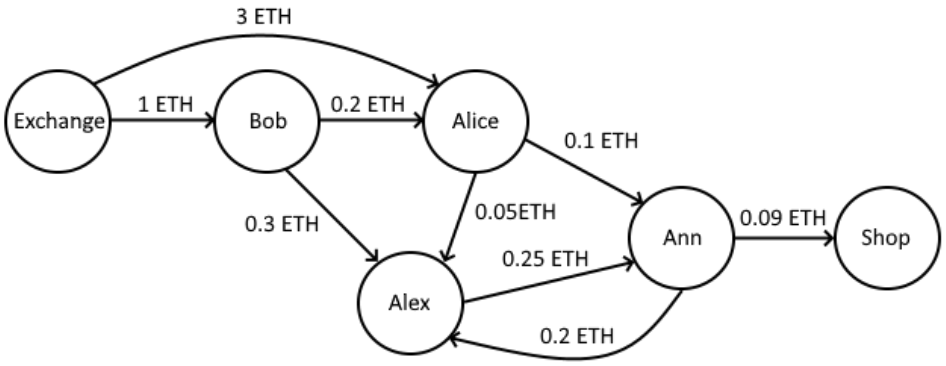


Figure 2.7: Transaction Graph without Mixer

Ethereum transactions disclose sender and recipient addresses, together with the transacted amounts. This allows constructing rich transaction graphs exposing wealth levels, services used and relationships. Figure 2.7, shows one such graph. Here Bob and Alice purchase ETH from an Exchange. Thereafter, they make transfers that also involve Alex and Ann. Furthermore, Ann is shown paying for some purchase. The transaction order is not shown to avoid clutter. Nevertheless, the sequence is also readily available from the transaction timestamps.

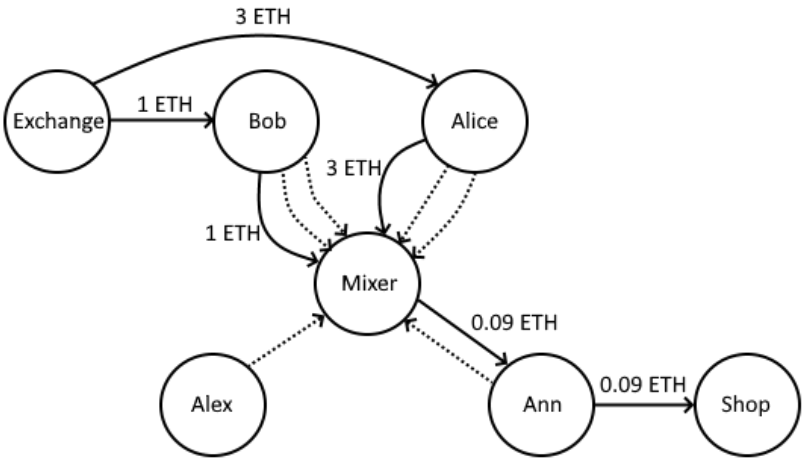


Figure 2.8: Transaction Graph with Mixer

Figure 2.8 shows what the graph would look like if the four used the mixer. Solid lines represent transactions leaking the transferred amounts. Dotted lines represent transfers fulfilled within the mixer, hiding both amount and receiver. An observer can see that the four are using the mixer, how many times each initiates a transfer and the transaction order. Beyond that, the amounts and the sender-to-recipient links are not available. Whereas some information is still leaked, this is significantly less.

### 2.4.3 | Private Token Design

To demonstrate how mixer obfuscation works, this section describes private token storage and handling.

#### 2.4.3.1 | Token Data

A private token is composed of the following data:

Ownership, $a_{pk}$ - A public key identifying the token owner.

Application Properties, $v$ - Application specific data. A currency token here includes the currency amount, a voting token could include the voting preference, etc.

Random Data, $r$ - Data for securing the on-chain token representation.

Token Identifier, $\rho$ - A unique identifier discussed in section 2.4.3.3.

Thus, a common token construction is composed of:
$Token = (a_{pk}, v, r, \rho)$

This data is either private or security sensitive and cannot be stored on-chain in cleartext. Instead, commitments and nullifiers are used.

#### 2.4.3.2 | Commitments

Commitments and nullifiers are two fundamental private token building blocks. These are implemented using cryptographic primitives, whose choice and application determine the solution security level. This dissertation completely reuses the primitives chosen for Zeth and closely replicates its commitment and nullifier construction.

A commitment scheme is composed of a Commitment Function (Com) that given a set of inputs returns an output $cm$:
$cm = Com(inputs)$

The scheme must provide strong hiding and binding properties. A hiding scheme is resistant to preimage attacks, where given the commitment output, an attacker is unable to discover the inputs. A binding scheme is collision resistant, an attacker is unable to find a different set of inputs that produce the same output.

Commitments allow storing tokens privately. The token is said to be the commitment opening. The binding property ensures that not even its creator is able to modify the token, once published. The hiding property ensures external observers cannot discover any of the token data. In Zeth, commitments are computed by hashing the token using Blake2s:
$cm = Blake2s(a_{pk}||v||r||\rho)$

For each new token a unique commitment is published on-chain, establishing a one-to-one mapping. However as highlighted earlier, mixer users should be able to spend tokens without anyone knowing which token is being spent. Figure 2.9 shows Bob making a transfer to Alice, who in turn transfers to Alex. The smart contract is now shown as a large set of opaque commitments. Dotted lines point at commitments that are being secretly referenced, whilst continuous lines point at commitments that are being publicly added. Here not even Bob, the original token creator, can tell whether Alice spent the token he created or some other token she owned.

Since users want to avoid disclosing which token is being spent, transfer/withdrawal operations cannot modify on-chain commitments. Furthermore, the solution must preclude double spending. Thus, a distinct record of spent tokens is required, raising the need for nullifiers.

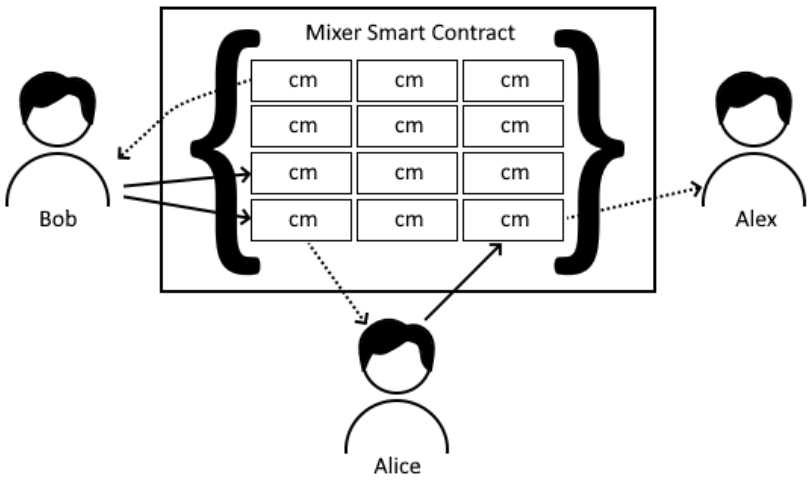


Figure 2.9: Unlinking Transactions - Token creator cannot track subsequent spends.

#### 2.4.3.3 | Nullifiers and Double Spending

Nullifiers are something akin to spending receipts. Users construct and submit nullifiers whenever spending tokens. Their construction ensures that a given token always produces the same nullifier, forming another secret one-to-one mapping. Mixers ensure that each spend corresponds to a unique nullifier, effectively blocking double spending.

For this to work, a nullifier satisfies these requirements:

1. The nullifier must represent the token being spent. This is achieved through the unique token identifier $\rho$.
2. The nullifier can only be computed by its owner. Tying its computation to the owner's private identity key $a_{sk}$ satisfies this requirement.
3. The nullifier should be unique, there should be no collisions across nullifiers.
4. The nullifier should not allow identifying the token to which it pertains.

A relationship that satisfies these requirements is:
$nf = PRF_{a_{sk}}(\rho)$

Here a Pseudo Random Function (PRF) satisfies the collision resistance and privacy requirements. It also brings together the owner's identity and the token identifier. Again, in Zeth nullifiers are computed by hashing all values using Blake2s. Roughly this is computed as follows:
$nf = Blake2s(a_{sk}||\rho)$

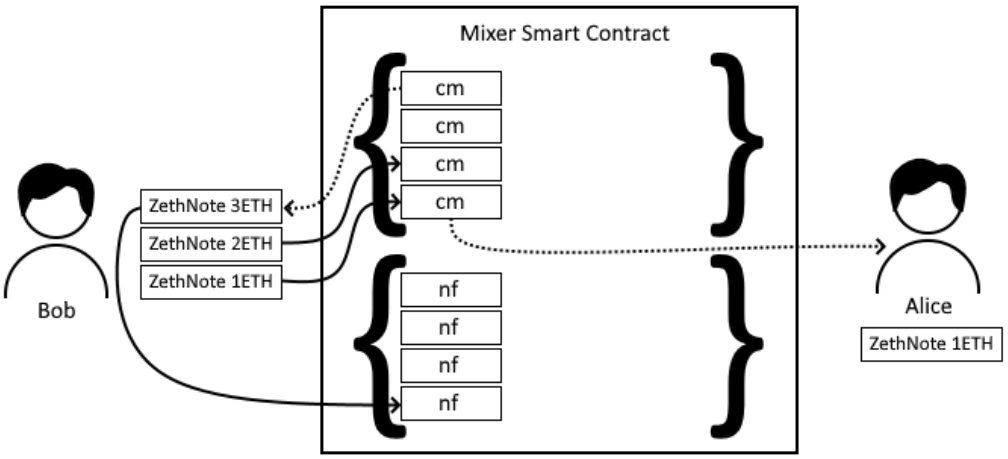


Figure 2.10: Tokens, Commitments and Nullifiers

Figure 2.10 brings together tokens, commitments, and nullifiers. Next to Bob and Alice, the figure shows the tokens known to them on transaction completion. Initially Bob owns a ZethNote wrapping 3ETH represented by an on-chain commitment, for which he computes its nullifier. He also constructs two new ZethNotes wrapping 1ETH and 2ETH, together with their commitments. The 1ETH ZethNote specifies Alice as the owner, whilst the 2ETH identifies Bob himself. The transfer is finally recorded by publishing the nullifier and commitments on-chain.

#### 2.4.3.4 | Anonymity Set

Section 2.4.1 introduced the anonymity set conceptually. Now, this can be defined in terms of commitments and nullifiers.

Users should never publish a token's commitment and nullifier together. On token creation only its commitment is published, whilst on spending, only its nullifier is published. Keeping them separate is what prevents transaction chaining. Even so, their on-chain publication discloses the total unspent tokens:
$n = Total\ Commitments - Total\ Nullifiers$

Hence, spending involves picking one token out of n elements, making n the anonymity set size.

#### 2.4.3.5 | Token Exchange

Storing tokens as opaque commitments is effective at blocking disclosure. So effective, that not even the intended recipients will recognize their tokens. Spenders must some-

how disclose the token details to their recipients, raising the need for a parallel communication channel depicted in figure 2.11.

Bob transfers 1ETH to Alice by publishing the commitments/nullifiers on-chain and passing its details over the encrypted channel. Alice identifies her tokens and ensures that the token:

- is wrapping the agreed amount
- correctly assigns ownership to her
- was correctly published on-chain
- is spendable (section 2.4.3.7)

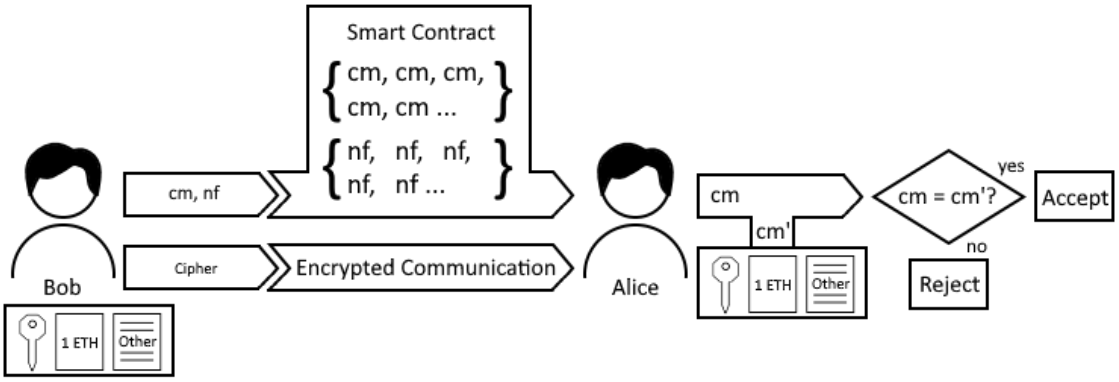


Figure 2.11: Exchanged tokens are cross verified by the receiver.

Correct commitment publication is verified as shown in figure 2.11. Alice recomputes the token commitment and verifies that it matches an on-chain commitment.

#### 2.4.3.6 | User Identities

The mixer is now composed of two parallel systems both requiring secure user identities. This raises the need for three asymmetric key pairs.

1. Ethereum key pair for users to submit transactions and pay for gas.
2. Identity key pair for assigning and proving token ownership.
3. Encryption key pair for transferring token data.

#### 2.4.3.7 | Spendable Tokens and the Faerie Gold Vulnerability

Section 2.4.3.5 stated that recipients should verify that tokens are spendable. This is necessary for Faerie Gold vulnerability[6] protection, a potential exploit resulting from the construction of tokens and nullifiers. To illustrate this, we put the two side-by-side:
$Token = (a_{pk}, v, r, \rho)$
$nf = PRF_{a_{sk}}(\rho)$

New tokens are constructed by spenders, who hand them to their intended recipients. The corresponding nullifier is only needed in a subsequent spending transaction. The problem arises if a recipient accepts two tokens having the same identifier $\rho$, since both would produce the same nullifier.

A transaction presenting an already recorded nullifier is blocked as a double spend. To prevent such a problem:

1. Spenders are forced to generate $\rho$ using a predefined collision resistant function such that to preclude malicious generation. For details, the reader is referred to the Zeth protocol specification document[7].

2. Recipients are required to verify that none of their previous tokens contained the same $\rho$. Furthermore, on receiving a token, recipients can immediately compute the nullifier and check that no matching nullifier is published on-chain.

#### 2.4.3.8 | Rule Enforcement in Zero Knowledge

So far, the smart contract was described as the commitment and nullifier store. In addition, the contract must also enforce the workflow rules. This is challenging since commitments and nullifiers look no different from random integers.

This void is filled using ZKPs. As explained in section 2.3.1, ZKPs allow constructing a proof of correct computation. Among others, this is applied to ensure that spenders correctly compute commitments and nullifiers. Figure 2.12 again depicts the 1ETH transfer already shown in figure 2.10, adding many new details. Bob now has a proof generator assisting him in preparing the transaction inputs. At the smart contract, a proof verifier assists transaction validation.

The proving key *pk* and verification key *vk* are shown to be integrated within the proof generator and verifier. These are constants to which the respective ZKP components have access.

[6] Faerie Gold vulnerability [Accessed Sep 2021] - `https://github.com/zcash/zcash/issues/98`
[7] Zeth Protocol Specification [Accessed Sep 2021] - `https://tinyurl.com/et8rdtd6`

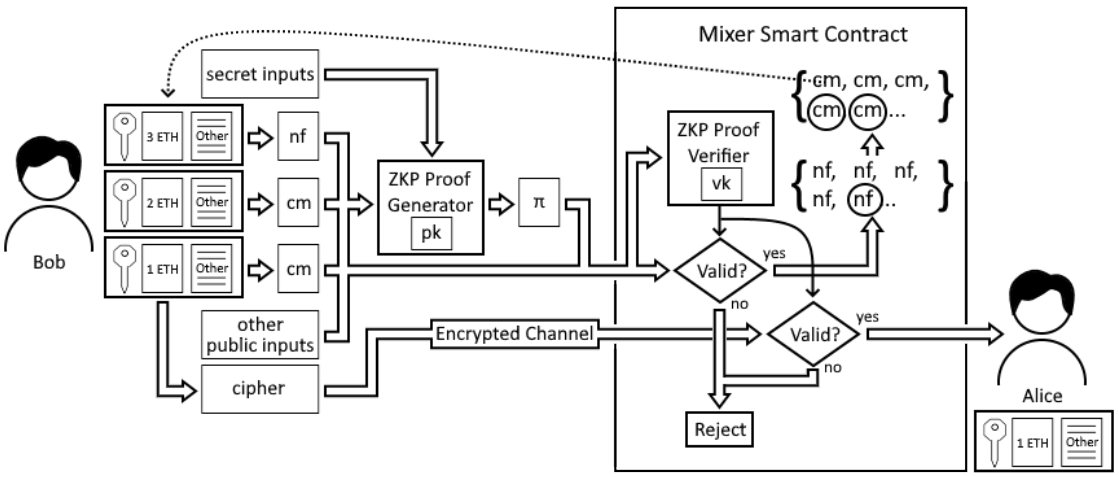


Figure 2.12: Mixer Proof Generation, Transmission and On-Chain Verification

On the left, Bob computes nullifiers and commitments for the spent and newly generated tokens. Two other input sets labelled as "secret inputs" and "other public inputs" are also provided. Together these compose the proof generator input. Next, if all inputs were honestly computed, the generator returns a proof $\pi$.

The proof and public inputs are passed to the smart contract. These are fed into the ZKP verifier, which returns a Boolean verification result. If not successful, the transaction is rejected. Otherwise, the commitments and nullifiers are persisted.

Lastly, the figure also shows the cipher exchange trail. The 1ETH token is encrypted and delivered to Alice. What's interesting, is the smart contract integration. Cipher delivery is shown to be dependent on the ZKP verification outcome. More on this in section 2.4.5.

### 2.4.4 | ZKP Constructs

This section discusses some ZKP constructs used in private token solutions. The libsnark development framework encourages the decomposition of ZKPs into gadgets. Individual gadgets host different ZKP constructs, that are then combined into one ZKP. Thus, each of the discussed constructs would indeed be normally developed as a separate gadget.

Examples are given with reference to Zeth although the same constructs are used in most ZKPs including the dissertation's consent solution.

#### 2.4.4.1 | Proof of Commitment/Nullifier Computation

Commitments and nullifiers should only result from the specified relationships. For Zeth these are roughly:
$cm = Blake2s(a_{pk}||v||r||\rho)$
$nf = Blake2s(a_{sk}||\rho)$

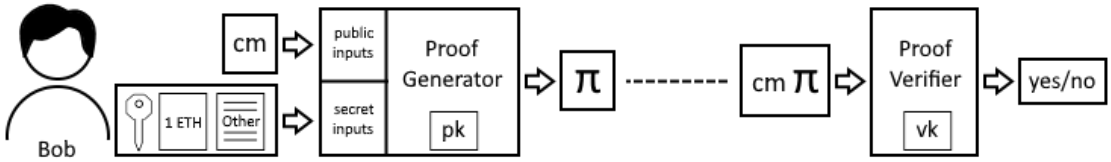


Figure 2.13: Proof of Commitment Opening - Prover provides a commitment and its preimage. Verification requires commitment and proof.

Figure 2.13 shows Bob, the prover, supplying a commitment and a token to the proof generator. The generator only produces a proof if the token is the correct commitment opening.

On the other side, the verifier is supplied a commitment and a proof. Verification succeeds if:

1. The proof was indeed produced by the correct generator. Hereafter, the discussion assumes this to always be the case.
2. The verifier's commitment matches the one supplied to the generator.

Bob thus proves knowledge of the commitment opening.

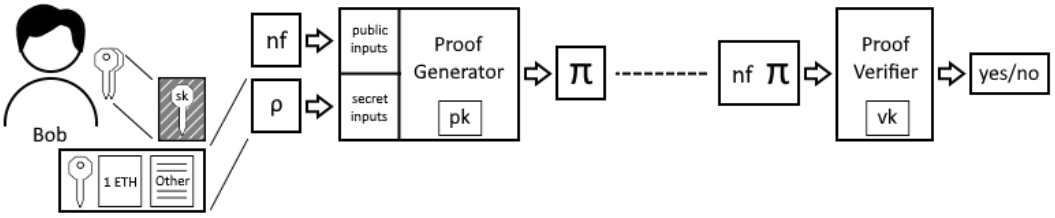


Figure 2.14: Proof of Nullifier Computation - Prover provides nullifier and its preimage. Verification requires nullifier and proof.

Figure 2.14 shows the prover supplying a nullifier, a private key, and a token identifier $\rho$ to the proof generator. The generator only produces a proof if the inputs satisfy the nullifier relationship. On the other side, the verifier is supplied a nullifier and a proof.

Verification only succeeds if the verifier's nullifier matches the one supplied to the generator. In this manner Bob proves knowledge of the private key and token identifier necessary for computing the nullifier.

#### 2.4.4.2 | Proof of Ownership

A token owner is established from the included public key. The owner is then required to prove having the corresponding private key. The nullifier ZKP construct in section 2.4.4.1, required Bob to provide his private key. Thus, the proof could only be generated by the token owner. In comparison, the commitment ZKP did not require the private key. Hence, only knowledge of the commitment opening was proven, not its ownership.

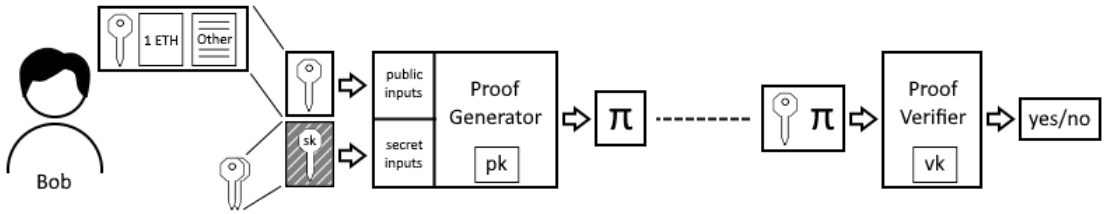


Figure 2.15: Proof of Ownership - Prover provides key pair. Verification requires public key and proof.

Figure 2.15 shows an attempt at proving token ownership. Bob supplies the proof generator with the token public key and his private key. The generator only produces a proof if the two keys form a pair. On the other side, the verifier is supplied a public key and a proof. Verification only succeeds if the verifier's public key matches the one supplied to the generator. This time, we have only proven key pair ownership. The verifier cannot tell whether the proof truly relates to a token.

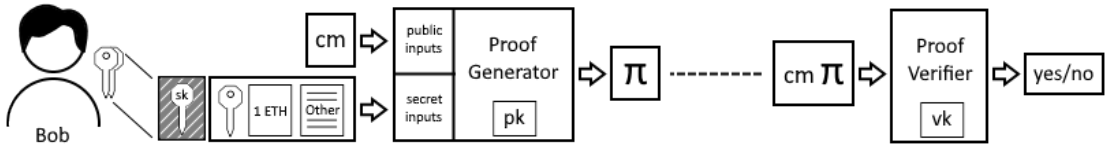


Figure 2.16: Merging a proof of commitment opening and a proof of ownership.

Figure 2.16 merges the commitment opening and key pair ownership proofs. Furthermore, it removes the public key from the public inputs. This became redundant since it is already included within the token secret input. More importantly obfuscating identities is a key mixer design goal. Its public presence would have undermined our

need to obfuscate the spender-to-recipient link. The proof generator now only produces a proof if:

1. The token is the correct commitment opening.
2. The token's public key and Bob's private key form a pair.

This time, the commitment opening can only be proven by the private key holder. On the other side, verification only succeeds if the verifier's commitment matches the one supplied to the generator. Thus, both knowledge of the commitment opening, and ownership are proven.

#### 2.4.4.3 | Proof of Membership

Section 2.4.3.3 described spenders secretly referencing the token being spent and computing its nullifier. Section 2.4.3.4 highlights how for the same token the commitment and nullifier are never published together. Thus, a mechanism to reference previously published commitments is needed. Proof of membership is a ZKP construct that allows doing just that.

From the state of the art, two proof of membership alternatives were identified, one based on RSA accumulators, and another based on Merkle tree accumulators. Of the two, only Merkle tree implementations were identified on Ethereum. Furthermore, Zcash includes an open github issue[8] discussing both alternatives. This justifies their Merkle tree preference based on lower computational costs. A similar issue[9] at the Semaphore (Gurkan et al., 2020) github was closed with the same conclusion. Here the cost is expressed in terms of gas. The code repositories for Zeth and Nightfall also confirmed Merkle tree implementations. Thus, this dissertation only considered proof of membership based on Merkle trees.

Understanding proof-of-membership and looking at its ZKP construct is insightful to appreciate how one can reference a set member without disclosing which member is being referenced. The sub-sections that follow describe this construct in more detail.

## Proof Input Parameters

To describe how a proof of membership is constructed an understanding of how Merkle tree leaves are referenced is necessary.

[8] Zcash Accumulator Selection Issue [Accessed Oct 2021] – `https://tinyurl.com/6yd8p3xe`

[9] Semaphore Accumulator Selection Issue [Accessed Oct 2021] – `https://github.com/barryWhiteHat/semaphore/issues/2`

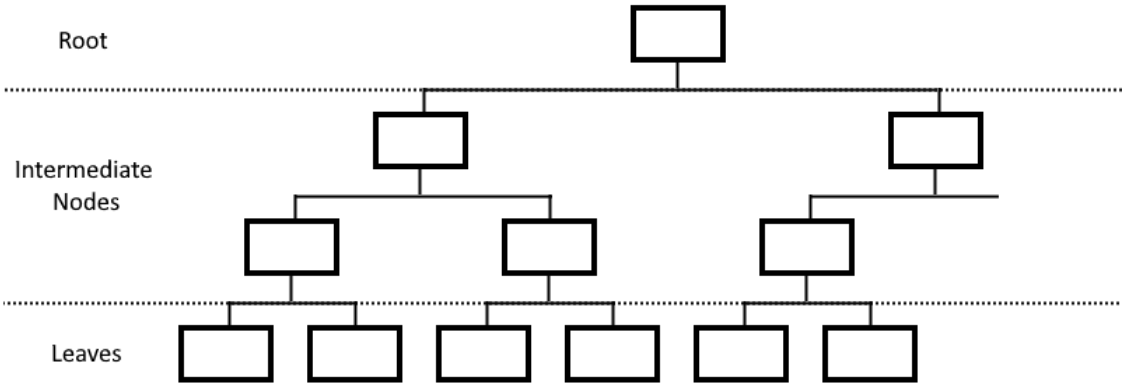


Figure 2.17: Binary Tree

Figure 2.17 shows a binary tree, composed from a set of nodes. The top-most node is the root, whilst the bottom-most nodes are the leaves. Each non-leaf node can be described as the parent of two child nodes. Two nodes having the same parent are each other's sibling.

A design pattern has been established around a very efficient, append-only, fixed size tree construction, whose leaves are set from left to right. Thus, right hand leaves are known to be empty. Based on this, Nightfall includes a well-documented reusable implementation called Timber[10].

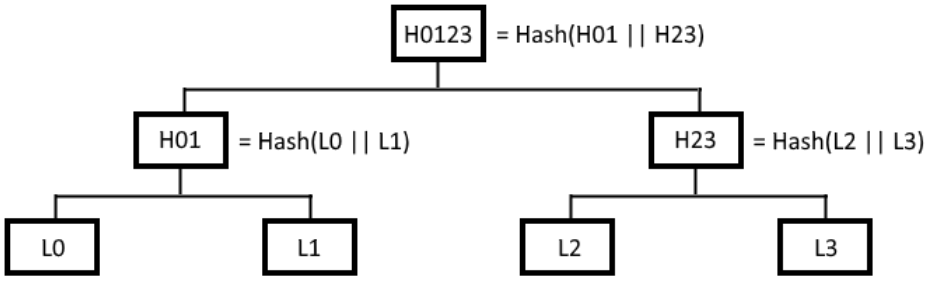


Figure 2.18: Tree Construction from Hashes

In private tokens, Merkle tree leaves store the anonymity set elements, commitments. Figure 2.18 shows the tree being constructed from bottom to top. Every pair of siblings is concatenated and hashed together, producing the parent's hash. In turn, the parent node is concatenated with its own sibling and hashed. This is repeated until the root hash is established.

[10]EY Nightfall Timber [Accessed Oct 2021] – `https://github.com/EYBlockchain/timber`

The Merkle tree hashing cryptographic primitive is required to provide strong collision resistance. This ensures having a unique root for every leaf update. Thus, the root provides a fingerprint representing the anonymity set at a given point in time.

A mixer smart contract persists Merkle tree root updates to its state, facilitating efficient proof of membership verification. This does not leak any new information, since it directly results from the commitment leaves, that are already public.

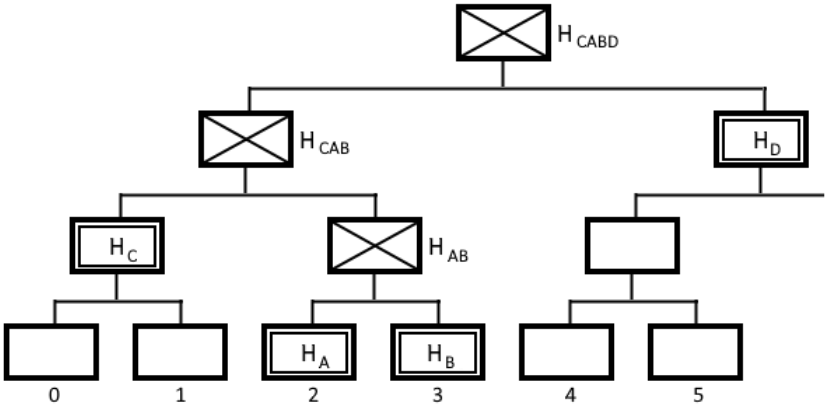


Figure 2.19: Referencing Leaves - Path for traversing tree from leaf to root.

Any leaf can be referenced by a path. This includes the hashes necessary for recomputing the root. Figure 2.19 illustrates how the path for leaf index 2 is obtained. Double edged boxes identify the nodes whose values must be provided. Crossed boxes identify nodes whose hash must be computed, producing intermediate values necessary for reaching the root. Starting from the referenced leaf, hashes are computed iteratively, moving upwards towards the root.

Since node hashes combine two concatenated values, a flag specifying the concatenation order is also required. This determines whether the current node value should be concatenated before or after its sibling. To begin, the reference leaf is concatenated to its sibling found on the right ($flag = 0$). Moving one level up, the resultant parent's hash is concatenated to its sibling found on the left ($flag = 1$). Moving another level up, the sibling node is found on the right ($flag = 0$). These flags are concatenated together into a value known as the address. Thus, a complete path includes the referenced leaf, a list of hashes and an address. Together with the expected root value, these form the proof of membership inputs.

### Merkle Tree Proof of Membership

In private tokens, proof of membership secretly references the commitment mapping to the token being spent.

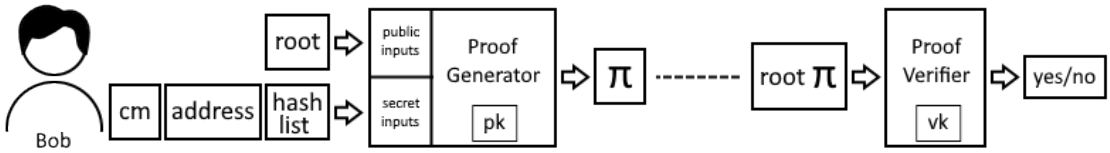


Figure 2.20: Proof of Membership - Prover provides path and root. Verification requires root and proof.

Figure 2.20 shows the prover supplying all the proof of membership parameters. Here the commitment *cm* is the leaf value being referenced. The Merkle root is the only public input, whilst all others are private. The generator will only produce a proof if the secret inputs truly compute the provided root. On the other side, the verifier is supplied a Merkle root and a proof. Verification only succeeds if the verifier's root matches the one supplied to the generator.

This verification is coupled with a smart contract check, confirming that the Merkle root is indeed listed within the set of roots held at the mixer. In this manner Bob proves that:

1. He truly knows the path leading from a commitment to a Merkle root.
2. The Merkle root represents a snapshot of the mixer's anonymity set.

Bob is thus privately referencing a commitment without leaking the specific commitment.

## 2.4.5 | Synchronization and Cipher Exchange

Throughout this chapter, mixer users required on-chain data. In section 2.4.3.5 recipients verified that received tokens mapped to on-chain commitments. In section 2.4.3.7 recipients verified that newly received tokens did not have their nullifier listed on-chain. Such data could be read directly from the blockchain. However, read requests can leak user interests. For example, asking for a specific commitment could disclose token ownership by inference. Malicious node operators could possibly map the user's IP to a

commitment. Subsequent transactions from the same IP could add the Ethereum address to the profile, further advancing deanonymization.

To address these concerns, Zeth implements a solution based on Ethereum events. These convey all relevant data, including token ciphers. Transmission follows a push model, where users listen to all updates. Once the data is downloaded, users privately identify what is relevant to them.

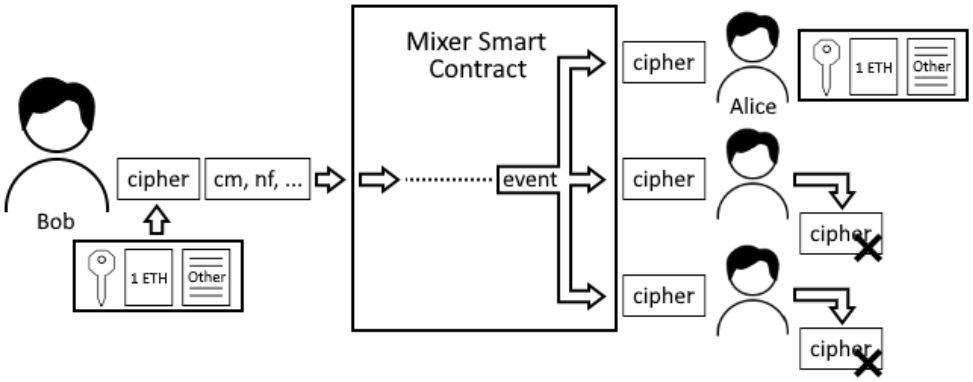


Figure 2.21: Cipher Propagation - Only the intended recipient decrypts the cipher.

Figure 2.21 depicts a token transfer from Bob to Alice. The token cipher is passed to the smart contact together with all other transaction parameters. On successful processing, the smart contract raises an event that includes the cipher. This is received by all mixer users. All of them try to decrypt the cipher, as they seek to discover tokens addressed to them. However, only Alice has the correct decryption key and succeeds in reading the token cleartext.

#### 2.4.5.1 | Data Propagation

Beyond the ciphers, mixer events also propagate nullifiers, commitments and Merkle root updates. Each user is thus able to maintain off-chain copies without raising explicit requests.

Figure 2.22 illustrates transaction data propagation. The mixer persists nullifiers to its list. Commitments are added to the Merkle tree, and the resulting root is added to another dedicated list. Finally, an event packaging the new commitments, nullifiers and the updated root is emitted.

Each mixer user is listening for events. Nullifiers are added to a local list. Commitments are added to a local tree for which the root $r$ is computed. This root is then compared against the one provided by the event. If the two match, the user is in-sync.

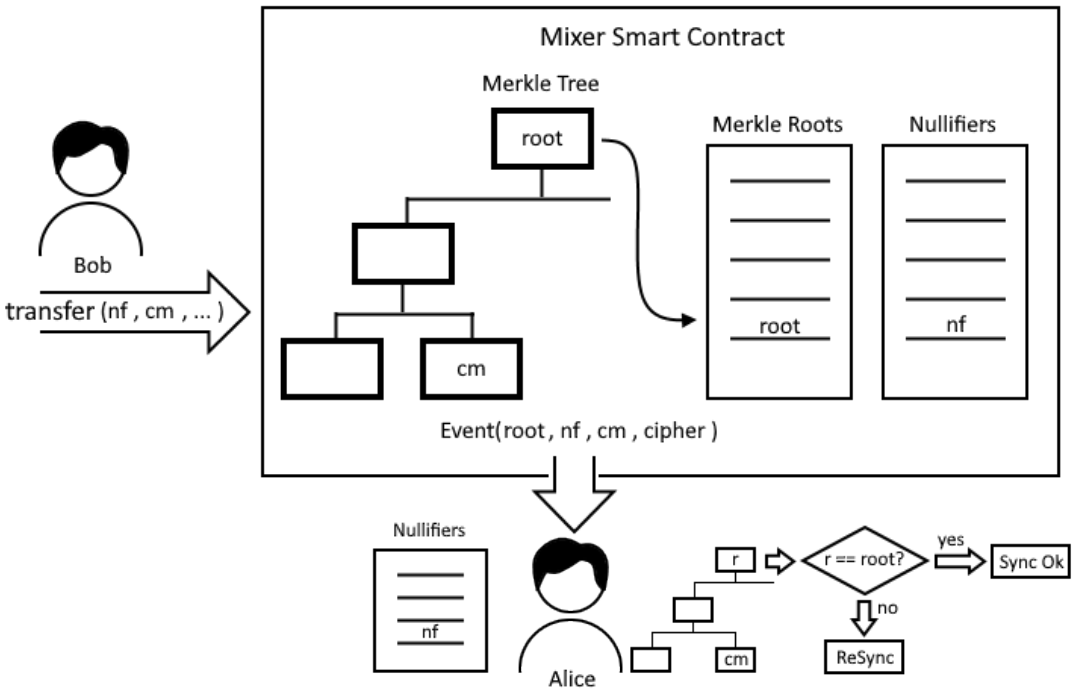


Figure 2.22: Data Propagation - Constructing off-chain data replicas.

Otherwise, the user backtracks and pulls the missing events. Even though a resync raises blockchain data requests, these do not ask for specific commitments or nullifiers. Hence, these offer less opportunity for inferring user interests.

## 2.5 | Dwarna's Dynamic Consent Solution

This dissertation redesigns the consent solution proposed in Dwarna (Mamo et al., 2019). This section presents an in-depth technical description of Dwarna, providing the necessary background for comparing the two designs.

### 2.5.1 | Problem Definition

Dwarna (Mamo et al., 2019) describes a workflow where a biobank intermediates between researchers and research partners. On one hand researchers are seeking access to biospecimens and data for conducting their studies. On the other, research partners may be willing to participate in such studies.

Research partners are individuals making available their biospecimen and personal data for inclusion in research studies. They start-off their biobanking participation with

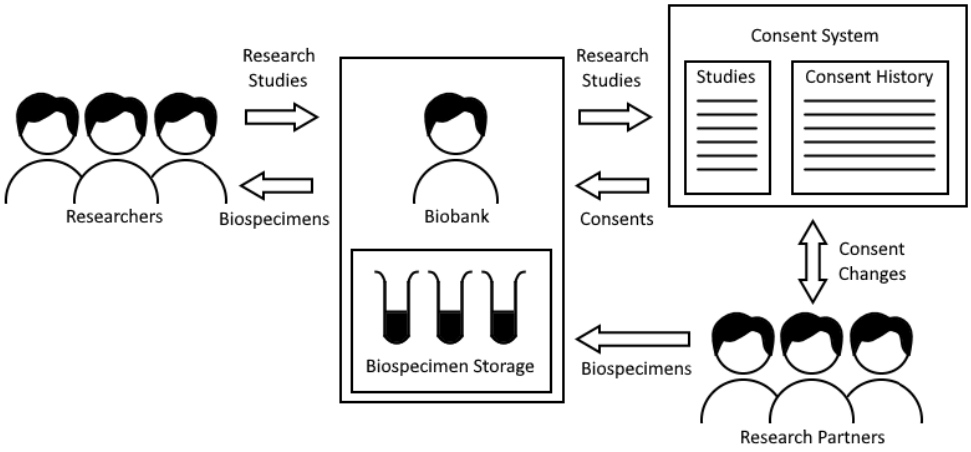


Figure 2.23: Participant Interaction in Dwarna

a visit to the biobank premises, providing the necessary biospecimen and data. Dwarna highlights how this is an on-going relationship that allows for data updates, which for example, may be required to reflect the partner's medical history.

The biobank thus becomes the trusted data custodian, responsible for securing personal data also with respect to GDPR regulatory obligations. Such regulation mandates that research partners retain control over their data, raising the need for a consent solution. This allows partners to efficiently dictate who should be granted/denied access to their data.

Apart for research partners, the biobank is also aliasing with researchers who are seeking data access for their studies. They provide the biobank with their research details. The biobank is then able to publish educational material, seeking the partners' educated consent. Next, given a set of consenting participants, the biobank provides researchers with access to their biospecimens and aggregated data.

Expressing consent is the key functionality enabled by the Dwarna platform. GDPR article 4 defines consent stating that this must be "freely given, specific, informed and unambiguous". Article 7 further states that data subjects retain the right to withdraw their consent. Thus, for a consent to be regulatory compliant, its workflow must satisfy all these properties.

Mamo et al. (2019) discuss different consent types, highlighting how these impact the efficient use of biospecimens and the relationship with participants. A broad consent applies a single consent to allow multiple studies, facilitating biospecimen reuse. However, broad consents risk to fall foul of the GDPR requirements for consents to be specific and informed. Beyond that, lack of participant control can undermine trust and

hinder enrolment. Potential participants may fear they might be contributing to research about which they have ethical reservations. At the other extreme, a one-time research specific consent accurately reflects the participant's intent. However, it is inefficient as it obstructs biospecimen reuse.

Dynamic consent aims to achieve both biospecimen reuse and to accurately reflect the participants' choices. For each study, participants choose whether to opt-in, whilst retaining the freedom to revert their choice. Participants may also choose to discontinue their biobanking relation, and for their data to be destroyed. This engagement level encourages participants to consider themselves as partners playing a pivotal role in advancing scientific research.

Figure 2.23 illustrates participant interaction focusing on the data flow. It shows how the consent system covers interactions between the biobank and research partners. The biobank is then tasked with publishing studies and providing biospecimens and data to researchers.

### 2.5.2 | Consent Platform Architecture

This section focuses on the technical implementation of Dwarna's dynamic consent platform.

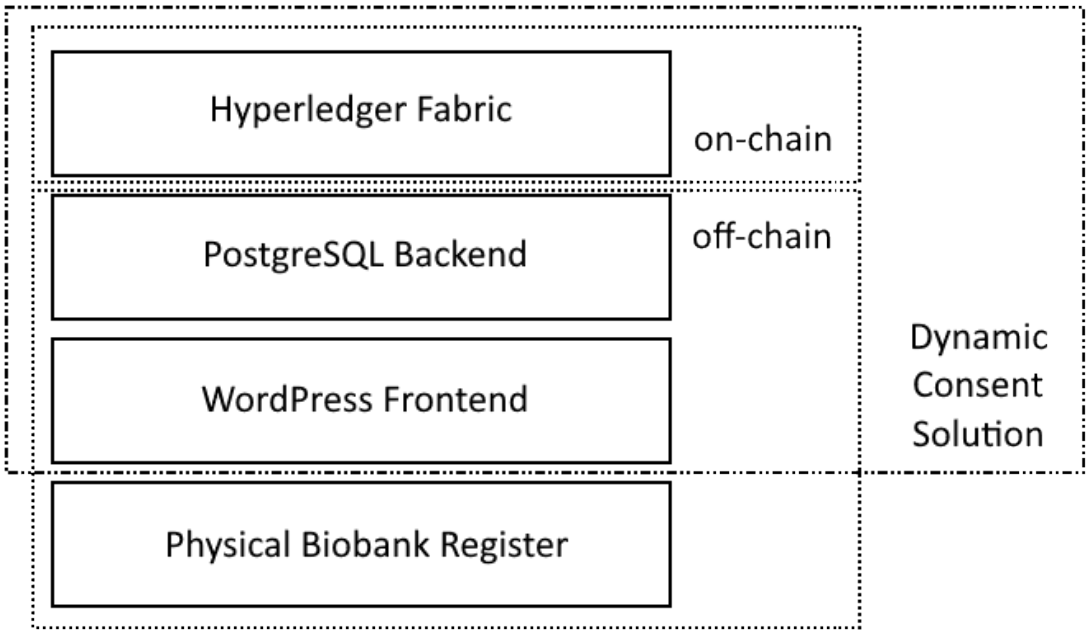


Figure 2.24: Dwarna's Consent Solution Layered Architecture

The system is composed of four layers, depicted in figure 2.24. The layers are shown

grouped based on two criteria. The first separates on-chain and off-chain layers. It thus separates the layer whose storage is immutable from those where data can be deleted. The second grouping identifies the layers through which consents are expressed and recorded.

The **Physical Biobank Register** stores biospecimens, private data such as medical histories, and the research partners' real identities.

The **WordPress Frontend** provides the interface for research partners to interact with the consent system. Dwarna describes this as the client that "makes all requests on behalf of its users". Partner data stored here includes a pseudonymous username, and encrypted contact details (email, first name and last name). The pseudonymous username is the first element decoupling partner data at the physical biobank register from the data at the immutable blockchain.

The **PostgreSQL Backend** provides a second identity decoupling layer, critical in achieving Dwarna's privacy goals. This layer brings together study identifiers and research partner identities. However, the username used at the frontend is now mapped to identities composed of:

- User credentials for accessing the permissioned Hyperledger Fabric blockchain.
- Unique Universal Identifiers (UUIDs) providing users with a unique identity for each research study.

The **Hyperledger Fabric** blockchain stores the immutable record of consent changes. A consent change is a timestamped record composed of a study id, a research partner's UUID and a Boolean choice between participating or withdrawing from a study.

Figure 2.25 depicts how a partner's identity propagates across the layers. At the biobank the partner's real identity is mapped to a pseudonymous username which is then mapped to UUIDs at the PosgreSQL backend.

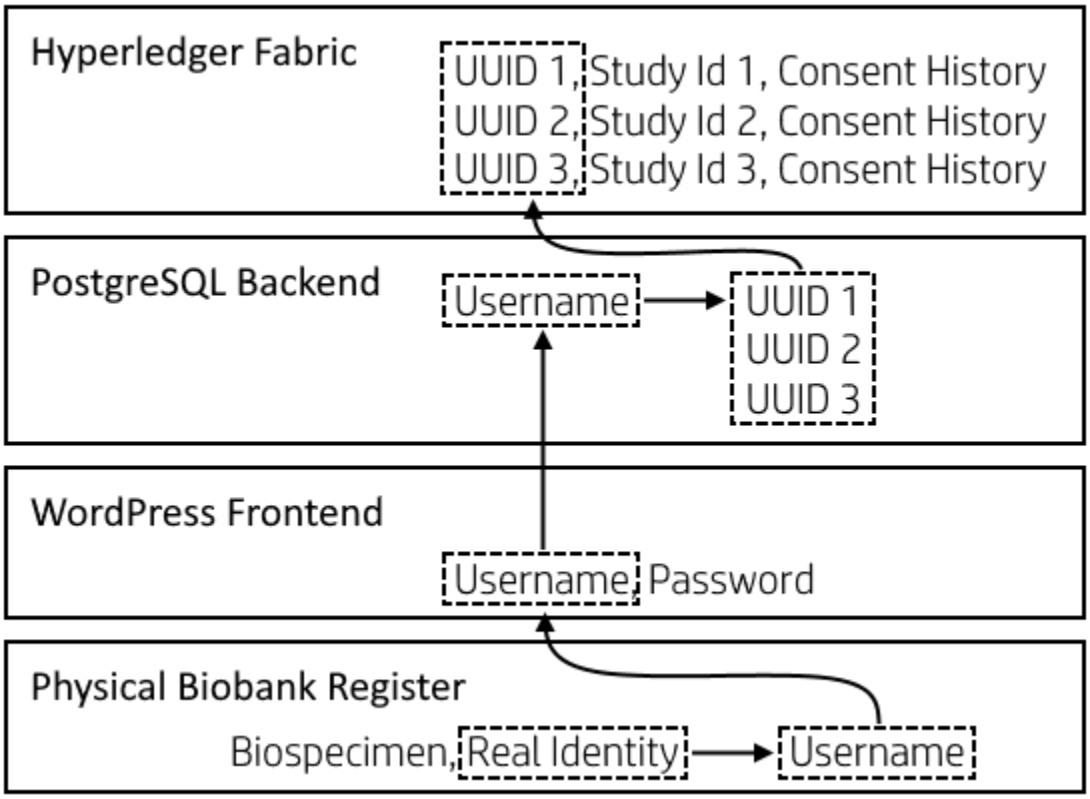


Figure 2.25: Decoupling partner data at the physical biobank register from the data held within the immutable blockchain layer.

### 2.5.3 | Privacy and GDPR Compliance

This section describes how Dwarna achieves its privacy goals and GDPR compliance.

Dwarna clearly defines the roles of the biobank, researchers, and research partners together with their data processing needs.

The biobank is the data custodian and privacy guarantor. The biobank manager is thus assigned the GDPR data controller role both for the biobank and the consent platform. Furthermore, the biobank is tasked with physically providing biospecimens and data to researchers.

Researchers fulfil the role of GDPR data processors whose purpose is to conduct well defined research studies. Dwarna provides researchers with the aggregated data of consenting partners, without identifying individual partners. This protects privacy and limits the researchers' regulatory burden.

Research partners are the data owners granting/withdrawing data access. Private access to their consent history is enabled through authentication and authorization within every layer, including the permissioned blockchain layer.

Consents are informed and specific. On signing-up, partners consent for the biobank to act as their data custodian. This consent does not grant access to researchers. Instead, through the consent platform, partners grant and withdraw study specific consents. The WordPress frontend integrates educational material for each study. Furthermore, on expressing a consent change, partners are required to complete a quiz confirming they understand their rights and what their consent entails.

When a partner discontinues study participation, the biobank stops including the partner's biospecimen and data in that study. A partner may also choose to terminate biobank participation altogether. Here the biobank fulfils the right to be forgotten by deleting all off-chain data. The identity decoupling mechanism illustrated in figure 2.25 ensures that the on-chain data becomes unusable on its own. Indeed, once all off-chain data is deleted, UUIDs cannot be mapped back to research partners.

## 2.6 | Permissioned vs Permissionless Blockchains

By redesigning Dwarna, this dissertation brings to the fore the contrast between permissioned and permissionless blockchains. Table 2.4 compares the key differences typical of these blockchain categories.

| **Property** | **Permissioned** | **Permissionless** |
|---|---|---|
| Decentralized Architecture | The pool of node operators is restricted to a group of trusted participants. A permissioned blockchain backed by a large consortium may achieve a high level of architectural decentralization. However, smaller permissioned blockchains may end up relying on just a few nodes becoming vulnerable to denial-of-service attacks. | Everyone is allowed to become a node operator. Everyone willing to invest in equipment, provide the necessary computational resources and possibly stake the required currency amount. If the benefits sufficiently outweigh the costs, the blockchain can attract thousands of node operators from across the globe. This gives significant resistance against denial-of-service attacks. |
| Decentralized Governance | Permissioned blockchains are often application specific with node operators having a direct interest in the application. Application and historical data availability relies on the on-going commitment of these node operators. Involving node operators with conflicting interests (e.g. regulators and competitors) improves governance decentralization. | Permissionless blockchains are application agnostic. The motivation to keep the blockchain and its ledger available is not application specific. Thus, permissionless blockchains provide a source of trust independent of application participants. |

| | | |
|---|---|---|
| Blocking Attacks | Being well known, the reputation of node operators plays an important role in incentivizing correct behavior. Misbehaving operators may be excluded from operating nodes. | The protocol aims to align incentives, such that the benefits of following the protocol honestly, outweigh the benefits of deviating from the protocol. Smaller permissionless blockchains relying on Proof of Work (PoW) have in the past struggled in blocking repeated attacks [11]. |
| Scalability and Transaction Latency | Relying on trusted node operators, the blockchain operates efficient consensus protocols achieving low latency and high scalability. | Consensus protocols must cater for the presence of malicious operators incurring a penalty in terms of scalability and transaction latency. Performance may also be impacted by the demand from larger unrelated applications running on the same blockchain. PoW blockchains are well known to suffer in this area, with Proof of Stake (PoS) performing much better. |
| Cost | May easily implement zero fee transactions. | Transaction fees may be very high and volatile. High costs challenge application feasibility. Volatility causes uncertainty around the long-term application feasibility. |

[11] Ethereum Classic Hit by Third 51% Attack in a Month [Accessed Aug 2022] - `https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/`

| | | |
|---|---|---|
| Privacy | Providing a permissioned access framework, applications can natively restrict blockchain read/write operations. Applications have a head start in restricting data visibility to legitimate owners and authorized data processors. | By default, all data is available for everyone to read/write. Data access restrictions must be developed on top of this open access platform, at application level. |

Table 2.4: Key Differences between Permissioned and Permissionless Blockchains

## 2.7 | Layer 1 and Layer 2 Blockchain Solutions

This dissertation presents a layer 1 blockchain solution. However, it also highlights areas where layer 2 components can deliver improvements. This section explains what constitutes layer 1 and layer 2 solutions.

Blockchains aim to provide a single source of truth through an immutable ledger. A layered model may then be defined based on the proximity of a given technology to this single source of truth.

Layer 1 technologies comprise the lowest layer, upon which other technologies depend. These are responsible for providing the immutable ledger and the core security guarantees. Bitcoin, Ethereum, and Cardano are examples of layer 1 blockchains.

Smart contracts running on layer 1 blockchains directly persist their state to the immutable ledger. Thus, these also form part of layer 1. The smart contracts designed in this dissertation run directly on Ethereum, composing a layer 1 solution.

Layer 2 technologies sit on top of layer 1 blockchains to improve or add functionality. Amongst others, improvements may target scalability, lowering fees and enhancing privacy. The Lightning Network is a layer 2 solution for Bitcoin. On Ethereum, layer 2 services based on different rollup techniques are prevalent.

Layer 2 services introduce an intermediate processing stage to which transactions are submitted. One common technique involves aggregating multiple transactions into a single layer 1 transaction. This reduces the amount of data saved to the layer 1 ledger, reducing the transaction fees for individual users.

Layer 2 services often achieve their goals by trading-off some benefits of directly transacting on layer 1. For example, aggregation may delay the recording of transactions

onto layer 1, increasing the transaction confirmation time. In addition, layer 2 services normally degrade decentralization as its preprocessing stage is less decentralized than the layer 1 blockchain.

## 2.8 | Summary

This chapter discusses various background topics. It starts by putting privacy preservation into perspective with a discussion around privacy attack vectors. This highlights how privacy is often diluted through data leaks that may seem harmless. However, attackers can link such leaks together amplifying the deanonymization effectiveness.

Next the discussion focuses on privacy preserving technologies and their application in permissionless blockchains. This leads to a discussion on Zero Knowledge Proofs, mixers, and private tokens. A specific mixer implementation is used as an example, Zeth. This mixer is important as it also provides the implementation starting point for the dissertation case study.

This chapter next focuses on the case study being tackled, Dwarna's dynamic consent solution. The discussion delves into the details of how Dwarna works and achieves its privacy goals. This is next complemented with a comparison between permissioned and permissionless blockchains, a key difference between Dwarna and the solution presented here.

Finally, the chapter discusses blockchain solution layering, a topic highly relevant to the presented solution.

# 3

# Consent Solution Design

To investigate the challenges of decentralized privacy, this chapter presents an alternative design to the real-world problem tackled by Mamo et al. (2019), described in section 2.5. The discussion starts from the key pillars of decentralization and privacy. Next the design for a permissionless dynamic consent solution is presented, together with the various considerations that shaped it.

## 3.1 | Trust Decentralization

The central role played by the biobank creates a power imbalance. Dwarna presents a consent solution running on top of a centralized permissioned blockchain. Centralized architectures often consolidate this imbalance by encouraging a top-down paradigm. The service provider deploys applications, caries out administrative duties, ensures service availability, and leads workflow execution. Such duties consolidate trust onto the service provider.

This dissertation explores solving Dwarna's problem in the decentralized permissionless context, binding itself with a key paradigm choice from the outset. Dwarna provides a starting point to explore the broader challenge of decentralized privacy. The discussion next looks at different decentralization approaches.

### 3.1.1 | Disintermediation

One possible decentralization approach is disintermediating the relationship between researchers and research partners.

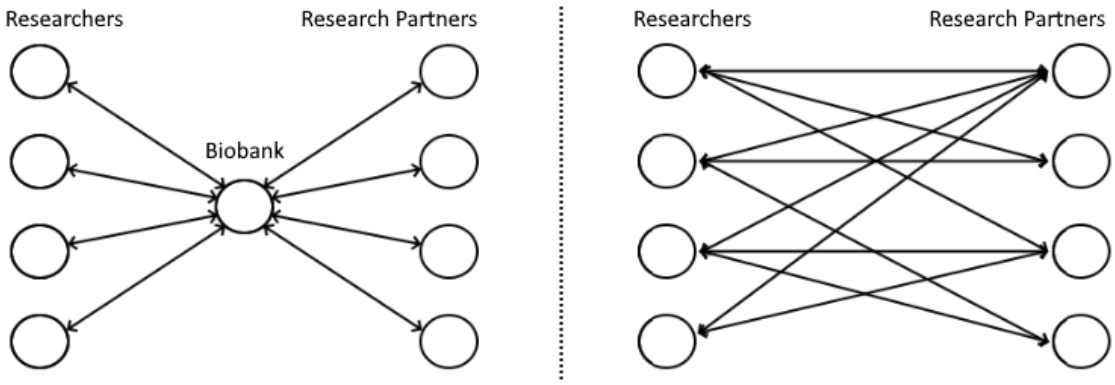


Figure 3.1: Intermediated vs Disintermediated

Figure 3.1 depicts intermediated and disintermediated communication side-by-side. In Dwarna, the biobank sits in the middle of all data exchanges. In the disintermediated alternative, researchers directly interact with research partners.

Research partners could possibly save their data in a consistent digital format. They would then require a platform for granting access to individual researchers. Such a solution would require anonymizing the data handed to researchers. Otherwise, the risk of leaks increases for each research subscription.

Upon consulting with researchers from Dwarna, it was established that disintermediation would not work in practice. A biobank is equipped to correctly store biospecimens (e.g., blood samples) and can effectively redistribute these to researchers. Research partners would not be able to do the same on their own. They would end up having to repeatedly provide fresh biospecimens to satisfy different research studies, something that makes the setup impractical.

### 3.1.2 | Multiple Poles of Trust

More decentralization opportunities arise from the involvement of additional parties. Two approaches are considered:

**Role Decomposition** - A centralized role is broken into multiple roles and assigned to independent parties.

**Oversight** - A centralized role is coupled with independent oversight improving monitoring and redress guarantees.

#### 3.1.2.1 | Role Decomposition

The discussion established the biobank as the real-world asset custodian. Beyond that, the biobank is also providing the consent solution, handling administration, and ensuring its availability. These roles are also relevant to decentralization.

Solution development can be open-sourced or closed-sourced, where the former allows code-level transparency. Dwarna is open-source[1], thus this trust decentralization element is already present and should be carried forward.

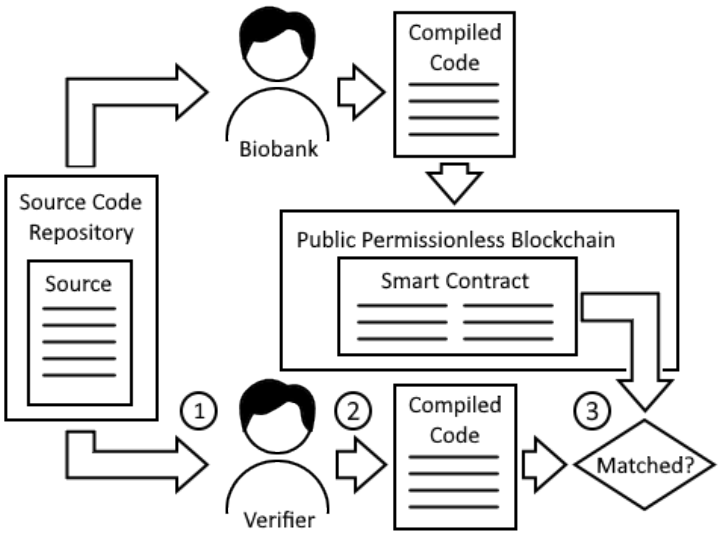


Figure 3.2: Comparing the smart contract source to the on-chain code.

In terms of deployment, administration, and service availability, Dwarna relies on the biobank. This is accomplished through a web application and a permissioned private blockchain, over which the biobank retains control. Here more decentralization is possible.

The combination of open-source and public blockchains allow for transparent deployment. Figure 3.2 illustrates deployment verification. Anyone could pull the source code (1), compile it (2), and compare the bytecode against the one deployed (3). Etherscan provides such a verification service[2], relieving users from this task.

This code transparency propagates throughout. The rules of engagement for different parties are written within the smart contract code. Even if a privileged role cannot be avoided, its range of activities are publicly defined. The smart contracts further determine whether these rules can be changed. If they do, the change is again a publicly

[1]Dwarna Github [Accessed Jun 2021] - `https://github.com/NicholasMamo/dwarna`
[2]Ethereum Verified Contracts [Accessed Aug 2021] - `https://etherscan.io/contractsVerified`

verifiable event. Thus, the public blockchain application hosting environment, provides an independent source of trust.

Lastly, the decentralized blockchain reduces the biobank's service availability concerns. The consent system is handed over to thousands of independent node operators, that have no stake in biobanking. Furthermore, they provide an immutable transaction history record, allowing retracing participant actions. Immutable records are also available on private blockchains. However public blockchains provide greater redundancy, independent repository ownership and an open access guarantee. No workflow participant controls such properties.

#### 3.1.2.2 | Oversight

Dwarna operates within a regulated environment and caters for satisfying GDPR compliance. Thus, external oversight already exists. Integrating oversight into the workflow provides another trust decentralization opportunity. This is embodied by a new workflow participant, an auditor. However, oversight requires information access, that in turn raises privacy concerns. So, a balance is required. At one extreme, no oversight is included, but private data only needs securing at the biobank. At the other extreme, the auditor has full data access. Oversight is at its highest, but so is the level of data disclosure.

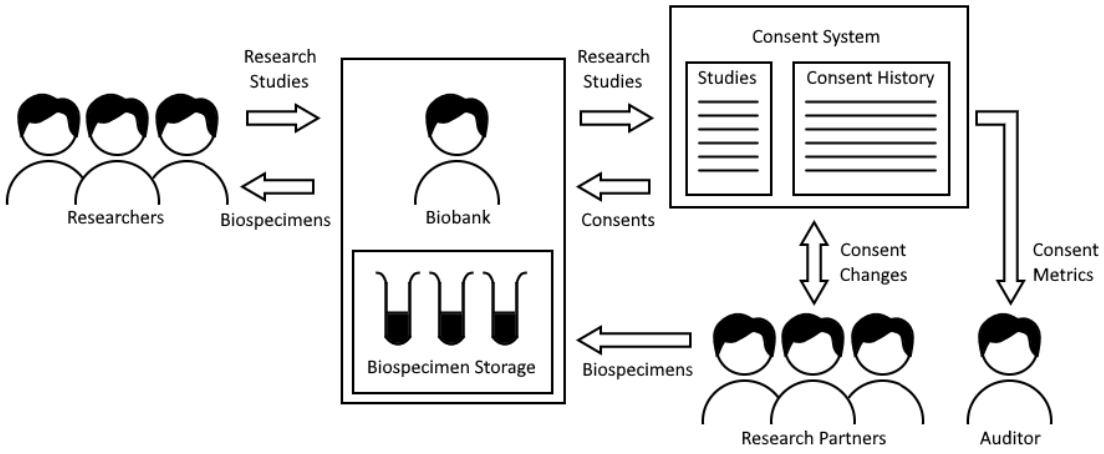


Figure 3.3: Auditor is granted read access to some consent system data.

Figure 3.3 extends Dwarna's setup to include an auditor, having read-only access to some of the system data. In the next section a model favouring limited data access is described. However, it is recognized that a different balance may be preferable. Thus,

a new design goal is added, that of allowing for flexibility in terms of the information provided to the auditor.

#### 3.1.2.3 | Finance based Auditing Model

Biobanks provide a critical service to the research community. However, their services cannot be easily commercialized without favouring revenue generating research. Research that is beneficial to the society, but with little profit-making potential, risks being less attractive. The non-profit model can also help engaging broad participation, from all strata of society.

Thus, the biobank is regarded to be a public service requiring external financing, raising additional accountability requirements. However, such requirements may conflict with a privacy preserving solution striving to obfuscate its usage.

For this reason, an auditor is given access to some of the consent system data. The goal is to provide enough information for the auditor to measure the biobank performance whilst minimizing access to sensitive data. Specifically, this allows financing the biobank based on the number of consents research projects are attracting. Two pieces of information are provided:

1. The number of participants currently signed up with the biobank.
2. The number of research projects to which research partners are currently consenting participation.

This is just one possible auditing model. The design later demonstrates how audits with a different focus could also be adopted.

#### 3.1.2.4 | Oversight - Immutable Consent Records

Apart for integrating an active auditor role, the immutable consent trail provides an opportunity to investigate past actions. The blockchain provides timestamped receipts for the actions taken by the biobank and research partners. An investigator having access to the biobank's backend, could then perform cross-verification against these receipts. In this manner, such receipts also decentralize trust. An example of such an investigation is described in section 3.7.

### 3.1.3 | User Empowerment

User Empowerment is another form of decentralization. The steps required for completing an operation can be organized to empower one party over another. Decentralized

solutions favour users over service providers. Thus, the aim is to empower research partners over the biobank. The design discussion provides practical examples of this decentralization vector. Such workflows give research partners confidence in the system.

## 3.2 | Privacy

Section 3.1 explored the first dissertation pillar, decentralization. The second pillar is privacy. Figure 3.3 depicted four roles, out of which three are interacting with the consent solution. Thus, the solution must cater for their distinct privacy expectations.

### 3.2.1 | Privacy Requirements

The biobank is the system promoter, the real-world asset custodian, and the one carrying the regulatory burden. This role does not afford anonymity, it rather aims for the highest level of transparency.

The auditor requires read-only access to the system and raises no privacy claims of his/her own. The only concern is allowing auditability without compromising the privacy of others.

The research partner role is hopefully fulfilled by many participants who must be assured that their data is handled with full respect for their privacy. Apart for the data handed on sign-up, the consent choices also need obfuscating. Such choices, for example might reflect personal ethical views. Thus, the highest level of privacy is required.

### 3.2.2 | Privacy Technology Choices

Different privacy expectations call for some key technology choices. A permissioned solution naturally lends itself to model these roles in terms of access rights. The decentralized context removes the rigidity of a ready-made framework, allowing for design freedom. The biobank is prioritizing transparency whilst research partners are expecting privacy. Both extremes are being implemented on decentralized blockchains.

Users who are upfront about their identity, like the biobank, can be catered for using the most common design pattern. Here, users access smart contracts by presenting their publicly known blockchain identity. Smart contracts verify whether the transaction sender identity matches the biobank identity. If it does, access to the biobank's functionality is granted.

For those requiring the highest level of privacy, reference is made to private tokens. As discussed in section 2.4, these could represent anything from a cryptocurrency to an identity, to a consent choice. Thus, the dissertation adopts the private token design patterns as implemented in Ethereum.

## 3.3 | Decentralized Dynamic Consent

So far, the solution requirements were presented together with the core decentralization and privacy pillars. Within academic literature, no blockchain-based permissionless consent solution that could readily cater for Dwarna's needs was identified. Thus, the following sections present an alternative design demonstrating how Dwarna could be implemented in the permissionless setting, answering research question 1.5.

### 3.3.1 | Research Partner Sign-up

In the decentralized solution, the biobank is still the data custodian. Hence research partners visit the biobank to provide their biospecimen and personal data.

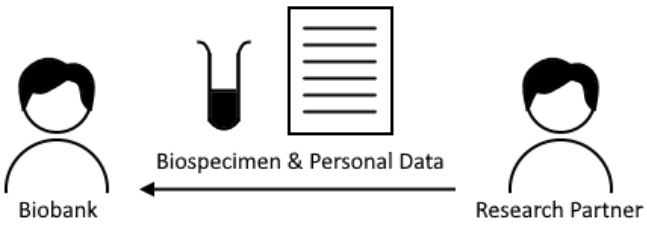


Figure 3.4: Research partner hands biospecimen and data to biobank.

Next, partners establish their consent solution identity. In centralized solutions, identities are often created by the service provider and assigned to users. In decentralized solutions, users create their identity independently. However, partner identities need to be linked to the data held at the biobank. Thus, the biobank must also provide confirmation that the identity owner truly signed-up.

Section 2.4.3.6, lists the identity schemes employed in private token solutions. These are integrated into the workflow as shown in figure 3.5. Research partners install a wallet-like application (1) that generates two key pairs, one for message encryption, and another for token ownership. The public halves are handed to the biobank (2). Thus, the biobank maps biobanking data to public keys (3). All of this happens off-chain.

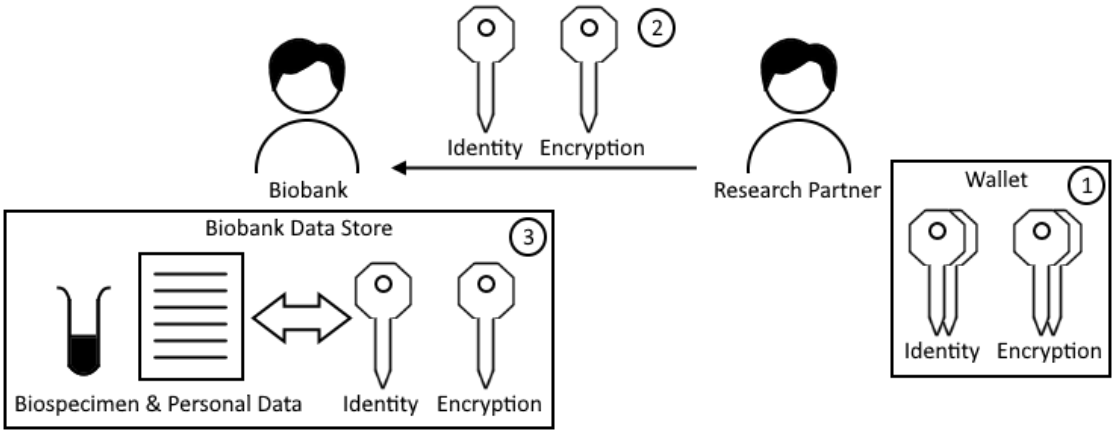


Figure 3.5: Research partner generates identity and encryption key pairs, handing the public keys to the biobank.

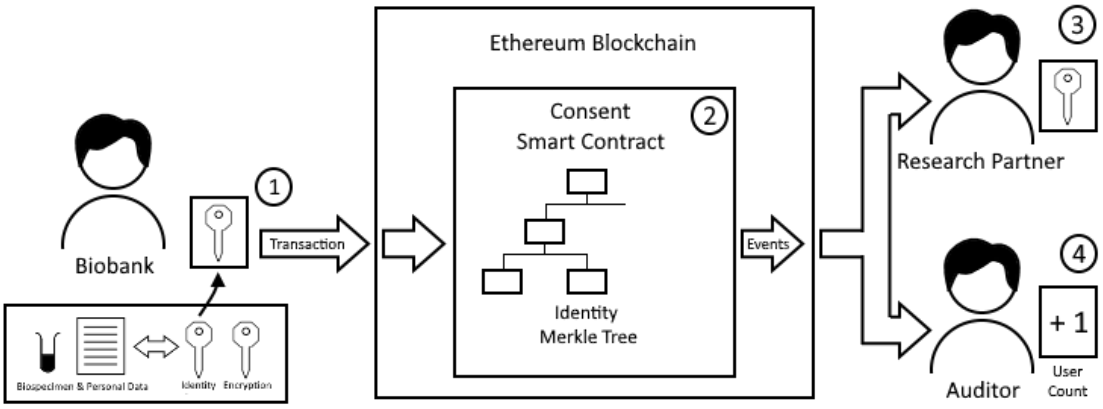


Figure 3.6: Biobank mints an identity token for a research partner.

The biobank's endorsement comes next, shown in figure 3.6. The biobank mints a private identity token encapsulating the partner's public identity key (1). This is similar to a ZethNote but is only required for its ownership property. The token commitment is then published on-chain (2).

The biobank also needs to encrypt and send the token cleartext to the new research partner. As for Zeth, event-based cipher propagation is used. Thus, partners discover their own identity token (3).

Also listening, is the Auditor, who observes the minting of a new identity token and updates the research partner count (4). No privileged access is required. Minting is a publicly observable operation, and the auditor exploits this information leak.

Given the identity token, a research partner is ready to interact with the consent system. Before that, some research studies must be added.

### 3.3.2 | Study Sign-up

The choice of studies is publicly known, as biobanks run educational campaigns encouraging partner enrolment. Thus, study obfuscation is unnecessary. What needs protecting, are the study choices research partners make. Proof-of-membership (section 2.4.4.3) provides such obfuscation also when applied to public data sets.

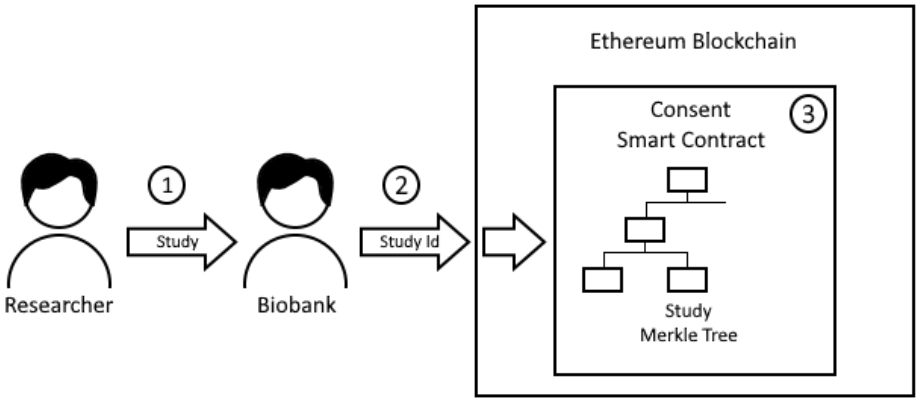


Figure 3.7: Biobank publishes new study identity.

Figure 3.7 shows a researcher passing study information to the biobank (1). The study is assigned a unique study identifier (2) and is published on-chain (3).

### 3.3.3 | Consenting to Studies

Consents express preferences between joining or leaving studies. Intuitively, these bring together a partner, a study, and a Boolean choice. The solution already represents identities and studies on-chain. Now, an object joining them to a choice is required. Choices are private; hence another private token is introduced.

Tokens are created through minting. The biobank could mint and distribute consent tokens to research partners. However, this would replicate a centralized model, where services are provisioned from the top. A decentralized alternative is the one leaving minting to research partners.

The biobank minting model requires creating tokens for all users. Having many tokens produces a large anonymity set. Yet, as an opt-in solution, these would all have a

well-known initial negated choice. Adding the fact that minting is a publicly observable event, these would be inferior in privacy terms.

Research partners can mint and immediately express any initial choice. One could argue that research partners are more likely to mint tokens in the affirmative state. However, this could be countered by a client application that encourages minting multiple tokens. One token would express a choice for the study of interest, whilst a second study is picked to mint a token denying consent. This becomes more feasible if research partners are relieved from paying for transaction fees. Hence, apart for being more decentralized, self-minting also has valuable privacy attributes.

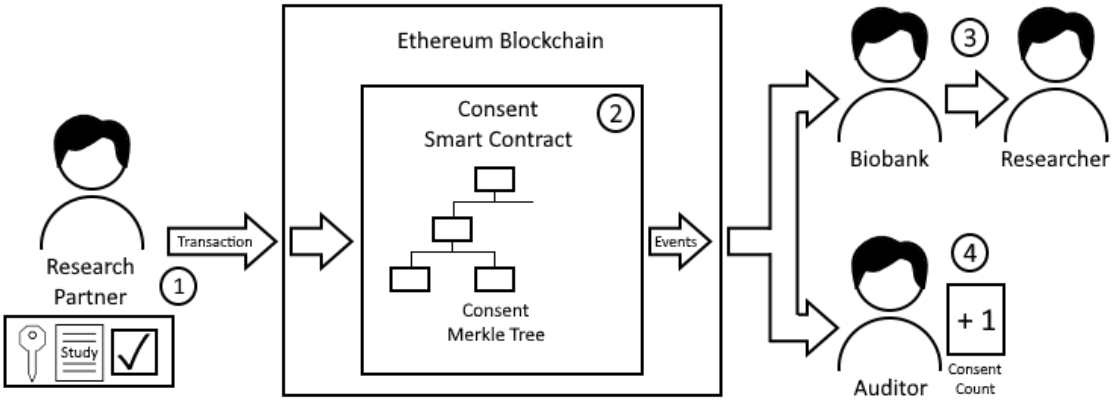


Figure 3.8: Research partner mints a consent token, expressing an initial in/out choice.

Figure 3.8 illustrates consent minting. Research partners construct tokens (1) and publish their commitment (2). Being self-minted, partners retain token ownership, hence retaining the ability to revert their choice. The biobank and auditor are both interested in this consent. The biobank needs to fulfil (3) the partner's choice, whilst the auditor is counting study subscriptions (4). This is discussed in the subsection that follows.

#### 3.3.3.1 | Consent Processing

Consent tokens are hidden behind opaque commitments. Thus, the biobank and auditor need research partners to disclose the consent details through the encrypted channel.

The biobank needs the complete raw token details. This will not convey ownership since that is determined by the identity key. However, it allows identifying the partner, fulfilling the consent choice, and blocking malicious consents.

The auditing model mandates measuring study participation levels. It would be ideal if this could be obtained whilst only the choice flag is disclosed. The auditor would

not know who submitted the consent, or to which study it pertains. For this to work, the solution must guarantee that:

1. Only legitimate research partners are submitting consents.

2. All consents pertain to on-going studies.

Such guarantees can be enforced within the ZKPs.

The auditor now can increment the count whenever a mint grants study participation. On the other hand, the auditor ignores a mint negating participation. As an opt-in workflow, when a new study is launched, all research partners are initially negating participation. Minting starts a partner's interaction with a study. Thus, a mint negating participation simply confirms the initial status, causing no change in the count of active consents.

Even though the auditor is not provided with the partner's identity key, identity information may still leak. In figure 3.8, steps (2) to (4) are atomic. The auditor sees which transaction raised the cipher propagation. If in (2) partners leak their Ethereum address, choice flags could be mapped to addresses. Such a leak could be avoided through a layer 2 solution hiding sender addresses.

#### 3.3.3.2 | Limiting Consent Token Minting

Research partner choices cannot be ambiguous. For a given partner and study, there can only be one valid consent token. This restriction is enforced by limiting minting to a single token per study per partner.

The consent solution achieves this by requiring partners to provide a nullifier composed of the partner's private identity key and the study identifier. The smart contract verifies that each minting transaction provides a unique nullifier, blocking double minting.

### 3.3.4 | Changing Consent

Research partners retain the right to revert their choices. Naturally the last expressed choice should override any preceding choices. Blockchain transactions are processed sequentially and timestamped. Thus, sequencing is built-in and verifiable.

This design considers two schemes, an alternating sequence, and a free sequence. An alternating sequence requires the choice flag to invert the previously expressed choice.

A choice denying study participation can only be followed by a choice granting participation and vice versa. A free sequence does not impose any pattern. Partners are free to express the same choice repeatedly.

The free sequence is more unpredictable. If someone discovers that a research partner is participating in a study, one cannot be sure what a subsequent consent change would express. One could argue that partners would not bother repeating the same choices. However, a client application could encourage such a practice, when combined with feeless transactions.

The alternating sequence is simpler to implement. A single choice flag discloses both the new choice and the choice being replaced. Thus, the auditor immediately knows whether the consent count needs incrementing or decrementing. In comparison, the free sequence requires more information. This dissertation implements the alternating sequence. However, the free sequence is recognized as a potential improvement.

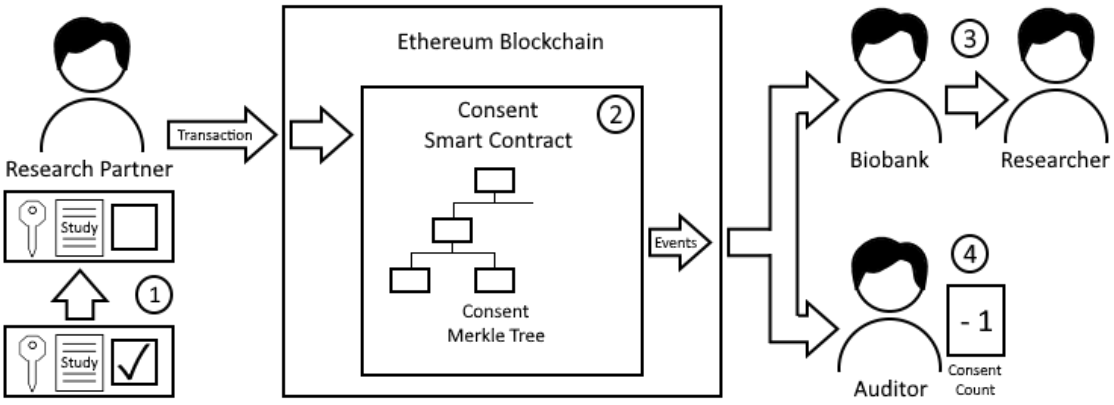


Figure 3.9: Research partner inverts the current study participation choice.

Figure 3.9 illustrates a consent change for withdrawing study consent. A research partner fetches the current consent token and constructs a new token with the flag inverted (1). Its commitment is written on-chain (2). Furthermore, the biobank and auditor are provided with the consent details through the encrypted channel. The biobank fulfils the consent choice (3). The auditor increments the consent count when the flag is set and decrements it when cleared (4).

#### 3.3.4.1 | Limiting Consent Token Ownership

Consent change must also preclude choice ambiguity. Following a consent change, the rule limiting supply to a single token per partner per study should still hold.

To satisfy this requirement a consent change is composed from two operations, executed atomically. The first burns the old consent token, whilst the second creates a new one. This sequence is identical to a private token transfer.

Even though this operation replicates a transfer, the research partner cannot enjoy complete freedom. Both old and new tokens must have the same owner and study. Allowing a change in any of these, would allow partners to have multiple tokens for the same study.

These limitations render consent tokens more predictable. Their privacy is inferior to private tokens that allow ownership transfer. The consent application is clearly operating within stricter requirements, necessitating such trade-offs. Nevertheless, handling tokens in zero knowledge still offers strong obfuscation. One still cannot link the old and new tokens together. Thus, the information source from which the new token properties could be inferred remains hidden.

### 3.3.5 | Consent Verification and Confirmation

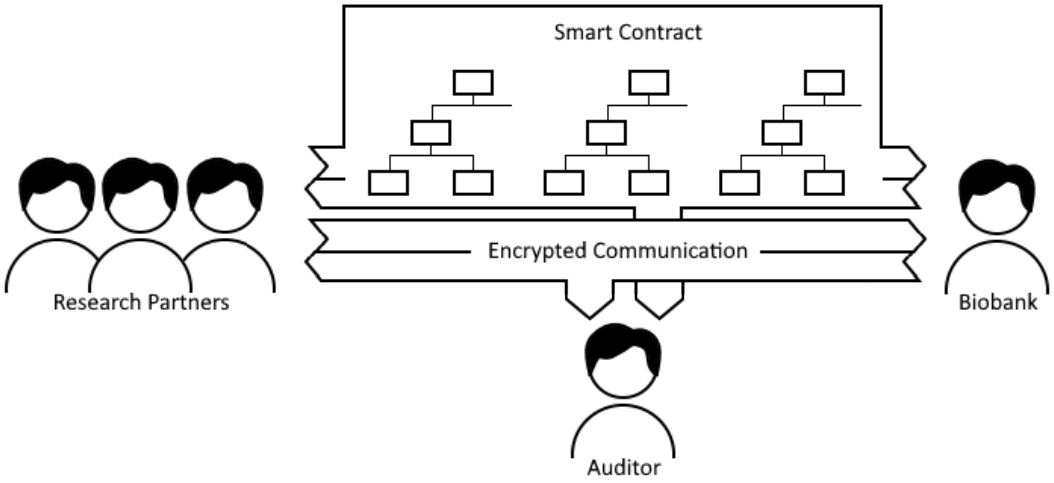


Figure 3.10: Independent Communication and Storage

Section 2.4.3.5 describes how recipients verify received tokens. This need arises from having two independent channels communicating/storing related information.

Figure 3.10 depicts the independent channels within the consent solution. Research partners and the biobank are reading/writing on both channels, whilst the auditor enjoys read-only access. This raises a data consistency challenge. What happens if a token whose commitment is written on-chain is not consistent with the details sent over the encrypted channel?

This problem is relevant to both consent minting and changing. A research partner might mint a consent token for study A and over the encrypted channel communicates that this was minted for study B. Even worse, if a research partner discovers the public identity key of another partner, an impersonation attack could be attempted. The attacker would follow all the on-chain rules, but at the encrypted channel the token identity key is swapped with that of the attack target. Unless verified, the biobank ends up executing a choice for the wrong partner.

This is especially tricky when the communication originates from research partners. The biobank also writes over the encrypted channel on partner sign-up. However, since its identity is well known, such abuse is easily dealt with, maybe by reporting it to the auditor. In comparison, research partners could abuse the system anonymously. Attempting to tackle this problem by deanonymizing the offender is not an option. Such an approach would undermine the privacy promise of the same application.

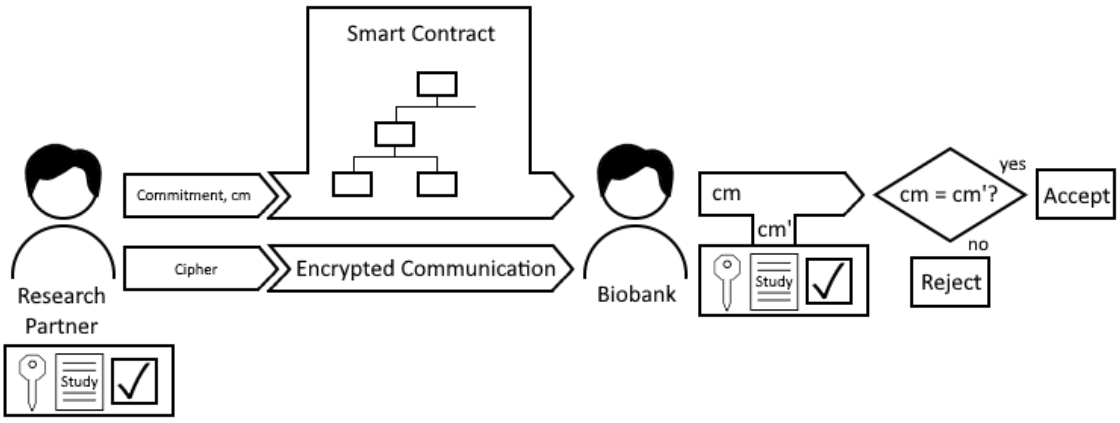


Figure 3.11: Biobank verifies a consent token commitment.

Solving the most serious issue involves replicating the commitment verification described for private tokens, as shown in figure 3.11. On minting or changing consents, partners disclose the token cleartext to the biobank. The biobank recomputes the commitment, verifying that it matches the one written on-chain. Only if this succeeds would the consent be fulfilled.

This verification is necessary, but the blockchain now lacks a definitive record of whether the consent was fulfilled. Thus, the biobank is further required to record the verification outcome on-chain as shown in figure 3.12. This forces the biobank to prove knowledge of the consent commitment opening. To research partners this is a private processing receipt. If a legitimate consent is not confirmed, one could report the biobank and possibly terminate participation.

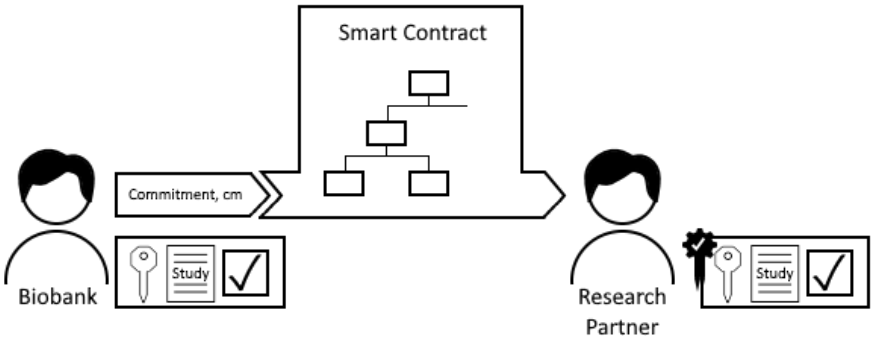


Figure 3.12: Biobank confirms consent validity on-chain.

#### 3.3.5.1 | Verified Consent Audit

Unlike the biobank, the auditor does not have the full token details. Hence the auditor cannot perform the same verification. Furthermore, one cannot require the biobank to voluntarily assist the auditor, as that would centralize trust on the biobank.

This dissertation proposes a solution based on Shamir's secret sharing (Shamir, 1979), a scheme that forces multiple parties to work together to discover a secret. Secrets are broken into shares and distributed across parties. The scheme defines a threshold k, determining the number of shares required for recomposing the secret. Even with k-1 shares, no information is disclosed.

Figure 3.13 illustrates how research partners construct and distribute shares whenever minting and changing consents. The scheme is applied twice, ensuring neither the biobank nor the auditor get any information, unless they cooperate. Starting from the top, a token is broken into two sets. A triplet is formed, leaving the choice flag on its own (1). The triplet is fed into the first secret generator (2) producing two shares, A1 and A2 (3). Share A2 is combined with the choice flag to form a new secret (4). This is fed into a second generator (5), producing shares B1 and B2 (6). Finally A1 and B1 are sent to the biobank (7), whilst B2 is sent to the auditor (8).

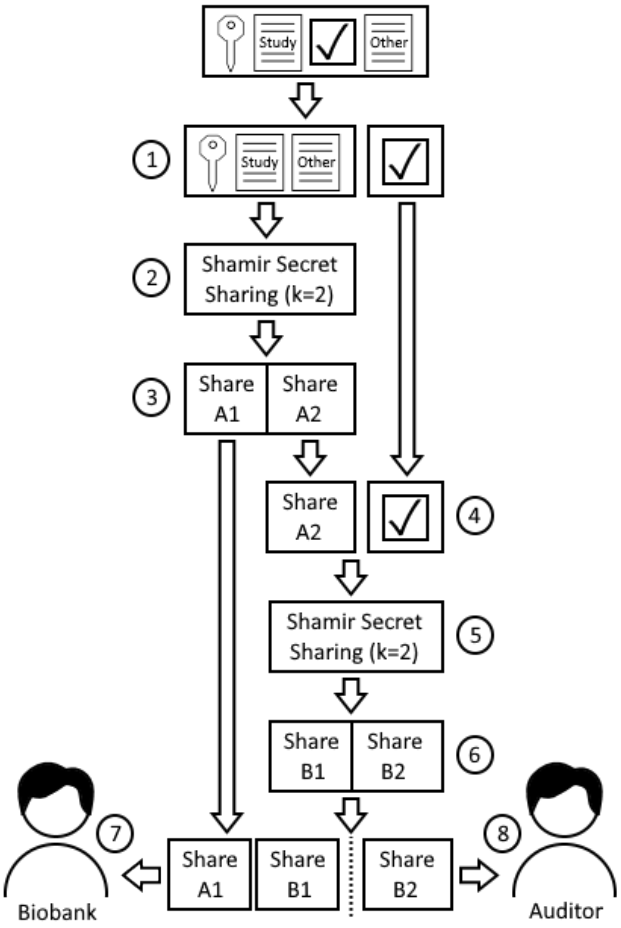


Figure 3.13: Shamir's Secrets Sharing - Splitting a token between biobank and auditor.

The biobank and auditor now have completely opaque shares. They engage together as illustrated in figure 3.14, stage 1. B1 and B2 are joined (1), discovering A2 and the choice flag (2). This is all the auditor needs, who cannot extract any additional information from A2. The biobank moves to the second stage independently. Joining A1 and A2 (3), the rest of the token data is discovered (4). The biobank may now proceed with token verification.

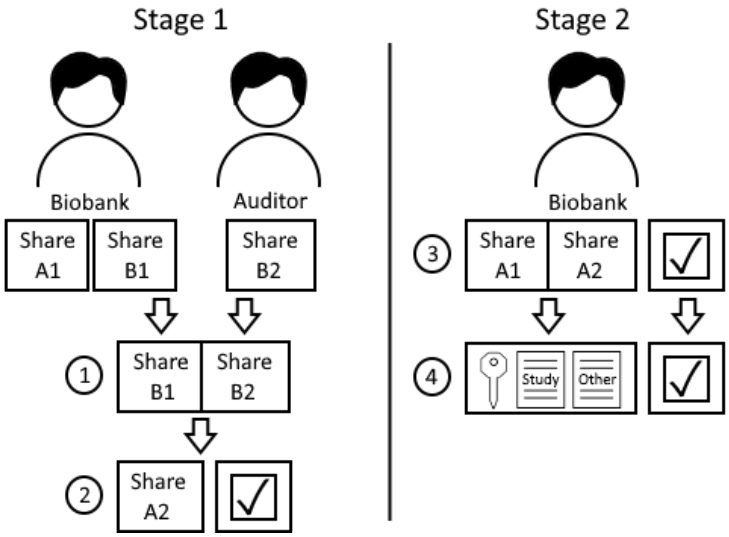


Figure 3.14: Shamir's Secrets Opening - Biobank and auditor recompose token.

Through Shamir's secrets, the biobank and auditor information become interdependent. Any common data disclosed to both parties is guaranteed to be identical. Thus, when the biobank confirms token validity, the auditor learns that his/her data pertains to a legitimate consent.

To close the circle, the auditor integrates the biobank's confirmation into consent counting. Figure 3.15 shows a partner submitting a consent (1) and secret discovery (2). However, the auditor does not immediately update the consent count. Instead, consents are recorded as a commitment-to-choice mapping (3) and waits for the biobank's confirmation.

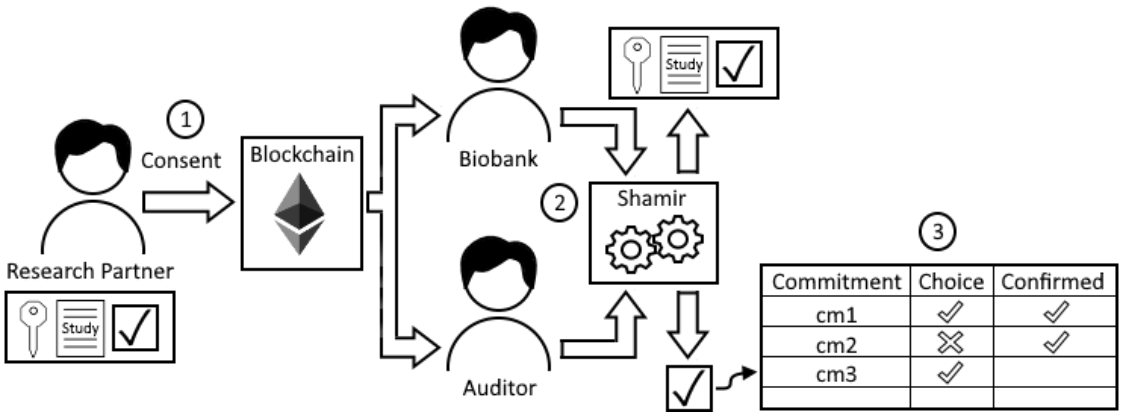


Figure 3.15: Auditor notes a consent change without updating the consent count.

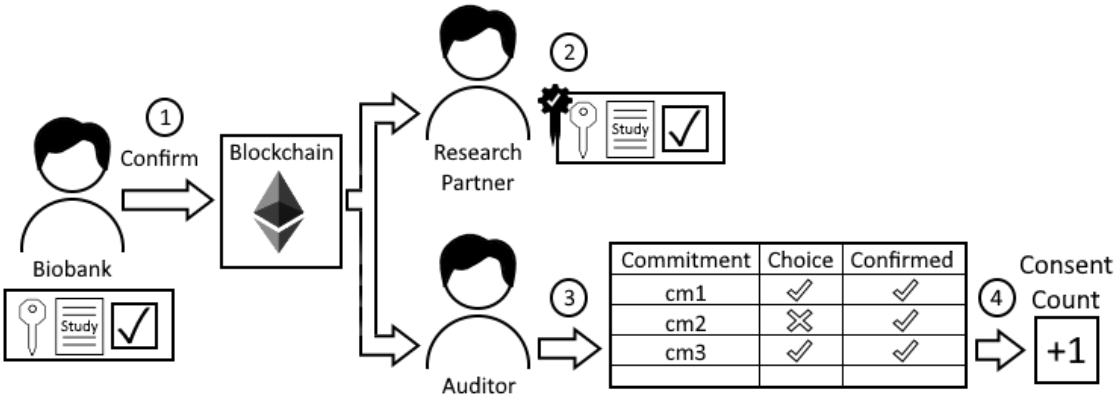


Figure 3.16: Auditor notes a consent confirmation updating the consent count.

Figure 3.16 shows the biobank submitting the consent confirmation (1), and its propagation. The research partner records his/her receipt (2). The auditor identifies the table entry with matching commitment (3), retrieves the choice flag, and updates the count (4).

#### 3.3.5.2 | Audit Scheme Flexibility

Section 3.1.2.2 identified audit scheme flexibility as a design goal. Changing the audit requirements, changes the information needs. This design can easily be modified to add/remove token data to the auditor's dataset by changing the secrets' composition. The first secret always hides data that is exclusive to the biobank. The second hides the shared data. In this way, no matter what the datasets include, the biobank's validation always confirms the auditor's data.

#### 3.3.5.3 | Consent Spam

Consent confirmations protect against malicious transactions. However, one can still create invalid consents. The problem may now be downgraded to spam, but it could still undermine solution feasibility.

In Zeth, one could also create ZethNotes that cannot be spent. Setting the wrong public key is enough. However, this is harmless since:

1. Receiver-side verification rejects such tokens (see section 2.4.3.5).

2. Spenders pay gas for submitting spam.

In comparison, the biobanking solution relies on altruistic participation. Feeless transactions become crucial, allowing spam at zero cost. This dissertation does not solve this issue, but possible solutions include:

1. Feeless transactions are achievable through layer 2 solutions, that relay and pay for transactions. One could consider circulating a private token within the relaying network. The biobank would top-up partners' token credit for every batch of valid consents. Thus, spammers run out of tokens, ending up paying for their spam.

2. Verifiable encryption, possibly the one proposed by Lee et al. (2019), provides another alternative. A verifiable encryption scheme ensures consistency between encryption inputs and ZKP proof inputs. For example, given a commitment opening proof, the prover can be bound to provide the same preimage to both cipher and proof generators. The ZKP verifier would then confirm such consistency. The solution would make away with Shamir's secrets. Instead, research partners would directly encrypt both biobank and auditor data using the verifiable encryption scheme.

### 3.3.6 | Binding Consents to Identities and Studies

Consents bring together a partner's identity, a study, and a choice. Additionally section 3.3.3.1 stated that consents should originate from legitimate partners and talk about on-going studies. This allows defining a consent token's lifespan in terms of its composing elements. Consent tokens should become unusable whenever:

- their owner terminates biobank participation.
- their study is discontinued.

Section 3.3.7 and section 3.3.9 will show how identity and study lifespans are aligned with their real-world counterparts. Meanwhile, this section describes how consents are tied to identities and studies.

In ZKP terms, among other things, partners are required to prove that:

1. There exists a valid on-chain identity token, matching the consent's owner.

2. There exists a valid on-chain study, matching the consent's study.

This is achieved by combining multiple proofs-of-membership, as shown in figure 3.17. The first identifies the consent token being spent. The second identifies an

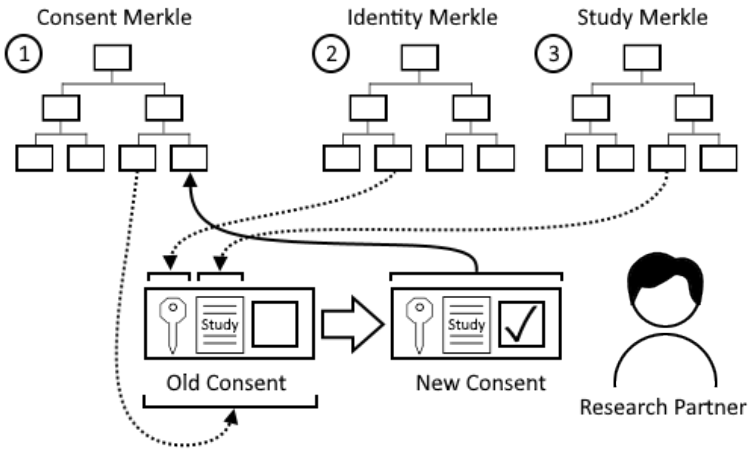


Figure 3.17: Constructing a consent change from 3 proofs of membership.

identity token matching the consent's owner. The third identifies the study for which the choice is being expressed.

#### 3.3.6.1 | Identity Refreshing

Section 2.4.3.4 highlights how spent commitments are not deleted from the data set. Furthermore, proof-of-membership does not distinguish between spent and unspent tokens. Thus, one must further prove that referenced tokens are unspent by publishing their nullifier.

The sequence that spends and creates a new consent token, shown in figure 3.17, allows for publishing the old token's nullifier. Thus, consent tokens are protected against double-spending. A similar sequence is also necessary for precluding identity token double-spending.

Figure 3.18 extends the construct for expressing a consent with an identity refresh sequence. Again, this is just a token transfer where both spender and recipient point to the same owner. However, the old identity token nullifier is now safely published, and the research partner ends up with a new token.

Identity refreshing also improves privacy. On sign-up the first identity token is minted by the biobank, who thus knows its cleartext. Expressing the first consent spends this identity token. Thereafter, partners work with self-created tokens, and subsequent consents continue morphing this identity.

A similar solution is not implemented for studies. These are not private tokens, and their lifespan is simply reflected by adding/removing Merkle tree entries. Thus, the tree only contains on-going studies. This is discussed further in section 3.3.9.

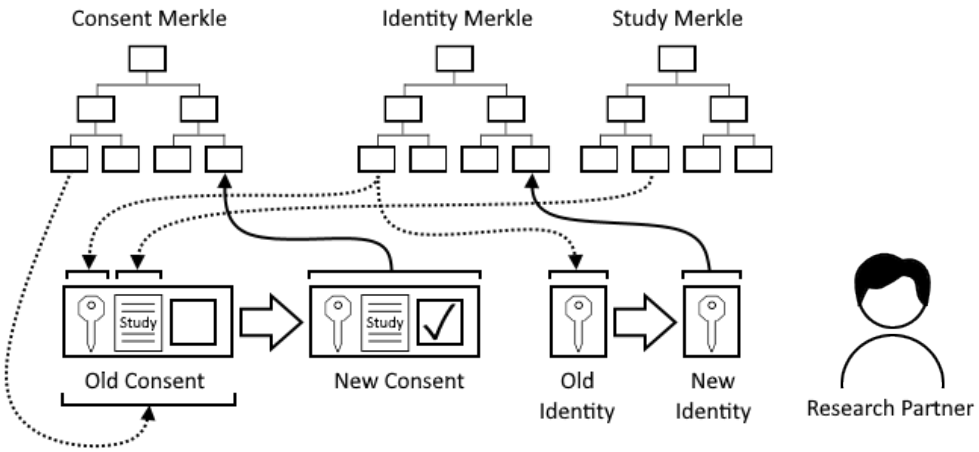


Figure 3.18: Identity Refresh - Creating a new identity token on changing consent.

### 3.3.7 | Research Partner Termination

When partners terminate biobank participation, Dwarna states that removing already-processed data from studies is usually not possible. Instead, termination causes the biobank to destroy the biospecimen and data, disallowing further use. Thus, on termination partners should no longer raise consents, as the biobank would be unable to fulfil them.

The decentralized paradigm aims to put partners in the driving seat. Ideally partners should drive the workflow, without any biobank dependency, whose only role is limited to fulfill the request.

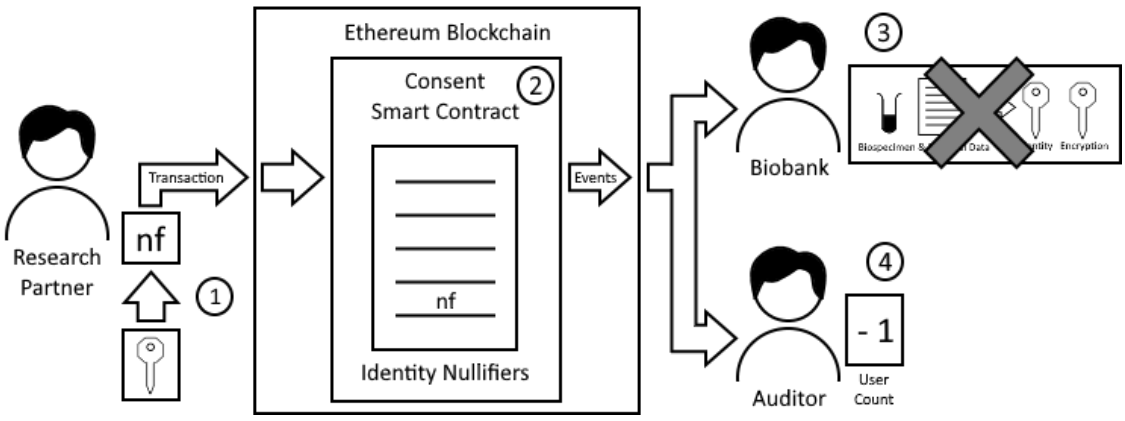


Figure 3.19: Partner Identity Burning

Burning the identity token as depicted in figure 3.19 takes us a long way. A part-

ner picks his/her identity token, computes its nullifier (1), and submits it to the smart contract. The nullifier is verified for double-spending and is persisted (2). Next, an event transfers the encrypted token to the biobank. The biobank identifies the partner and destroys locally held data (3). As a publicly observable event, burning also alerts the auditor to decrement its user count. Finally, the partner will have no unspent identity token, blocking further consents. Furthermore, the transaction itself provides an immutable receipt.

This would be enough if it were not for a potential impersonation attack, like the one described in section 3.3.5. This time, a partner could encrypt someone else's public key misleading the biobank to destroy the wrong data.

#### 3.3.7.1 | Verifiable Termination

The problem with simple identity burning is that the biobank cannot verify it. Unlike commitments, a nullifier can only be computed by the private key owner. Thus, a partner must provide something else, something that is verifiable.

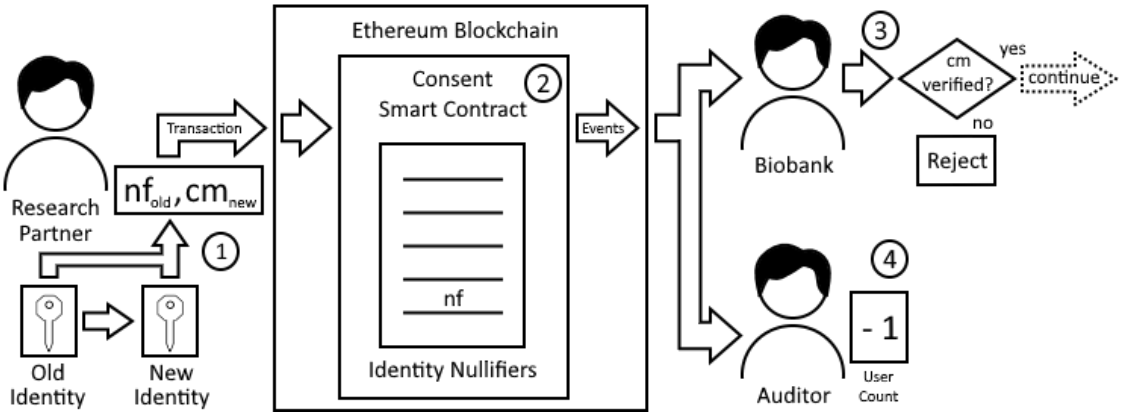


Figure 3.20: Partner generates a temporary identity token allowing biobank verification.

The identity refresh shown in figure 3.20 fills this gap. Partners compute the nullifier for their current identity and a commitment for a new identity (1). Everything is passed to the smart contract. However, only the nullifier is persisted (2). The new identity commitment is not saved and cannot be spent. The encrypted new identity token and its commitment are propagated through an event. The biobank is now able to verify the commitment (3). The transaction still leaks that someone is terminating participation and the auditor updates the user count (4).

Unlike consent verifications, the auditor does not integrate the biobank's identity verification. Even if verification fails, someone still lost the ability to submit further

consents.

### 3.3.8 | Confirmed Research Partner Termination

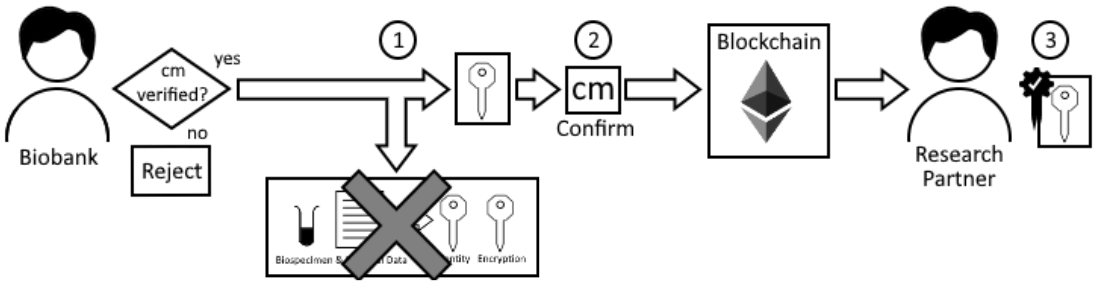


Figure 3.21: Biobank confirms partner termination and destroys off-chain data.

Termination confirmation is depicted in figure 3.21. Once the commitment is verified, the biobank deletes the partner's data (1). Next, termination confirmation is submitted, proving knowledge of the identity commitment opening (2). This confirmation provides partners with a termination fulfilment receipt (3).

### 3.3.9 | Study Termination

Sections 3.3.2 requires study choices to be private, whereas section 3.3.6 requires on-chain studies to mirror study duration. These are satisfied by an editable Merkle tree, to which studies are added/removed matching study duration.

The ZKP implementation for studies is simpler, no commitment/nullifiers are involved. However, editable Merkle trees do not benefit from the on-chain storage optimizations afforded by append-only trees[3]. One can expect faster proof generation, but higher on-chain storage costs.

Here, a more detailed analysis of gas costs is appropriate. Alternative designs are also worth considering, possibly based on proof-of-non-membership or sparse Merkle trees. Yet, time constraints limited this design to a naive editable Merkle tree solution.

Editable Merkle trees allow adding/removing entries as depicted in figure 3.22. Here, the first update discontinues study 9. Thereafter, if one could provide proof-of-membership against the initial tree, study 9 would be resurrected. This can be prevented by requiring proofs to reference the latest tree. However, such a restriction creates a race condition. For example, study 10 is present throughout. Yet, each tree update changes

[3]EY Nightfall Timber [Accessed Oct 2021] – `https://github.com/EYBlockchain/timber`

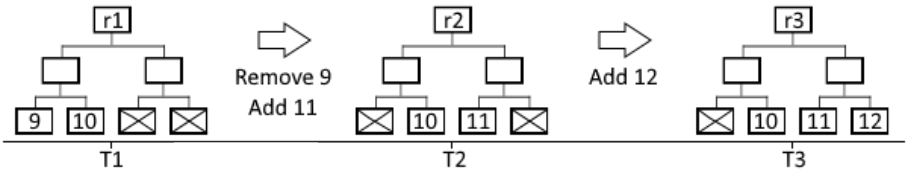


Figure 3.22: Study Merkle Tree Updates

the Merkle root. Consequently, a proof-of-membership for study 10 constructed against the second tree, cannot be verified against the third tree. This leads to a race between off-chain proof generation and on-chain tree updates.

To solve this problem a compromise solution is adopted. Study updates are batched to a maximum of one per hour. Next, proof-of-membership for the last n updates are allowed, creating a tolerance window. Thus, consents for recently discontinued studies are accepted. These are verified and confirmed as usual, but no partner data is forwarded for discontinued studies.

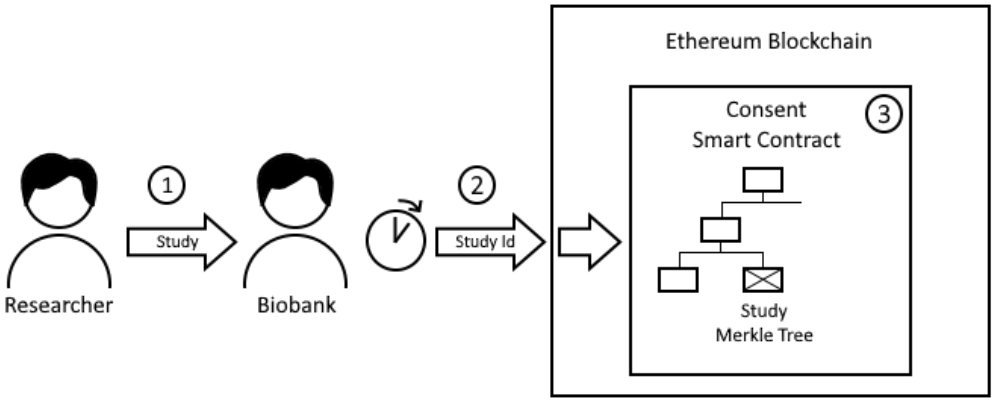


Figure 3.23: Biobank publishes study termination updates.

Figure 3.23 depicts study termination. A researcher notifies the biobank that a study is being terminated (1). The biobank waits until the next update is due and submits a study termination transaction (2), deleting the study identifier (3).

## 3.4 | Object Construction

Whereas the system simply represents studies by a numeric identifier, partner identities and consents are implemented as private tokens. This section presents the construction of each of these objects.

**Study** $(S_{id})$
: Represents a research study and is composed of a single value, the study identifier $S_{id}$.

**Identity token** $(a_{pk}, \rho_{id})$
: Represents a research partner. The token is composed of the partner's identity public key $a_{pk}$ and a random token identifier $\rho_{id}$.

**Consent token** $(a_{pk}, S_{id}, ch, r, \rho_c)$
: Represents a research partner's consent choice. The token is composed of the partner's identity public key $a_{pk}$, the study id $S_{id}$, the choice flag $ch$, randomness $r$ and a token identifier $\rho_c$.

Note that unlike Zeth, our token identifiers $\rho_x$ are random values. This will be explained in more detail when discussing exposure to the Faerie Gold Vulnerability in section 3.8.

## 3.5 | Data Exchange Across Participants

The consent solution includes eight distinct transaction types. For each, the data exchanged between participants is summarized. This also includes some additional data, resulting from the private token design patterns:

1. All research partners are syncing off-chain Merkle trees and nullifier lists as described in section 2.4.5.1.

2. Whenever a research partner submits a consent, new self-addressed tokens are created. Self-addressed ciphers encrypting these tokens are also generated. These are submitted together with the auditor and biobank ciphers, and likewise are distributed through events. This approach is identical to how private tokens return change and is most useful in wallet recovery. Given the identity and encryption key pairs, one can always recover owned tokens from the blockchain.

| **Research Partner Sign-up** | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| New Research Partner | Creates public/private identity key pair | Off |
| | Creates encryption/decryption key pair | Off |
| | Receives new identity token | On |
| | Reads current Merkle trees and nullifier lists | On |
| | Syncs off-chain Merkle trees and nullifier lists | Off |
| All Research Partners | Receive new identity token event | On |
| | Update off-chain identity Merkle tree | Off |
| Biobank | Receives partner identity public key | Off |
| | Receives partner encryption key | Off |
| | Receives partner biospecimen, contact details, etc. | Off |
| | Creates new identity token | On |
| Auditor | Receives new identity token event | On |
| | Increments partner count | Off |

Table 3.1: Research Partner Sign-up

| **Study Sign-up** | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| All Research Partners | Receive new study event | On |
| | Update off-chain study Merkle tree | Off |
| Biobank | Creates new study identifier | On |
| Auditor | Nothing | |

Table 3.2: Study Sign-up

| Consent Minting | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| Consenting Research Partner | Spends one-time study mint | On |
| | Creates new consent token | On |
| | Spends current identity token | On |
| | Creates new identity token | On |
| All Research Partners | Receive new consent token event | On |
| | Receive new/removed identity tokens event | On |
| | Update off-chain consent and identity Merkle trees | Off |
| | Update off-chain identity nullifier list | Off |
| Biobank | Receives Shamir's secret shares | On |
| | Recovers consent token details with auditor | Off |
| Auditor | Receives Shamir's secret share | On |
| | Recovers choice flag with biobank | Off |
| | Maps consent commitment-to-choice | Off |

Table 3.3: Consent Minting

| Consent Change | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| Consenting Research Partner | Spends current consent token | On |
| | Creates new consent token | On |
| | Spends current identity token | On |
| | Creates new identity token | On |
| All Research Partners | Receive new/removed consent tokens event | On |
| | Receive new/removed identity tokens event | On |
| | Update off-chain consent and identity Merkle trees | Off |
| | Update off-chain consent and identity nullifier lists | Off |
| Biobank | Receives Shamir's secret shares | On |
| | Recovers consent token details with auditor | Off |
| Auditor | Receives Shamir's secret share | On |
| | Recovers choice flag with biobank | Off |
| | Maps consent commitment-to-choice | Off |

Table 3.4: Consent Change

| Consent Confirmation | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| Consenting Research Partner | Receives consent confirmation | On |
| Other Research Partners | Nothing | |
| Biobank | Creates consent confirmation<br>Adds/Removes Research Partner to Study | On<br>Off |
| Auditor | Receives consent confirmation<br>Updates consent count | On<br>Off |

Table 3.5: Consent Confirmation

| Research Partner Termination | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| Terminated Research Partner | Spends current identity token<br>Creates ephemeral identity token | On<br>On |
| Other Research Partners | Receive removed identity token event<br>Update off-chain identity nullifier list | On<br>Off |
| Biobank | Receives ephemeral identity token details | On |
| Auditor | Receive removed identity token event<br>Decrements partner count | On<br>Off |

Table 3.6: Research Partner Termination

| Research Partner Termination Confirmation | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| Terminated Research Partner | Receives termination confirmation | On |
| Other Research Partners | Nothing | |
| Biobank | Creates termination confirmation<br>Deletes partner data | On<br>Off |
| Auditor | Nothing | |

Table 3.7: Research Partner Termination Confirmation

| Study Termination | | |
|---|---|---|
| **Participant** | **Operation** | **On/Off Chain** |
| All Research Partners | Receive removed study event<br>Update off-chain study Merkle tree | On<br>Off |
| Biobank | Removes study identifier | On |
| Auditor | Nothing | |

Table 3.8: Study Termination

## 3.6 | Audit Data

The sign-up endorsement described in section 3.3.1 gives the biobank significant control. The biobank could potentially inflate sign-ups by generating fake identity tokens. Tying biobank financing to sign-ups also provides a financial incentive for this to happen. To prevent this, the auditor needs to cross verify the on-chain identity count against the biobank's off-chain data. The dissertation recognizes this to be an open problem, requiring an auditing element that is tailormade for the biobanking context.

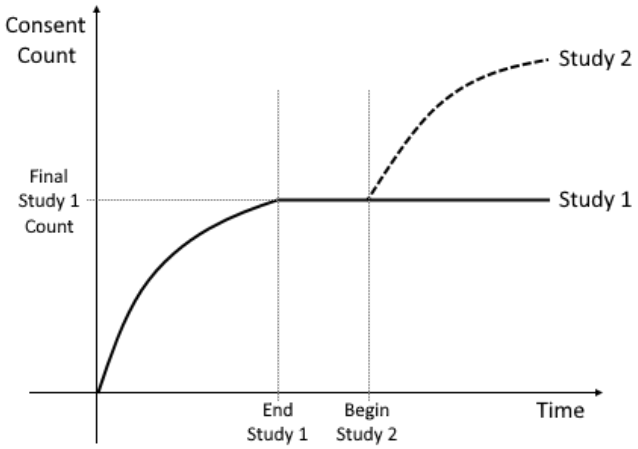


Figure 3.24: Consent Count vs Time, without zeroing the count for discontinued studies.

Fake sign-ups open the opportunity for fake consents. Thus, preventing them is also critical to the consent count. Given adequate fake sign-up protection is in-place, the consent count still needs some interpretation. When a study is terminated, all its consents are frozen. Figure 3.24 illustrates this for two studies. When the first is terminated, the baseline for the second study is offset.

The auditor is better off measuring the rate of consents. This can still be an effective performance indicator. If an exact count is required, the auditor dataset can be extended to include study identifiers as discussed in 3.3.5.2. Consents for terminated studies can then be zeroed.

## 3.7 | Selective Disclosure

Section 3.1.2.4 introduced immutable receipts as a source of trust. A research partner can selectively disclose these, even after terminating biobank participation. Partners only need their identity and encryption key pairs to retain this ability.

Consider a data leak that discloses the public keys of partners included in some ethically sensitive research. One partner checks his on-chain transaction history, discovering that he/she never minted a consent for this study. To verify this claim, an independent judge constructs a ZKP for proving that:

1. The partner owns one of the leaked public keys.
2. The public key corresponds to an identity token.
3. The partner never minted a consent token for this study.

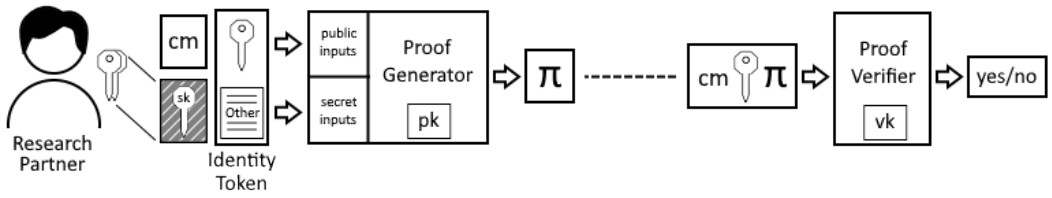


Figure 3.25: Selective disclosure of identity ownership.

Figure 3.25 shows a ZKP for proving the first two statements. This is very similar to the commitment opening proof discussed in appendix 2.4.4, whose public key is moved to the public input set. The judge confirms that:

1. The proof verifies correctly.
2. The public key is one of the leaked public keys.
3. The commitment is within the smart contract identity commitment set.

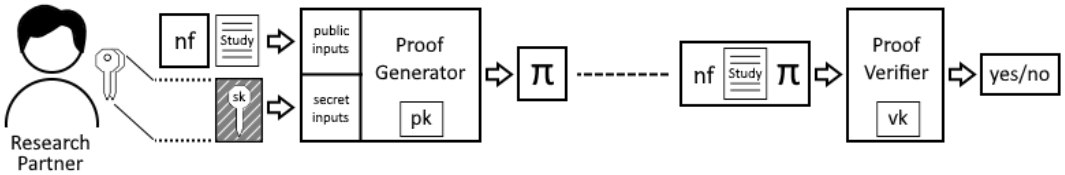


Figure 3.26: Selective disclosure of nullifier knowledge.

Figure 3.26 shows a ZKP for proving the third statement, requiring the prover to correctly compute and disclose the minting nullifier for the study of interest. The judge thus confirms that this nullifier is not listed at the smart contract, hence such a mint never took place.

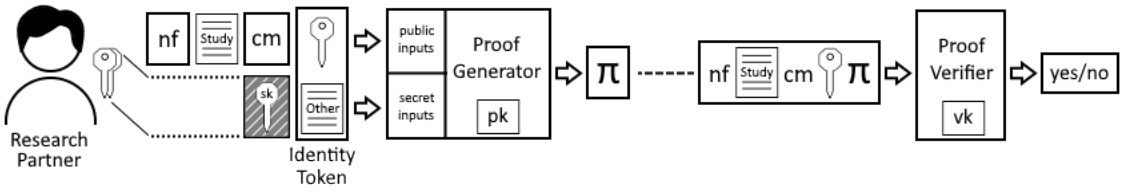


Figure 3.27: Selective disclosure of both identity ownership and nullifier knowledge.

Lastly the judge must ensure that both proofs used the same key pair. This results from merging the ZKPs as shown in figure 3.27, ensuring that a common secret key is used throughout.

The claim is proven without any biobank involvement. The partner is relinquishing some privacy. However, the proof and public inputs are only available to the judge. In a similar manner, ZKPs can be constructed to disclose any other commitment and nullifier. Transaction receipts become tangible evidence, holding the biobank accountable.

## 3.8 | Faerie Gold Vulnerability

Section 2.4.3.7 describes the Faerie Gold vulnerability, exploiting token identifier reuse. This problem is relevant when different parties exchange tokens. Malicious senders try to trick recipients with tokens that cannot be spent. The consent solution largely relies on self-minting and transfers having the same sender and recipient, thus making protection against this vulnerability unnecessary.

The only exception arises on partner sign-up, where the biobank mints the first identity token for new partners. Even if the biobank were motivated to attempt such an exploit, this requires transferring two tokens with the same identifier. Yet, the workflow only provides for one such transfer per partner.

For this reason, no Faerie Gold protection is included. Token identifiers are generated randomly.

## 3.9 | Summary

This chapter presented an alternative design to Dwarna's dynamic consent solution. The design starts from the guiding pillars of decentralization and privacy. Various decentralization vectors were applied. The blockchain was employed as an independent source of trust, an active auditor role was introduced, and workflows were redesigned to empower research partners. Privacy was primarily tackled through the exchange of

private tokens. On the other hand, ZKPs were earmarked to verify the honest workflow execution.

The chapter presents workflows covering research partner sign-up and termination, study sign-up and termination, changing consents, and confirming user-initiated requests.

The solution presents an auditing model tying biobank financing to user sign-ups and study participation. This transformed some workflows into an exchange involving three parties. Here private tokens were combined with a scheme based on Shamir's secret sharing.

The chapter finally shows an example of how research partners can selectively disclose information. This positions the immutable consent history as a permanent record of actions keeping the biobank accountable.

# 4

# Consent Solution ZKP Implementation

This dissertation implements the consent system ZKPs. Various ZKP schemes with different properties are available. However, time limitations did not allow for an in-depth study of alternative schemes. Instead, the choices made for Zeth were carried forward, specifically Groth16 (Groth, 2016) and PGHR13 (Parno et al., 2013).

Various libraries and toolsets for implementing ZKPs are available. The Zcash initial release was developed using the libsnark C++ library[1], later they switched to the Bellman Rust library[2].

A higher-level alternative is ZoKrates (Eberhardt and Tai, 2018). This sits on top of libraries like libsnark and Bellman to provide a domain specific language for defining ZKPs. ZoKrates also provides the functionality for ZKP setup, proof generation and verification. Furthermore, it includes the generation of a verification smart contract ready for Ethereum deployment.

Zeth builds on top of libsnarks. This dissertation heavily reuses Zeth's code and carried forward the same library choice. All code is available on github[3].

## 4.1 | Malleability

Blockchains like Ethereum support a native identity scheme, allowing users to sign and claim transaction ownership. Attackers cannot modify transactions as that would cause signature validation failure.

[1] libsnark C++ library [Accessed Jul 2021] – `https://github.com/scipr-lab/libsnark`
[2] Bellman Rust library [Accessed Jul 2021] – `https://docs.rs/bellman/0.11.1/bellman/`
[3] zkConsent [Accessed Aug 2022] – `https://github.com/kaxxa123/zkconsent`

Smart contracts inherit this anti-tempering protection whenever they integrate the native identity scheme. This is how the biobank accesses its functionality. However, the same is not true for users whose identity is being anonymized, like research partners. They, access the consent system by proving ownership of a private identity token, which is independent of the blockchain native identity.

Having two independent identity schemes, may open gaps in identity enforcement, leading to malleability attacks. These allow attackers to grab transaction data and re-submit it in their own transactions.

Groth16 proofs are malleable. Anyone observing the public inputs and proof, can generate a new unique valid proof without knowing the secret inputs. Thus, a unique valid proof does not imply honest computation.

Another potential malleability problem arises from the transaction data typical of ZKP solutions. These often include ciphers, a proof, and other data. A vulnerable application could allow attackers to replace one data element, maybe a cipher, without breaching transaction validation.

Proof malleability can be addressed through non-malleable ZKP schemes (Groth and Maller, 2017). Even so, solutions may still be exposed to transaction data malleability. Zeth implements a solution based on one-time signatures dealing with both. Zeth's solution is also applicable to the consent system. For more details, the reader is referred to the Zeth Protocol Specifications[4].

[4]Zeth Protocol Specification [Accessed Sep 2021] - `https://tinyurl.com/et8rdtd6`

## 4.2 | Consent Transaction ZKPs

The consent solution is composed of eight transaction types, five originate from the biobank, whilst three originate from research partners. Out of these, only five implement ZKPs. Table 4.1 and table 4.2 describe the ZKPs for each transaction type.

The choice whether a transaction necessitated a ZKP was based on the privacy requirements of the data being handled. Transactions for study sign-up and termination handle public data. Hence ZKPs were not required.

Privacy issues arise when dealing with research partner data. The system includes six such transactions, out of which five employ ZKPs. The exception here is the partner sign-up transaction for which no ZKP is employed. This transaction raised interesting considerations that are discussed further in section 5.4.1.

Some additional points to note:

1. A hashing wrapper ZKP, discussed in section 4.5, is applied to all research partner transactions.

2. On consent minting and changing, the non-zero study identifier proof, ensures that the proof is not referencing an empty Merkle tree leaf.

3. On consent changing, the inverted choice flag proof ensures an alternating sequence of choices as discussed in section 3.3.4.

| **Biobank Transaction** | **ZKP Proof of…** |
|---|---|
| Partner Sign-up | – |
| Study Sign-up | – |
| Study Termination | – |
| Consent Confirmation | Consent commitment opening |
| Termination Confirmation | Identity commitment opening |

Table 4.1: ZKPs for Biobank Transactions

| **Partner Transaction** | **ZKP Proof of…** |
|---|---|
| Consent Minting | 1. Hashing wrapper preimage<br>2. Identity key pair ownership<br>3. Identity token nullifier computation<br>4. Identity token commitment computation<br>5. Study minting nullifier computation<br>6. Consent token commitment computation<br>7. Identity set membership<br>8. Study set membership<br>9. Non-zero study identifier<br>10. Malleability protection computation |
| Consent Changing | 1. Hashing wrapper preimage<br>2. Identity key pair ownership<br>3. Identity token nullifier computation<br>4. Identity token commitment computation<br>5. Consent token nullifier computation<br>6. Consent token commitment computation<br>7. Identity set membership<br>8. Consent set membership<br>9. Study set membership<br>10. Non-zero study identifier<br>11. Inverted choice flag<br>12. Malleability protection computation |
| Partner Termination | 1. Hashing wrapper preimage<br>2. Identity key pair ownership<br>3. Identity token nullifier computation<br>4. Identity token commitment computation<br>5. Identity set membership<br>6. Malleability protection computation |

Table 4.2: ZKPs for Research Partner Transactions

## 4.3 | Differences from Zeth

The consent system development started by forking Zeth's code. Thus, the dissertation results includes some comparisons against Zeth. To allow for this, a high-level description of Zeth's implementation is necessary.

Some key differences between the consent system and Zeth include:

1. Zeth has a single ZKP for all operation types; minting, transferring and burning. This allows combining these operations into a single transaction. The consent solution maps each operation to a different transaction and a distinct ZKP.

   Zeth's approach only requires a single trusted setup, whereas the consent solution requires five such setups. Consequently, Zeth is leaner, with a single client-side proof generator and a single on-chain verifier. All of this gets multiplied by five in the consent system, with five client-side proof generators and five on-chain verifiers.

2. Zeth allows spending two tokens and creating two new tokens in a single transaction. The consent system, at most, also spends and creates two new tokens on changing consent.

3. Zeth users may have multiple identities. Within one transaction, each input ZethNote could have different ownership identities. The consent system allows for one identity per partner. Consequently all consumed tokens share a common ownership identity.

4. Zeth gives the option to address output tokens to anyone. In comparison, all identity/consent token transfers are self-addressed.

From a high level the Zeth ZKP construct is described in table 4.3.

| **ZKP Proof of...** |
| --- |
| 1. Hashing wrapper preimage<br>2. ZethNote key pair ownership (x2)<br>3. ZethNote nullifier computation (x2)<br>4. ZethNote commitment computation (x2)<br>5. ZethNote set membership (x2)<br>6. Malleability protection computation (x2)<br>7. Faerie Gold protection computation (x2)<br>8. Total input/output balances match |

Table 4.3: Zeth ZKPs

## 4.4 | Smart Contracts

Smart contracts relying on ZKPs can be described to be made up of:

1. ZKP verifier(s)
2. Support components (Merkle trees, nullifier lists etc.)

These are depicted in figure 4.1.

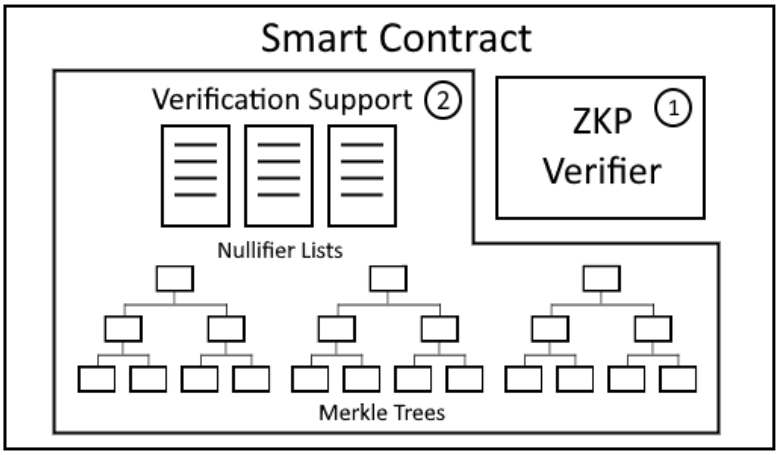


Figure 4.1: Smart contract composed of ZKP verifier and support components.

This dissertation only included the ZKP development, proof generation and verification. Thus, from figure 4.1 only the ZKP Verifier (1) was developed.

## 4.5 | ZKP Input Hashing Wrapper

An important implementation detail, helpful in interpreting the results in chapter 5, is the presence of a hashing ZKP wrapper. This reduces ZKPs that would normally take multiple public inputs to a single public input.

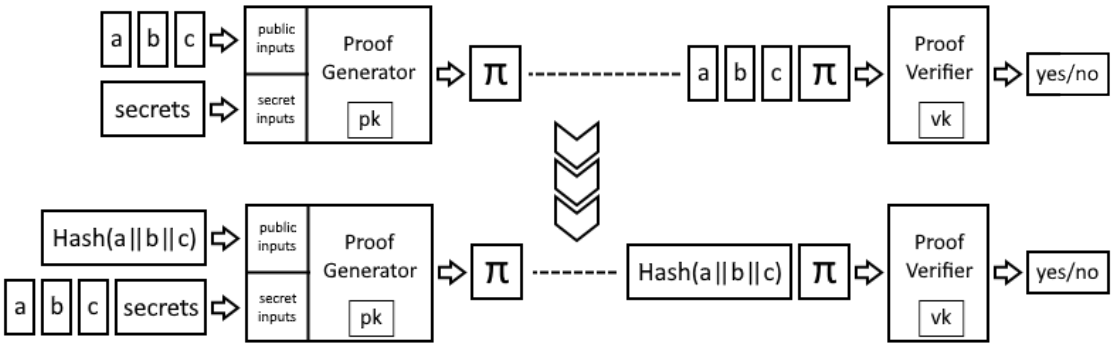


Figure 4.2: ZKP hashing wrapper, moving public inputs to the set of secret inputs.

Figure 4.2 depicts how the wrapper transforms a ZKP. At the top, the original ZKP takes three public inputs. At the bottom, the hashing wrapper is added, allowing proving knowledge of a hash preimage. It reduces the public inputs to a single hash, whose preimage is the set of original public inputs. Consequently, the original public inputs become secret.

## 4.6 | Summary

This chapter discusses the implementation of five ZKPs required by the consent solution. It provides a high-level description of the computations verified by each ZKP. A description of Zeth's ZKP, and how it differs from the consent solution is also included.

This chapter also breaks verification smart contracts into two blocks, a ZKP verifier and other support components. Lastly the construction of a ZKP for hashing proof inputs is described. All this information is important for interpreting the results presented in chapter 5.

# 5

# Results and Discussion

This chapter presents test results for both the ZKP-based consent system and Zeth. Having developed five ZKPs of differing complexity, provides an insight into how complexity impacts storage requirements, processing time and transaction fees. Thus, a measure of the resource requirements is presented, addressing the first research question. Whilst clearly implementation specific, these results can be indicative to similar solutions. Further research into alternative blockchains and different technology choices is required to address the research question more comprehensively.

Following that, a discussion of the common challenges that privacy preserving solutions need to tackle is presented, addressing the second research question.

## 5.1 | Proving and Verification Keys

This section focuses on the ZKP proving and verification keys, returned by the libsnark setup API. This provides a measure of the ZKP client-side and on-chain storage requirements. Furthermore, it allows comparing different ZKP outcomes.

Key sizes depend on how ZKPs are coded. Running the same setup multiple times always returns the same metrics. The complete libsnark output is presented in Appendix A.

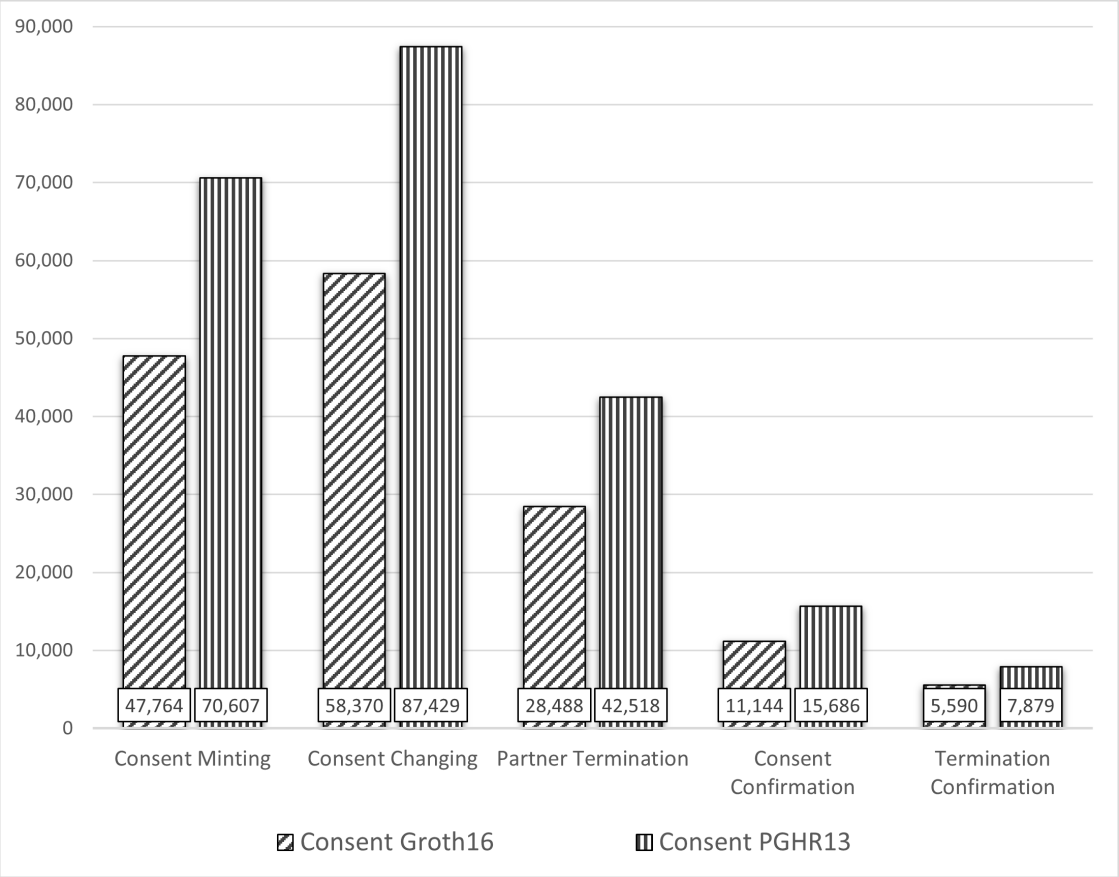


Figure 5.1: Consent Groth16 and PGHR13 Proving Key Sizes (KB)

Figure 5.1 charts the consent solution proving key sizes for Groth16 and PGHR13. It highlights a significant difference between the two. PGHR13 returned proving keys with a size increase ranging between 41% (Consent Confirmation) and 50% (Consent Changing).

Proving keys also registered significant size variation across ZKPs. Consent Changing has the largest key, whereas Termination Confirmation has the smallest key. Referring to table 4.1 and table 4.2, one observes the correlation between size and complexity. Consent Changing and Minting are the most complex ZKPs. However, the former includes an additional proof of membership resulting in the largest key. At the lower end, both Consent Confirmation and Termination Confirmation perform a commitment opening proof. Yet, the latter deals with a smaller preimage producing the smallest key.

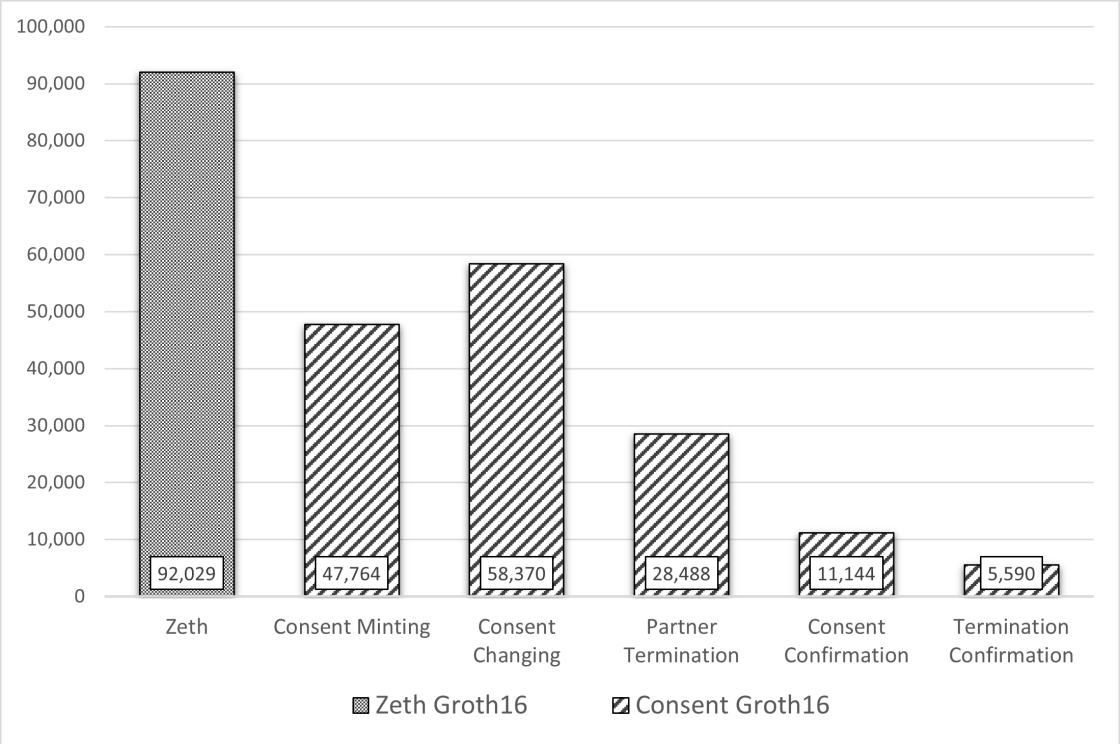


Figure 5.2: Consent and Zeth Groth16 Proving Key Sizes (KB)

Zeth's default build runs Groth16, and only data for this scheme is presented. Figure 5.2 charts the Groth16 proving keys for both Zeth and the consent solution. Comparing Zeth's proving key to that for Consent Changing, one observes that Zeth's key is larger by 58%. Both deal with four tokens, two inputs and two outputs. However, differences include:

1. All Zeth input/output tokens are ZethNotes. In the consent system, both inputs and outputs are composed of an Identity and Consent token pair. The Consent token construction mimics a ZethNote. However, the Identity token is much simpler.

2. Consent system tokens share a common ownership proof, whilst Zeth requires an ownership proof for each input token.

3. Consent Changing requires one less malleability proof than Zeth. This results from the number of ownership proofs.

4. Zeth requires two proofs for Faerie Gold protection whilst Consent Changing implements none.

These differences inflate Zeth's proving key. Consent Changing does perform an extra proof of membership. However, ultimately Zeth still ends up with the largest key.

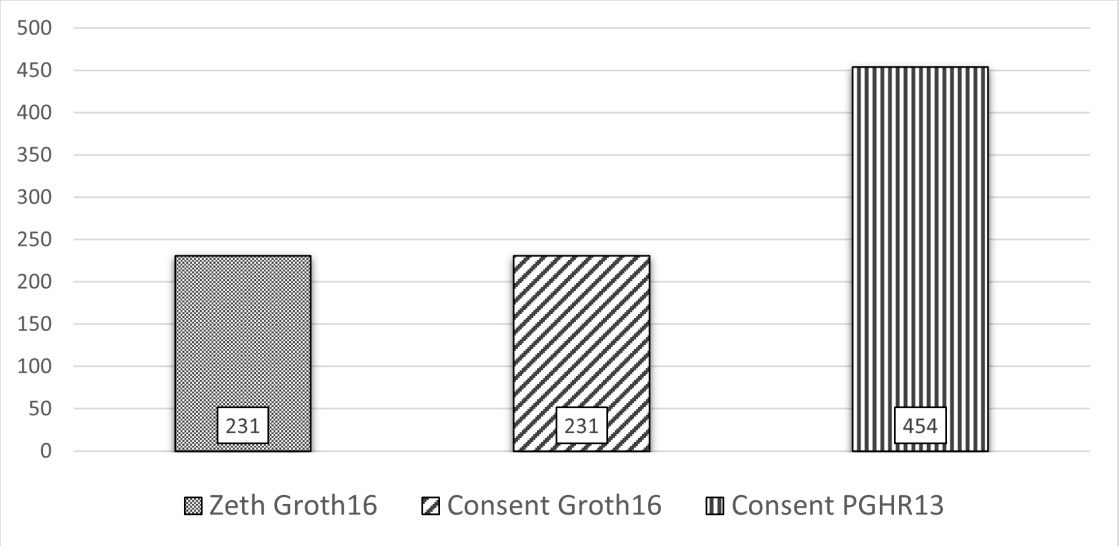


Figure 5.3: Consent and Zeth Verification Key Sizes (Bytes)

Figure 5.3 charts the verification key sizes for both solutions. For a given scheme, all ZKPs have the same verification key size. Groth16 verification keys also have the same size across applications, Zeth and the consent system. In general, verification key size depends on the number of public inputs. However, since both solutions use the wrapper described in section 4.5, all multi-input ZKPs are being reduced to a single input.

Across schemes, PGHR13 inflates size by 97%. In all cases the verification key is much smaller than the corresponding proving key. Being stored on-chain, their small size is important in minimizing transaction fees.

## 5.2 | Proof Generation

This section presents and compares proof sizes and generation times. Proofs are generated on the client-side and passed as transaction parameters. Their size adds to the transaction cost. Furthermore, the proof generation waiting time is relevant to the user's experience.

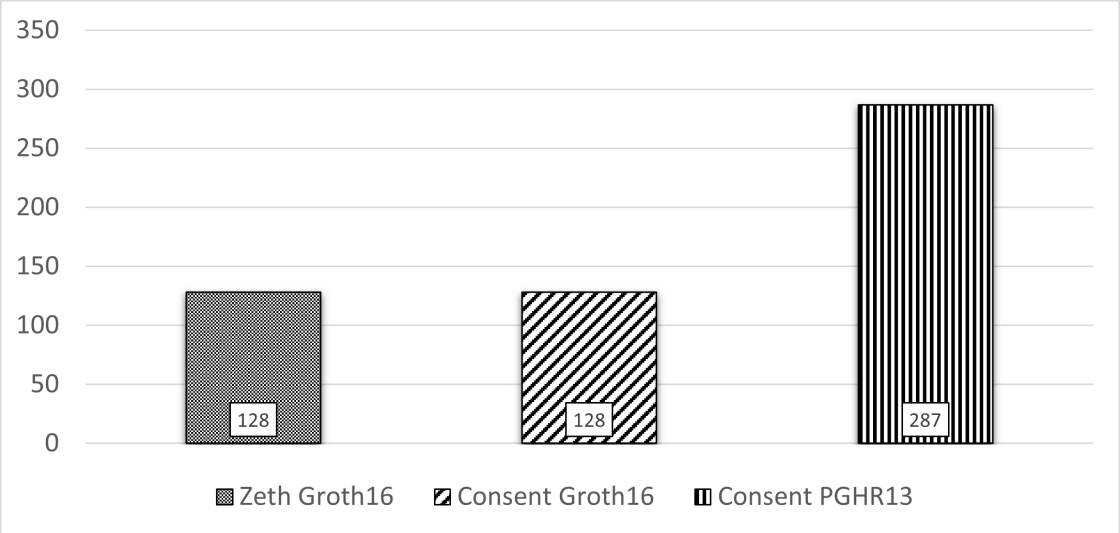


Figure 5.4: Consent and Zeth Proof Sizes (Bytes)

Figure 5.4 charts the proof sizes for all ZKPs. For a given scheme, all proofs have the same small, fixed size. Here, ZKP complexity and number of inputs make no difference. This proof succinctness property is a key requirement for blockchain implementation feasibility. Across schemes, PGHR13 results in a 125% size increase.

Figure 5.5 and figure 5.6 chart the average proof generation times. Consent system timings were computed by averaging the time for generating 10 proofs. As for Zeth, 10 proofs were generated for each operation type: transfer, mint, and burn. No significant difference was observed across operations. Thus, Zeth's timing is the average of all 30 tests.

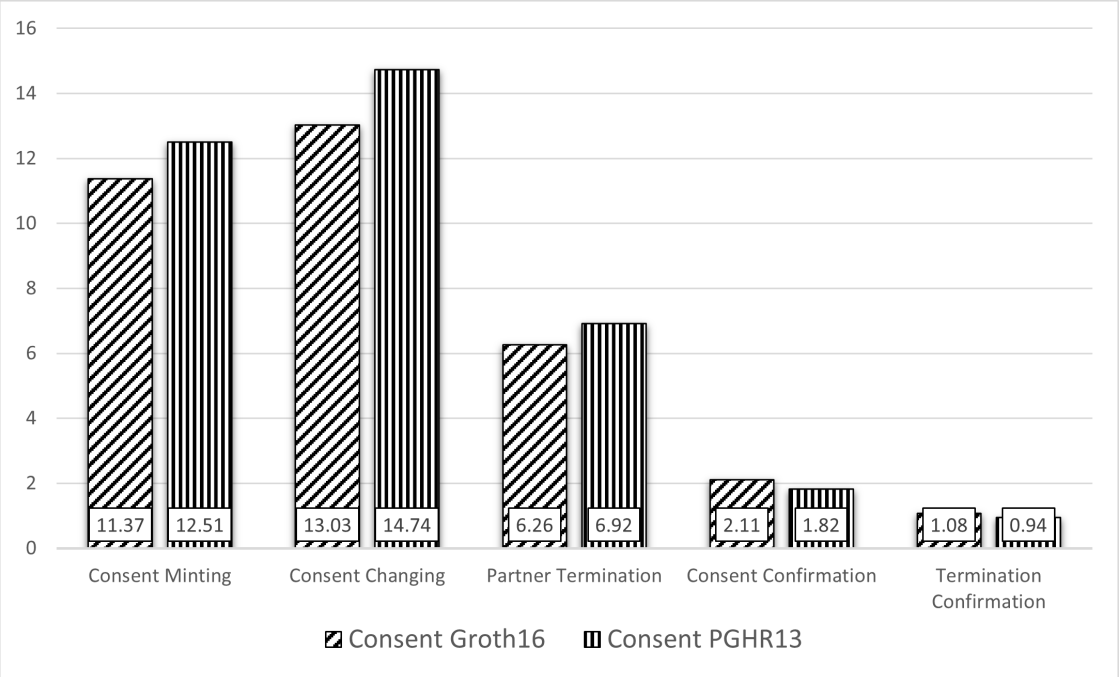


Figure 5.5: Consent Groth16 and PGHR13 Proof Generation Times (s)

Figure 5.5 charts the consent solution proof generation time for Groth16 and PGHR13. Here little variation was observed across schemes. However, the correlation between ZKP complexity and proof generation time is again evident. The more complex the ZKPs, the longer proof generation takes.

For consistency, figure 5.6 charts the Groth16 proof generation time for both Zeth and the consent solution. However directly comparing Zeth to the consent system isn't appropriate. The two were tested in different environments. Zeth ran within a Docker container based on Alpine Linux v3.12. The consent system ran on Ubuntu 20.04 LTS hosted over Windows Subsystem for Linux 2.

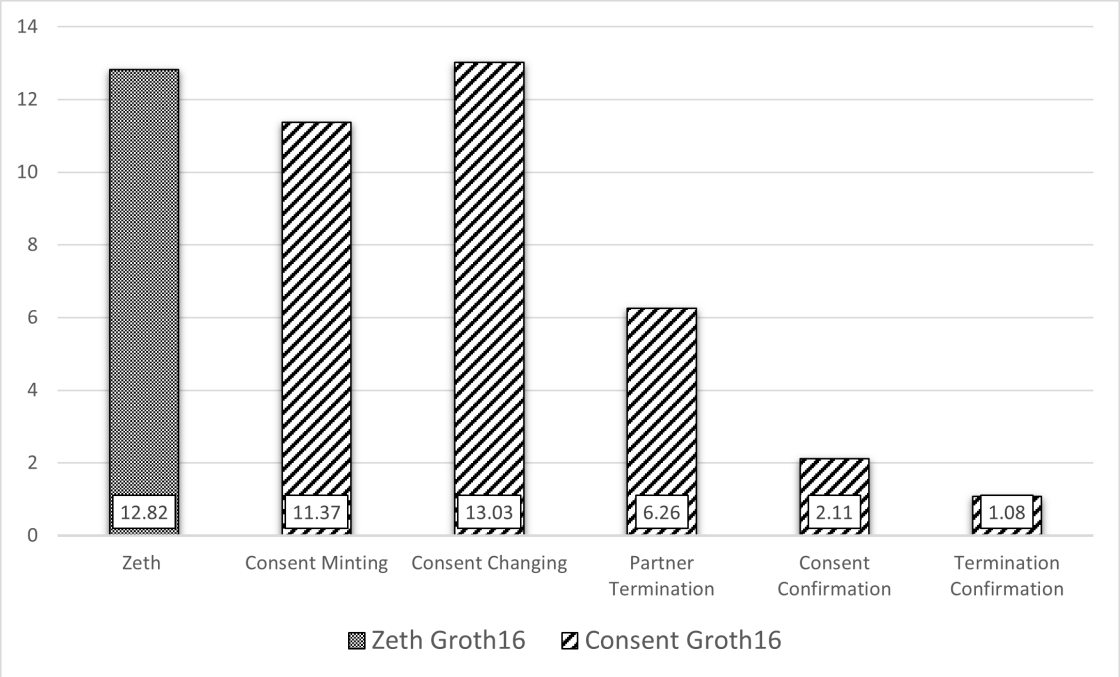


Figure 5.6: Consent and Zeth Groth16 Proof Generation Times (s)

## 5.3 | Proof Verification

This section explores another feasibility metric, the smart contract transaction fees. First, the Ethereum gas cost estimate is presented. Next, the estimate is converted to Euro.

As section 4.4 explains, consent system development only included the ZKP verifiers. This mainly involved extracting Zeth's verifier implementation. Thus figure 5.7 only shows the gas consumption for the ZKP verifiers. As for Zeth, Clearmatics provides the complete smart contract. Figure 5.8 shows a breakdown of Zeth's gas consumption, separating ZKP verification from the total.

These gas estimates result from the estimateGas Web3 API run against a local ganache-cli instance. Zeth's total gas estimate was obtained from the custom ganache-cli provided by Zeth. For a given scheme, the variation in gas consumption was negligible. Across schemes, PGHR13 is 220% more expensive than Groth16.

Normally gas consumption depends on the number of public inputs. However, as discussed in section 4.5, all ZKPs are being reduced to a single public input. Thus, gas is fixed to a single input verification.

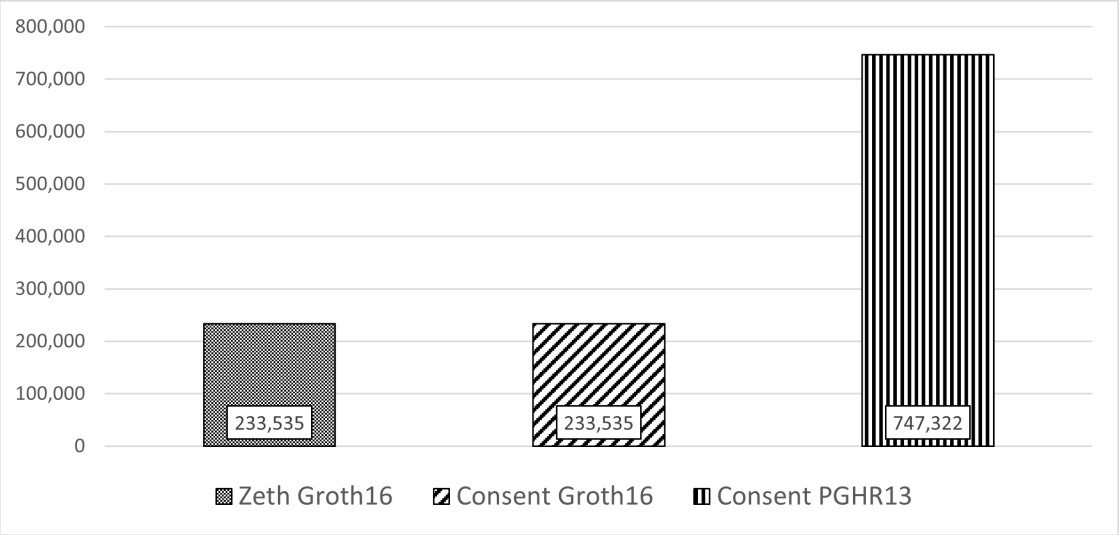


Figure 5.7: Consent and Zeth ZKP Verification Cost (Gas)

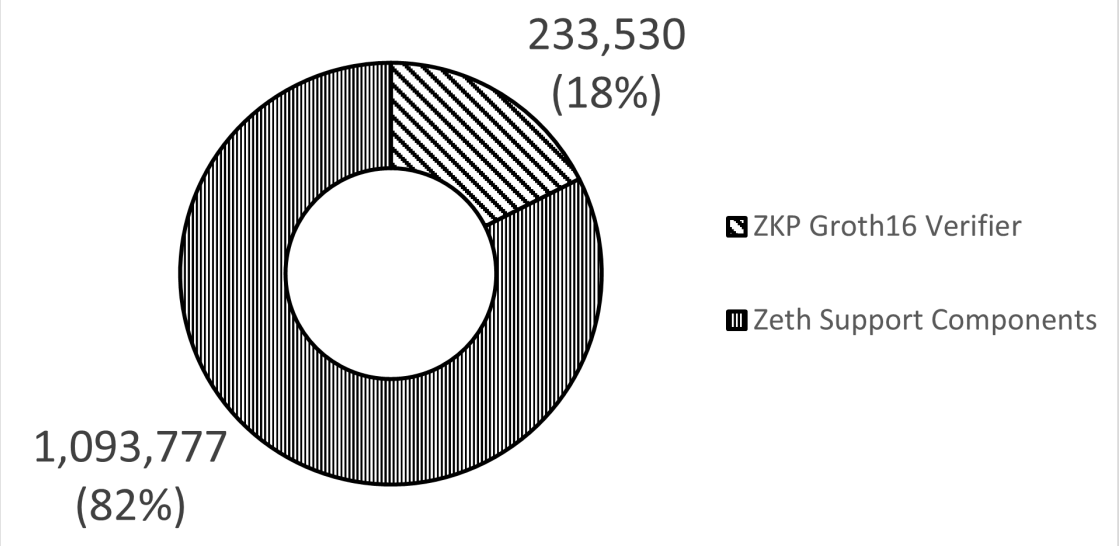


Figure 5.8: Zeth's Gas Consumption for ZKP Verification and Support Components

Figure 5.8 shows that in Zeth, the ZKP verification only consumes 18% of the total cost, the rest is consumed by the support components. One can safely assume that the consent system total gas will be even higher. Whereas Zeth only requires a single Merkle tree and nullifier list, the consent system requires three of each.

At the time of writing (December 2021) the Ethereum block limit is 30,000,000 gas units. Ethereum aims to consume half of this. Either way, Zeth's mixer total gas is well

below the limit. Even if we included its additional Merkle trees, one expects the consent system to stay below this limit.

To obtain an estimate of the transaction fee costs, the base fee for each mined block during November 2021 was retrieved, covering block range [13527859, 13717847]. Since the London fork[1], the base fee provides the minimum transaction gas fee. On top of that, one would pay an additional tip, which is not included here. Table 5.1 summarizes the fee data.

| **November 2021 Fees** | **Base Gas Fee (Wei)** | **Base Gas Fee (GWei)** | Block Number |
|---|---|---|---|
| Lowest | 41,802,844,503 | 41.80 | 13701717 |
| Highest | 1,804,238,207,798 | 1804.24 | 13717205 |
| Median | 119,216,670,515 | 119.22 | |
| Mean | 127,059,985,292 | 127.06 | |
| Standard Deviation | 47,409,562,748 | 47.41 | |

Table 5.1: Ethereum Base Gas Fee - November 2021

Next, table 5.2 shows the daily high and low EUR/ETH pricing, for November 2021 from CoinMarketCap[2].

| **Euro per Ether Price** | **Euro** | **Day** |
|---|---|---|
| Lowest | 3,498 | Nov 18, 2021 |
| Highest | 4,302 | Nov 16, 2021 |
| Median (Low) | 3,782 | |
| Mean (Low) | 3,766 | |
| Standard Deviation (Low) | 168 | |
| Median (High) | 3,966 | |
| Mean (High) | 3,996 | |
| Standard Deviation (High) | 155 | |

Table 5.2: Ether Euro Pricing - November 2021

Finally, all this data was combined to obtain the Euro transaction fee estimates shown in table 5.3. For simplicity these are only based on the lowest Euro price point (€3,498/ETH).

The Groth16 verifier on its own, requires fees between €34 and €1,474, with an average of €104. The complete Zeth smart contract starts from €194 up to €8,377 with

[1]The history of Ethereum [Accessed Nov 2021] - `https://ethereum.org/en/history/`
[2]CoinMarketCap Historical Data [Accessed Nov 2021] – `https://tinyurl.com/mc82jcu5`

| | Gas | Lowest | Highest | Median | Mean | Standard Deviation |
|---|---|---|---|---|---|---|
| Groth16 Verifier | 233,535 | €34 | €1,474 | €97 | €104 | €39 |
| PGHR13 Verifier | 747,322 | €109 | €4,716 | €312 | €332 | €124 |
| Zeth No Verifier | 1,093,777 | €160 | €6,903 | €456 | €486 | €181 |
| Zeth Total | 1,327,307 | €194 | €8,377 | €553 | €590 | €220 |

Table 5.3: Smart Contract Transaction Fee Cost in Euro

an average of €590. These fees are per transaction. The consent smart contract, with its additional support components, is expected to be even more expensive. Thus, one concludes that this solution is not financially feasible if implemented on Ethereum.

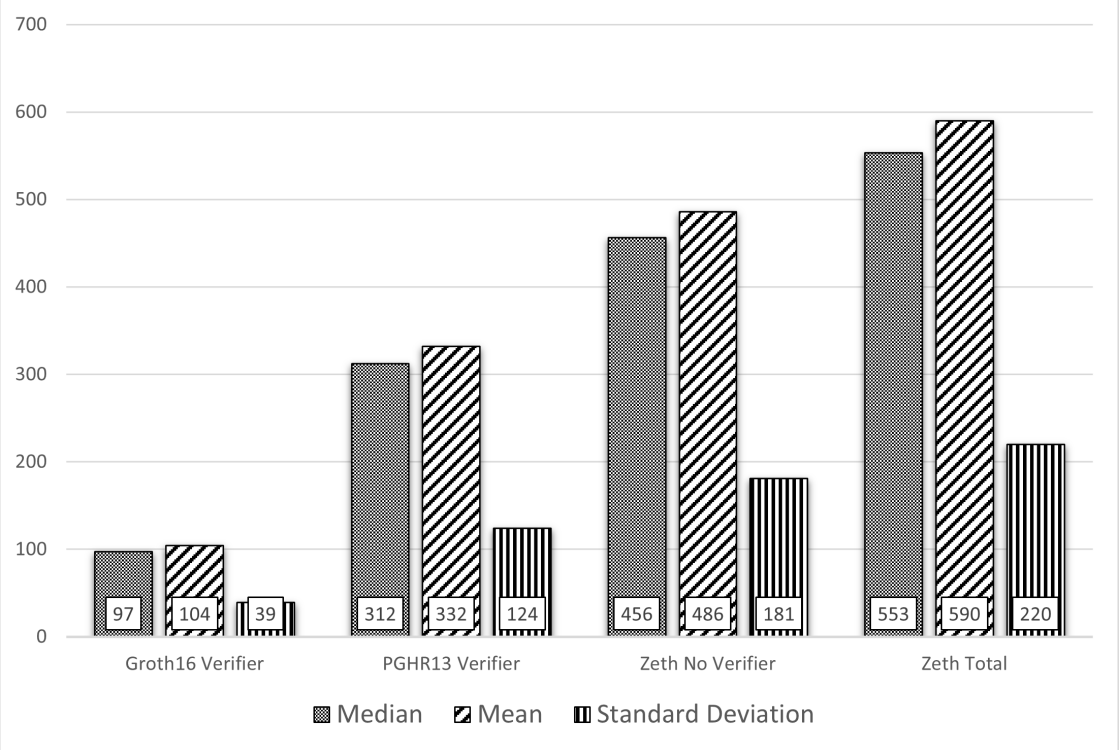


Figure 5.9: Smart Contract Transaction Fee Cost (Euro)

Figure 5.9 charts the EUR cost results for the mean, median and standard deviation side-by-side.

## 5.4 | Common Design Challenges

Redesigning the consent solution, provided practical insight into the challenges of decentralized privacy. The following sections discuss such challenges, addressing research question two.

### 5.4.1 | Representing Real-World Assets On-Chain

Partner sign-up transactions, expose the challenges of representing real-world assets on-chain. Here, the biobank creates a new identity token. An on-chain transaction persists the token's commitment and forwards its details to the partner. Despite dealing with a private token, no ZKP is employed.

Without a proof of correct computation, the workflow is relying on the biobank's honest execution. Fact is, that even if a ZKP had to be in-place, the smart contract cannot block the creation of fake identity tokens. This problem was first introduced in section 3.6.

A ZKP only guarantees honest computation for the provided set of inputs. However, on sign-up, these inputs result from an off-chain process over which the smart contract has no control. At best a ZKP could ensure that an identity token commitment results from two variables purporting to be a public key and a token identifier. Whether the public key truly represents a partner is beyond the contract's scope.

Sign-up verification requires a process having access to both real-world and on-chain data. As suggested in section 3.6 to bridge together these two data silos, a manual cross-verification might be the only solution.

One observes that a similar problem does not exist when ZethNotes wrap Ether. In this case all assets exist on-chain. However, ZethNotes could represent any fungible token. As soon as ZethNotes wrap tokens having real-world counterparts, similar challenges arise.

### 5.4.2 | Financial Feasibility

Transaction fees play an important role in curbing blockchain misuse. However, these pose a financial feasibility hurdle. The tests carried out for Zeth resulted in fees exceeding €500 on average. Very few applications can justify such fees.

Ethereum's fees are expected to improve with the release of Ethereum 2. Otherwise, alternative blockchains could be considered. Or else, one could consider layer 2 scaling

solutions.

Most often, scaling solutions rely on an intermediate service that aggregates multiple transactions. The batch is committed onto layer 1 through a single transaction. Users pay a lower fee to the layer 2 service provider, part of which covers the layer 1 transaction fee.

ZKPs are enabling layer 2 solutions based on zk-rollups. The layer 2 provider validates each transaction and generates a single proof attesting for the transaction batch. This proof, together with a compressed version of the transactions are passed to the main blockchain. Here, the batch is only persisted if the proof is verified successfully. Whilst cutting the cost for individual transactions, when dealing with privacy solutions, one must watchout not to disclose private data to the layer 2 provider.

Zk-zk-rollups extend the same concept, employing ZKPs within both layers. ZKP based transactions are run at the layer 2 network. Multiple such transactions are then aggregated by a zk-rollup onto layer 1. This restores end-to-end privacy.

Aztec[3], is building such a scaling solution. This project includes Noir, a domain specific language for defining the layer 2 ZKPs. It would be interesting to redevelop the consent solution using Noir and compare the outcome. Clearmatics also have their own zk-zk-rollup aggregator project Zecale[4].

### 5.4.3 | Security and User Privacy Regulation

Privacy regulation is a major concern for applications handling user data. Just like Dwarna, the decentralized solution keeps most user data off-chain. Data that is immediately linkable to users is easy to delete. On-chain data is primarily composed of encrypted tokens, proofs, commitments, and nullifiers. Deleting these from an immutable ledger is not possible, thus keeping them available for attack.

Immutability could potentially conflict with the GDPR principle of storage limitation. Furthermore, a solution failing to meet security best practices is unlikely to satisfy GDPR article 5, mandating securing data from "unauthorised or unlawful processing".

Commitments and nullifiers that are computed as hashes could be targeted by a preimage attack. The commitment preimage is the token cleartext, whereas the nullifier preimage includes the user's identity private key. Alternatively, one could attempt breaking the ciphers. These would disclose all tokens owned by the attack target. In this case, a user's entire consent history could be disclosed.

[3]Aztec zk-zk-rollup [Accessed Jan 2022] – `https://aztec.network/`
[4]Clearmatics Zecale [Accessed Jan 2022] – `https://github.com/clearmatics/zecale`

Thus, the choice of security primitives is critical. This dissertation did not delve into this aspect. It rather adopted the primitives chosen for Zeth. Clearly blockchain immutability further accentuates the need for future-proof designs and building on the work of expert cryptographers helps avoid compromising security.

In such solutions, very often stealing user secret keys is the easiest attack vector. Users require these keys for interacting with the solution. Following termination, users may still want to retain their keys to keep service providers accountable, as discussed in section 3.7. Thus, the security of the wallet application storing these keys is critical.

The consent solution presented here, unlike Zeth, does not allow partners to own multiple identities. This is not ideal, as all secrets are tied to two key pairs. Privacy applications should instead strive to encourage the use of multiple identities as this helps reducing their exposure to successful attacks.

A compromised ZKP, enables an attacker to generate fake proofs. Schemes requiring a trusted setup, become vulnerable to such an attack if the setup secret parameters are not destroyed. For example, the attacker could gain the ability to spend tokens owned by others. In the biobanking case, partners could be enrolled to studies against their will.

Such an attack can be prevented through schemes that do not require a trusted setup. Otherwise, a trusted setup can be executed through a Multi-Party Computation (MPC). An MPC is secure if at least one participant is honest and does not retain his share of secret parameters. Thus, an MPC should involve participants with conflicting interests so as to ensure they do not collaborate.

### 5.4.4 | Holistic Approach to Privacy

The privacy level enjoyed by application users results from a combination of factors. The choice of security primitives and their implementation is fundamental. Beyond that, the design must be grounded onto a clear understanding of the application usage.

Currency mixers are known to have usage pitfalls that dilute privacy. A large user pool (anonymity set) is a basic requirement. Additionally, users are encouraged to transact using round currency units. A currency amount of 3.14159 is too unique. Someone depositing that amount, effecting a few transfers, and withdrawing the change, is easier to deanonymize. The uniqueness of the deposited and withdrawn amounts facilitates transaction linkage.

One should look for similar weaknesses in any privacy application. In the biobanking case, a setup involving just a few research studies, or lacking study variety, dilutes

the benefit of selecting studies in zero-knowledge. Small biobanks could merge together, providing a wider study choice and improving privacy.

Another relevant property is the one relating consents to study duration. Studies might attract most consents immediately on launch. Even if multiple diverse studies are available, if study releases are spread apart, one might correlate consent waves to study releases. Releasing diverse studies in batches could help countering this.

An alternative measure is to broaden the scope of the consent system. Instead of an application specific consent solution, one could consider a consent platform for diverse applications. However, such a change in scope would require a significant redesign effort.

User behaviour also impacts privacy. Section 3.3.3 highlights how users are more likely to mint consents on joining a study. It suggests countering this behaviour through the minting of additional tokens. However, such countermeasures again assumes a large selection of studies.

The application workflow rules may also enhance/degrade privacy. Section 3.3.4 discusses two consent change patterns labelled as the alternating and free sequence. This discussion exposed how even such design decisions impact privacy.

From these observations, one recognizes the need for a holistic approach to privacy. Some information leaks might appear too vague to undermine privacy. However, as Béres et al. (2021) show, deanonymization is often an exercise that correlates multiple data sources, amplifying deanonymization effectiveness.

### 5.4.5 | Spam/Malicious Transactions

This dissertation explored the issues around malicious/spammy transactions. Privacy can open opportunities for abuse. Section 3.3.5 describes attacks ranging from impersonation attacks to resource wasting spam.

This problem impacts some applications more than others. A key differentiator is the application economics. The contrast between Zeth and the biobanking example highlights this fact. Transaction fees are an anti-abuse safeguard whenever users are made to pay.

In Zeth, users pay the fees if the transacted asset value justifies the cost. The biobanking application lacks this economic incentive. It rather relies on altruistic participation, where usage cost is an adoption barrier. Such applications are likely to absorb end-user costs, losing this anti-abuse protection.

Countering invalid tokens increases design complexity. Zeth leaves it up to the transaction receivers. Invalid token reception is not blocked. Instead, receivers deal

with them within the scope of their real-world engagement with the sending party.

The biobanking application required the inclusion of countermeasures against this problem. The most serious attacks were dealt with through confirmation workflows. Options for the outstanding spam problem were discussed in section 3.3.5.3.

### 5.4.6 | Decentralization Approaches

Decentralization should not be approached with an all-or-nothing mindset. Real-world problems are often tied to entities that harbour poles of trust. Eliminating such roles may not be possible. Yet plenty of decentralization opportunities may still be available.

Decentralized blockchains provide application hosting and immutable storage, independent of workflow participants. This is already a significant decentralization opportunity. Whereas immutability makes GDPR compliance harder, it provides a permanent record of actions keeping service providers accountable.

Involving additional participants improves trust decentralization but complicates design. In this dissertation, integrating the auditor's role was the most challenging design element. Without the auditor, the solution would be a two-party exchange, perfectly fitting a private token transfer. The auditor transforms this into a three-party exchange, each having different data access levels.

Shamir's secrets proved invaluable, because of their flexibility. This approach can easily accommodate additional participants by changing the number of shares required to disclose secrets. On the downside, workflows become more complex, requiring extra work for disclosing secrets.

Additional participants may also be critical in bridging off-chain and on-chain processes as in partner sign-ups (section 5.4.1). Here some applications might be able to integrate IoT elements, making the system less dependent on human intervention.

User empowerment is another decentralization approach adopted by this dissertation. On partner sign-up, the centrality of the biobank could not be avoided. Beyond that, workflows are user driven. Consents are first persisted on-chain, immediately recording the partner's intent. The same is true for partner termination. This gives freedom of expression over which the biobank plays no role. Designing decentralized solutions should start by identifying the core services around which the system is centralizing trust. Even if the core service cannot be decomposed, anything beyond it, should be looked at from a decentralized perspective. This helps transforming workflows and reassigning tasks that would otherwise be centralized.

# 6

# Related Work

From a technical standpoint, the dissertation was largely about applying the tokenized privacy design patterns. Zeth is obviously the closest project, as the consent solution repurposed its code. This was already discussed extensively and test results for both Zeth and the consent solution were presented side-by-side. Other Ethereum layer 1 tokenized privacy projects that could have been considered as a starting point included Nightfall (Konda et al., 2019) and the Aztec Protocol (Williamson, 2018).

As for dynamic consent solutions, Dwarna (Mamo et al., 2019) is the closest work comparable to this dissertation. No other blockchain-based consent solution, that could readily cater for Dwarna's needs, was identified within academic literature.

## 6.1 | Nightfall

Nightfall (Konda et al., 2019) is similar to Zeth. It follows the mixer paradigm where users handle tokens within an anonymity set. However, apart for supporting fungible ERC20 tokens, it also directly supports wrapping non-fungible ERC721 tokens.

So far, there were three Nightfall releases, the third being from July 2021[1]. Subsequent releases aimed at reducing transaction fees. The latest includes a layer 2 solution based on optimistic rollups.

Nightfall initially implemented the non-malleable zkSNARK GM17 (Groth and Maller, 2017) scheme. However, their latest release switched to Groth16, the same scheme used in Zeth and the consent solution. They explain[2] that this switch brought about shorter computation times and lower gas fees.

[1]Nightfall July 2021 Release [Accessed Jan 2022] - `https://tinyurl.com/2p8mdwnj`
[2]G16 Simulation Extractability [Accessed Jan 2022] - `https://tinyurl.com/2cwww4r4`

Their zkSNARKs were developed using ZoKrates (Eberhardt and Tai, 2018), a higher-level alternative to the C++ libsnark library employed in Zeth. Thus, most likely ZoKrates would also satisfy the needs of the consent solution ZKPs. It would be interesting to compare the outcome of such an alternative implementation.

Just like the dissertation, but unlike Zeth, Nightfall relies on multiple ZKPs. Zeth merges all possible operations (mint/transfer/burn) into a single function and a single ZKP. Nightfall provides six distinct functions, three for fungible and three for non-fungible tokens, each having a distinct ZKP. Thus, Nightfall relies on multiple trusted setups against the single setup required by Zeth.

Another highly relevant Nightfall component is their reusable append-only Merkle tree implementation Timber[3]. This project also supports constructing off-chain tree replicas based on event updates as explained in section 2.4.5.

## 6.2 | Aztec Protocol

The AZTEC protocol (Williamson, 2018) is another layer 1 privacy solution sharing similar private token design traits but is unique in many ways. Private data handling is composed of homomorphic arithmetic and ZKPs. Instead of zkSNARKs, a different ZKP class is employed, range proofs.

Range proofs do not necessarily require a trusted setup. However, AZTEC still relies on a trusted setup, to improve efficiency and lower gas costs.

What really sets the AZTEC protocol apart is its architecture. It provides seven distinct operations that may be combined in various ways, facilitating the development of new privacy solutions. At its heart, **join-split** allows for joining the value of input notes and produce new output notes whose total balances out. Next the **mint** and **burn** operations allow for the creation and destruction of notes. Essentially these three operations cover the token mixing functionality.

Four more operations are also available. The **bilateral swap** allows swapping two asset types atomically. The **dividend** proof, verifies that an output note is the result of multiplying an input note by some interest rate. **Private range** compares two private assets, whereas **public range** compares a private asset to a public value.

Having these individual operations brings an interesting level of flexibility. Furthermore, the AZTEC Cryptography Engine integrates the ability to handle multiple private tokens within one application.

[3]EY Nightfall Timber [Accessed Oct 2021] – `https://github.com/EYBlockchain/timber`

More recently AZTEC released a new privacy solution, distinct from the AZTEC protocol. This is a layer 2 zk-zk-rollup solution that combines scalability and privacy. This project makes use of zkSNARKs and aims to allow the development of private layer 2 solutions. This project has mostly advanced in delivering the private zkDAI and zkWETH tokens. However, it promises to also provide support for custom privacy solutions.

## 6.3 | Comparison against Dwarna's Consent Solution

In this section the dissertation consent solution is compared to that provided by Dwarna.

| **Property** | **Dwarna's Solution** | **Dissertation's Solution** |
|---|---|---|
| GDPR Compliance | Dwarna focused on providing a GDPR compliant solution. This regulation is dependent on centralized data controllers and data processors, roles which Dwarna carefully fulfills. In practical terms Dwarna also satisfies the right to be forgotten. | This dissertation focused on decentralized privacy in full awareness that today regulators have not embraced decentralization. The solution does not prioritize compliance. Most notably all the consent solution data is immutable and does not satisfy the right to be forgotten. |
| Decentralized Governance | The biobank is an all comprising pole of trust. It is both the data custodian and the consent platform administrator. The biobank grants access to the consent platform and ensures its availability. This centrality allows the biobank to delete all off-chain data. This deletion is so effective that on-chain data cannot be linked back to users. | The biobank continues to be the data custodian and a pole of trust. Yet, various decentralization approaches help reducing its centrality. The auditor provides a counterbalancing pole of trust. Application and historical data availability is moved to independent node operators. Consent and termination workflows are autonomously initiated and recorded by research partners. |

| | | |
|---|---|---|
| Decentralized Architecture | Dwarna heavily relies on an off-chain WordPress frontend and PostgreSQL backend. Its architectural decentralization is typical of web2 solutions.<br>This renders any discussion of how decentralized the permissioned blockchain is, superfluous. | The solution directly inherits the architectural decentralization of Ethereum. Research partners access it through wallet applications running on their own devices.<br>One could argue that such solutions often rely on less decentralized web3 API providers. Even so, the proposed solution can achieve a much higher degree of architectural decentralization. |
| Accountability | As a regulatory compliant solution, research partners benefit from the rights granted by the GDPR, holding the biobank accountable in this respect.<br>Access to the consent history is administered by the biobank. Furthermore, the consent history becomes unusable once a partner terminates biobank participation.<br>This limits the research partner's autonomy in enforcing his rights and the span of time during which the consent history may be referenced. | The application directly integrates an active auditor providing supervision and a channel for redress.<br>Research partners access their historical data autonomously without any biobank involvement.<br>Historical data is retained indefinitely. Research partners can retain full access to their consent history even after terminating biobank participation. |
| User Experience | As a closed system, Dwana can focus resources in providing optimal user experience.<br>The lack of architectural decentralization makes it more vulnerable to denial-of-service. However, under normal conditions, optimal user experience is easier to achieve. | Properties like scalability and latency are a work-in-progress in the permissionless blockchain space.<br>Running on a permissionless blockchain, the consent solution will compete for resources with other applications. This impacts responsiveness. |

| | | |
|---|---|---|
| Fees | Implements zero fee transactions. | Fees are unfeasibly high and volatile requiring further work to both lower them and to transfer the cost away from research partners. |
| Privacy | Dwarna achieves its privacy goals through encryption and permissioned access. On-chain data is kept at a minimum. Its effectiveness in deleting identifiable data greatly reduces the risk of data exposure once a partner terminates biobank participation. | The dissertation achieves privacy through encryption and ZKPs. The entire consent history is on-chain and retained indefinitely. Thus, the data is indefinitely available for attack, also from vectors that are yet unknown. |
| Adoption | Dwarna relies on a well-known, mature design paradigm. A solution that is ready for immediate adoption. | The dissertation aims to contribute towards a new, fast evolving design paradigm. Most projects in this space, including this dissertation, are not adoption ready. |

Table 6.1: Comparison between the Dissertation's Consent Solution and Dwarna

# 7

# Conclusions

## 7.1 | Tackled Objectives

This dissertation aimed to contribute towards overcoming the challenges hindering the adoption of permissionless blockchains for privacy preserving solutions. Its first objective was that of tackling a problem for which a real-world need was already established. Dwarna fulfilled this need with a problem that was especially challenging because of two properties. The first was its mix of on-chain and off-chain elements. The two data silos had to be bridged whilst preserving privacy. The second key challenge arose from the intrinsic centrality of the biobank, making decentralization harder to achieve.

This objective was satisfied with a design providing the same core functionality as Dwarna. From the outset, privacy and decentralization were established as the guiding pillars. Privacy was tackled through private tokens and ZKPs. This involved repurposing and extending Zeth, inheriting its long research trail.

Decentralization was approached from multiple angles. The blockchain was employed as an independent source of trust. An auditor was introduced, counterbalancing the trust that the main service provider still retained. Additionally, workflows were redesigned to empower end-users.

The second objective required implementing the core design elements. This was satisfied through the development of five ZKPs of differing complexity. This implementation thus provided the basis for satisfying the outstanding objectives.

The third objective required measuring key application metrics, including storage requirements, processing time, and verification costs. This was fulfilled with results for both Zeth and the consent solution ZKPs.

The correlation between ZKP complexity, proving key sizes and proof generation times was immediately evident. Proving keys for PGHR13 were up to 50% larger than

those for Groth16. However, the two schemes only registered marginal differences in proof generation times. The largest consent system proving key was that for changing consents. Yet Zeth's proving key was even larger, registering a 58% increase.

On the verification side, differences across ZKP complexity was neutralized by a hashing wrapper, that led to identical key sizes and on-chain costs. However, across schemes PGHR13 inflated the key size by 97% and the gas cost by 220%.

A very interesting result was the one showing that ZKP verification costs are dwarfed by other smart contract support components. Indeed, Zeth's proof verification only accounted for 18% of the total. The total verification costs on Ethereum for November 2021 averaged at €590, whereas those for ZKP verification averaged at €104.

The fourth objective required producing a list of design challenges having general relevance. This was satisfied with a list focusing on various themes. Bridging off-chain and on-chain data silos can be even more challenging in privacy preserving solutions. Immutability amplifies the need for future-proof designs. Dealing with transaction spam becomes harder if users are relieved from paying for transaction costs. Tokenized privacy is harder to apply in workflows involving more than two parties.

Feasibility limitations and the technology learning curve today seem to be the biggest implementation hurdles. However, these are being addressed through scalability projects and toolsets that will eventually bring privacy solutions to a wider developer audience. A more elusive challenge is ensuring that a solution is tackling privacy holistically. Transforming real-world objects into private tokens does not guarantee optimal privacy. Privacy ultimately depends on usage behaviors that may only become evident following solution deployment.

## 7.2 | Critique and Limitations

This dissertation explored technologies that are still maturing, aiming to contribute to this fast-evolving process. Permissionless blockchains and NIZKs schemes are continuously improving, making the state-of-the-art a moving target. This unavoidably condemns the implementation and results to become obsolete. Yet the compiled list of design challenges is expected to outlive them, as its focus is broader.

This dissertation recognizes that disruptive technologies cannot be fully exploited if bottled within today's limitations. It favors a forward-looking approach without allowing today's regulatory frameworks to limit its scope. The result is a consent solution that is not GDPR compliant.

One key GDPR objective is that of keeping data controllers accountable. This dis-

sertation also prioritizes accountability. Its immutable consent history, whilst failing compliance, provides a permanent reference for enforcing accountability. Here the right to be forgotten can be seen to conflict with enforcing accountability, with the dissertation prioritizing the latter.

Taking a closer look at the presented solution, various limitations can be highlighted. Mixer-based privacy requires large anonymity sets. Thus, the solution is not suitable for small biobanks, with a limited choice of studies.

The running costs are too high. This limitation is even more relevant to the biobanking ecosystem which requires zero fee transactions. Closely related, is the problem with blocking transaction spam. This solution rejects processing of invalid transactions, but the possibility of resource wasting transactions still stands.

The solution ties a research partner's identity, to a single key pair. The identity token refreshing mechanism is valuable, but ultimately all tokens are tied to the same key pair. This is not ideal. A compromised key pair would expose all its resulting tokens. Additionally, transaction senders are not being obfuscated. Research partners are thus always using the same pseudonymous address, weakening their privacy.

Despite these limitations, this dissertation confirmed how mixers like Zeth offer a starting point for a wide array of solutions. Mixers are not an exclusive playground for money launderers. The technology has broad applicability, and one hopes that enforcement on specific applications does not hinder its advancement.

## 7.3 | Future Work

This dissertation scratched the surface of a broad topic and the scope for future work is extensive.

Obtaining more comprehensive data for verification costs is one direction for furhter work. Starting from Ethereum, one could estimate the cost variation over a longer time window. Furthermore, one should obtain fresh cost estimates once Ethereum switches to PoS. This would provide a direct comparison against its former PoW iteration. Beyond Ethereum, one should test the same case-study on alternative Ethereum Virtual Machine (EVM)-based blockchains. This may require additional implementation work since support for ZKP primitives is not part of the EVM standard.

Another work direction is one that utilizes a different implementation starting point. Beyond Zeth, Ethereum hosts another two fairly mature layer 1 private token projects. One should also look for similar projects on other blockchains. Implementation involves security primitives and a choice of libraries that are bound to produce different results.

For example, ZKPs could be developed using the Bellman rust library instead of the C++ libsnark library employed in this dissertation.

Investigating layer 2 solutions is another direction for future work. This dissertation highlights how a complimentary layer 2 solution should be considered to obfuscate senders, shifting transaction costs onto the biobank, creating a layer 2 token for curbing transaction spam, and also lowering the overall cost.

Alternatively, instead of adding a complimentary layer 2 solution, one may want to reimplement the solution on a layer 2 that directly supports running application specific ZKPs. This would shift the bulk of the solution onto layer 2, leaving layer 1 for rollups.

More work is also possible by revisiting certain design decisions. Each of these have an impact on privacy and resources. This dissertation provides an example of how research partners could have been allowed to express any consent choice instead of restricting them to a rigid alternating sequence. More complex design changes include allowing research partners to have multiple distinct identities and extending the consent system scope to comprise other applications beyond biobanking.

All these work directions are possible whilst retaining the same case study. However, to satisfy the aim of producing results having general applicability, more diverse case-studies need to be tackled. Today we have design patterns around implementing mixers and tokenized privacy. The next step is to have design patterns around applying them to a broad spectrum of real-world problems.

# A

# Complete ZKP Test Data

The following tables show a complete dump of the output returned by the libsnark API. The entries referring to G1 and G2 elements are included for completeness. A commentary on these elements, would require discussing the cryptography behind ZKPs, an aspect that this dissertation does not cover as it fully carries forward Zeth's design in this regard.

| **Groth16 Proving Key** | **Consent Minting** | **Consent Changing** | **Partner Termination** | **Consent Confirmation** | **Termination Confirmation** |
|---|---|---|---|---|---|
| G1 elements | 1,022,878 | 1,228,462 | 598,820 | 239,530 | 119,561 |
| Non-zero G1 elements | 1,004,936 | 1,204,719 | 588,346 | 235,870 | 118,022 |
| G2 elements | 253,580 | 322,108 | 155,918 | 58,000 | 28,933 |
| Non-zero G2 elements | 235,639 | 298,366 | 145,445 | 54,341 | 27,395 |
| Size (bits) | 391,279,763 | 478,166,999 | 233,368,151 | 91,284,179 | 45,792,881 |
| Size (KB) | 47,764 | 58,370 | 28,488 | 11,144 | 5,590 |

Table A.1: Complete Data - Groth16 Consent Proving Keys

| PGHR13 Proving Key | Consent Minting | Consent Changing | Partner Termination | Consent Confirmation | Termination Confirmation |
|---|---|---|---|---|---|
| G1 elements | 1,783,621 | 2,194,789 | 1,066,577 | 397,149 | 198,171 |
| Non-zero G1 elements | 1,628,673 | 1,998,228 | 971,423 | 357,223 | 179,216 |
| G2 elements | 253,579 | 322,107 | 155,917 | 57,999 | 28,932 |
| Non-zero G2 elements | 235,639 | 298,366 | 145,445 | 54,341 | 27,395 |
| Size (bits) | 578,406,618 | 716,211,314 | 348,300,402 | 128,492,554 | 64,543,335 |
| Size (KB) | 70,607 | 87,429 | 42,518 | 15,686 | 7,879 |

Table A.2: Complete Data - PGHR13 Consent Proving Keys

| Groth16 Proving Key | Zeth |
|---|---|
| G1 elements | 1,975,619 |
| Non-zero G1 elements | 1,942,900 |
| G2 elements | 483,779 |
| Non-zero G2 elements | 451,061 |
| Size (bits) | 753,897,389 |
| Size (KB) | 92,029 |

Table A.3: Complete Data - Groth16 Zeth Proving Key

| Verification Key | Zeth Groth16 ZKP | Consent Groth16 ZKPs | Consent PGHR13 ZKPs |
|---|---|---|---|
| G1 elements | 2 | 2 | 3 |
| G2 elements | 2 | 2 | 5 |
| Size (bits) | 1,847 | 1,847 | 3,629 |
| Size (bytes) | 231 | 231 | 454 |

Table A.4: Complete Data - Consent and Zeth Verification Keys

| **Proof** | **Zeth Groth16 ZKP** | **Consent Groth16 ZKPs** | **Consent PGHR13 ZKPs** |
|---|---:|---:|---:|
| G1 elements | 2 | 2 | 7 |
| G2 elements | 1 | 1 | 1 |
| Size (bits) | 1,019 | 1,019 | 2,294 |
| Size (bytes) | 128 | 128 | 287 |

Table A.5: Complete Data - Consent and Zeth Proofs

# References


Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. doi: 10.1109/SP.2014.36.

Béres, F., Seres, I. A., Benczúr, A. A., and Quintyne-Collins, M. Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 69–78, 2021. doi: 10.1109/DAPPS52256.2021.00013.

Eberhardt, J. and Tai, S. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091, 2018. doi: 10.1109/Cybermatics_2018.2018.00199.

European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), May 2016. URL `http://data.europa.eu/eli/reg/2016/679/oj`. Accessed Jul 2021.

Feng, Q., He, D., Zeadally, S., Khan, M. K., and Kumar, N. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, 126:45–58, 2019. ISSN 1084-8045. doi: 10.1016/j.jnca.2018.10.020.

Groth, J. On the Size of Pairing-Based Non-interactive Arguments. In Fischlin, M. and Coron, J.-S., editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-49896-5. doi: 10.1007/978-3-662-49896-5_11.

Groth, J. and Maller, M. Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs. In Katz, J. and Shacham, H., editors, *Advances in Cryptology – CRYPTO 2017*, pages 581–612, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63715-0. doi: 10.1007/978-3-319-63715-0_20.

Gurkan, K., Jie, K. W., and Whitehat, B. Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum, 2020. URL `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-semaphore.pdf`. Accessed Sep 2021.

Konda, C., Connor, M., Westland, D., Drouot, Q., and Brody, P. Nightfall, May 2019. URL `https://github.com/EYBlockchain/nightfall/blob/master/doc/whitepaper/nightfall-v1.pdf`. Accessed Jul 2021.

Lee, J., Choi, J., Kim, J., and Oh, H. SAVER: SNARK-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. Cryptology ePrint Archive, Paper 2019/1270, 2019. URL `https://eprint.iacr.org/2019/1270`. Accessed Aug 2021.

Mamo, N., Martin, G., Desira, M., Ellul, B., and Ebejer, J.-P. Dwarna: a blockchain solution for dynamic consent in biobanking. *European Journal of Human Genetics*, 28:1–18, 12 2019. doi: 10.1038/s41431-019-0560-9.

Miers, I., Garman, C., Green, M., and Rubin, A. D. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, 2013. doi: 10.1109/SP.2013.34.

Parno, B., Howell, J., Gentry, C., and Raykova, M. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, 2013. doi: 10.1109/SP.2013.47.

Petkus, M. Why and How zk-SNARK Works. *CoRR*, abs/1906.07221, 2019. doi: 10.48550/arXiv.1906.07221.

Reitwiessner, C. zkSNARKs in a nutshell, 2016. URL `https://chriseth.github.io/notes/articles/zksnarks/zksnarks.pdf`. Accessed Sep 2021.

Ron, D. and Shamir, A. Quantitative Analysis of the Full Bitcoin Transaction Graph. In Sadeghi, A.-R., editor, *Financial Cryptography and Data Security*, pages 6–24, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39884-1. doi: 10.1007/978-3-642-39884-1_2.

Rondelet, A. and Zajac, M. ZETH: On Integrating Zerocash on Ethereum. *CoRR*, abs/1904.00905, 2019. doi: 10.48550/arXiv.1904.00905.

Shamir, A. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov 1979. ISSN 0001-0782. doi: 10.1145/359168.359176.

Unterweger, A., Knirsch, F., Leixnering, C., and Engel, D. Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018. doi: 10.1109/NTMS.2018.8328739.

Williamson, Z. J. The AZTEC Protocol, Dec 2018. URL `https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf`. Accessed Jul 2021.

Wood, G. Ethereum: A secure decentralised generalised transaction ledger, 2014. URL `https://ethereum.github.io/yellowpaper/paper.pdf`. Accessed Sep 2021.

Zyskind, G., Nathan, O., and Pentland, A. S. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184, 2015. doi: 10.1109/SPW.2015.27.