Harmony Privacy Report

Common Prefix

May 2022, Version 1.1

1 Introduction

This report explores existing solutions for private transactions in blockchains, with a focus on their applicability for use on the Harmony blockchain. Harmony is a proof of stake blockchain with an account model for transactions. Additionally, it supports EVM-compatible smart contracts and features sharding to improve scaling. While there exists a plethora of solutions in the bibliography, most of them are defined in a setting that differs from Harmony's.

The aim of this report is to provide a short overview of these solutions and offer insight into the process of adapting them to Harmony. Section 2 provides some necessary background on the cryptographic tools and primitives that are used in there solutions. Sections 3 and 4 describe on-chain and smart contract based solutions respectively. Section 5 highlights a number of common challenges to be overcome when implementing and deploying a private transaction solution. Finally, 6 concludes the report with a number of recommendations for action.

2 Background

2.1 Homomorphic Commitments

Cryptographic commitment schemes allow a party to present an obscured view of some value to their counter party while guaranteeing they are unable to later change their mind. Concretely, we say that a commitment scheme is *hiding* if no adversary can determine if a given commitment c corresponds to message m_0 or m_1, even when the messages are chosen by the adversary. A commitment scheme is *binding* if it is infeasible for an adversary to prove a commitment c corresponds to two different messages m_0, m_1 even when given free choice over c, m_0, m_1.

A commitment scheme is homomorphic, if carrying out an operation (e.g. multiplication) in the commitment space corresponds to an operation on the message space. In applications, this allows for calculations to be performed over committed values without the need to reveal them.

2.1.1 Pedersen Commitments

Given a group $\mathbb{G}, g$ of order q, where the discrete log problem is presumed hard and an element h we commit to a message $m \in \mathbb{Z}_q$ as follows:

$$Com(m) : r \leftarrow \mathbb{Z}_q; c \leftarrow g^m h^r; \text{Return } (c, r)$$

The commitment c can be used as a representation of m, whereas the *randomizer* or *nullifier* r is used as proof that c indeed corresponds to m. For ease of notation, we will also use $Com(m, r)$ to refer to $c = g^m h^r$.

$$Ver(c, m, r) : \text{ if } c = g^m h^r \text{ return 1 else return 0}$$

The binding property holds computationally for the Pedersen commitment scheme (i.e., an adversary who is able to solve discrete logs would be able to open a single commitment in multiple ways), however the hiding property holds unconditionally. Even with unlimited computational resources an adversary cannot recover the commited message: for a given commitment c there exists an opening for every possible message.

The pedersen commitment scheme is additively homomorphic, so that multiplication between commitments corresponds to addition modulo q between the commited values and randomizers.

$$Com(m, r) \cdot Com(k, s) = Com(m + k, r + s)$$

2.2 Homomorphic Encryption

Whereas commitments intend to obscure a value so that the sender is later able to reveal it, encryption intends to obscure it so that only the recipient is able to reveal it. Public key encryption operates by allowing each receiver to generate a key pair ek, dk so that senders are able to encrypt messages for them using only ek which they are, then, able to open using dk.

Encryption schemes are related to commitments: one may consider a public key encryption scheme as a commitment by allowing the encrypting party to save and reuse the corresponding randomness. Uniqueness of decryption implies perfect binding, otherwise encrypted messages would be ambiguous. On the other hand, the (computationally prohibitive) possibility[1] of an adversary deriving dk from ek implies that hiding can only be computational. Constructions such as Zether (Sect. 4.3) and Quisquis (Sect. 3.6) are examples where the requirement for users to decrypt their balances necessitates the use of encryption-based commitments.

[1] A trivial idea would be to iterate over all the possible random choices of the key generator and keep the output that produces ek as the encryption key. The corresponding decryption *key* must be effectively unique, as otherwise decryption would be ambiguous.

2.2.1 Lifted ElGamal Encryption

This popular variant of the ElGamal encryption scheme is additively homomorphic at the cost of introducing potential inefficiencies in decryption. It operates in a group $\mathbb{G}, g$ of order q, where the Decisional Diffie-Hellman (DDH) problem is presumed hard, and has message and randomizer space $\mathcal{M}, \mathcal{R} = \mathbb{Z}_q$. The DDH problem in a group $(\mathbb{G}, g, q)$ involves distinguishing between random triples of the form g^a, g^b, g^c and triples of the form g^a, g^b, g^{ab} (DDH triples).

$\mathsf{Gen}()$: Let $dk \leftarrow \mathbb{Z}_q; ek \leftarrow g^{dk}$, Return (dk, ek)

$\mathsf{Enc}(ek, m)$: Let $r \leftarrow \mathbb{Z}_q; u \leftarrow g^r; v \leftarrow ek^r g^m$, Return $c = (u, v)$

$\mathsf{Dec}(c, dk)$: Let $(u, v) = s$, let $w \leftarrow v \cdot u^{-dk}$, Return $m = \log_g(w)$.

It is clear from the above that the textbook decryption of lifted ElGamal is not tractable in the general case, as we need to calculate a discrete logarithm. Under the assumption that the message m is bounded, e.g. a n-bit integer, decryption becomes tractable (with $O(2^{\sqrt{n}})$ time and space cost). It is also possible to verify a given value of m from an external source against the w produced via decryption. Thus, the inefficiency of decryption can often be avoided (cf. Section 4.3).

2.3 Collision Resistant Hash Functions

A hash function h is collision resistant if it is infeasible for an adversary to produce x_1, x_2 such that $h(x_1) = h(x_2)$. This is equivalent to the binding property of a commitment, and implies we can use collision resistant hash functions as binding commitments with no modifications. A hash function is one-way if it is infeasible to derive x^* from $h(x)$ such that $h(x^*) = h(x)$ (x^* itself may or may not equal x). Without adding some source of randomness, this is strictly weaker than the (similar) hiding property of a commitment: given m_1, m_2 one can determine if $c = h(m_1)$ or $c = h(m_2)$ even if h is one-way.

2.4 Merkle Trees

A Merkle Tree is a data structure that allows one to commit to a large number of independent values $v_1, \ldots, v_N$ of arbitrary size so that:

- The commitment will be constant size.
- The size of the proof[2] that v_i is contained in c will be logarithmic in N.

For simplicity, we assume that the maximum size of the tree is $N = 2^n$ and that it is known in advance. In the general case, we can ensure this by padding with an appropriate vector of null values. We then have:

[2] discounting the size of v_i itself.

- $MTCom(v_0, \ldots v_{k-1})$:
  - Assign item v_i to each of the first k leaves of the tree. Assign a default value $v_{def} = 0$ to all other leaves. If $k > N$, abort.
  - Label each leaf i with the hash of its assigned value $h(v_i)$.
  - Recursively label each internal node of the tree with $h(L||R)$ where L, R are the labels of its left and right child respectively.
  - Let T be the label of the root, return T.
- $MTVer(T, v, i, \mathbf{p})$: Checks that value v is contained in index i of a tree with root T along audit path $\mathbf{p} = \mathbf{p_j}$.
  - Let d_j be the j-th digit of i in binary, d_0 being the LSB, let $\bar{d_j} = 1 - d_j$.
  - $h_0 \leftarrow h(v)$
  - For $i = 1$ to n:
    $h_i \leftarrow h(p_i \cdot d_i + h_{i-1} \cdot \bar{d_i} || p_i \cdot \bar{d_i} + h_{i-1} \cdot d_i)$
  - If $h_n = T$, Return 1, else Return 0.

Verification consists of the value that is claimed to be present in the tree, its index, and the labels of nodes in the complement of the path from the indexed leaf to the root.

2.4.1 Updating Merkle Trees

In our context we will require appending values to an existing tree, but not removing them (or updating them). A simple approach to appending would be to recreate the entire tree, but that would imply that (a) the entirety of the tree contents must be made available during the update and (b) the cost will be linear in the number of items already present in the tree. Fortunately, it is simple to define a small set of data that can be used to speed up updates so that their cost is similar to that of verification.

For a tree that contains k items, consider the audit path and labels for the first empty leaf, with index k and value $v_k = v_{def} = 0$. If we populate leaf k by setting its value to $v_k \neq 0$, the audit path is sufficient to recompute the new root. In this way, we can use the audit path of the first empty leaf as a cache: we can then update the root using the cache at the same cost as verification. What remains is to update the cache for the next insertion: this is straightforward as the labels of the new audit path will either be values we have just derived or defaults (i.e., hashes of subtrees containing only zeroes).

2.4.2 Proving Non-Inclusion

Standard Merkle Trees can produce very efficient membership proofs, but not proofs of non-membership. For that, we may use either sorted or sparse Merkle Trees.

Using a sorted Merkle Tree, we can demonstrate that value v is not present by proving that there exist values $s < v$ and $t > v$ that *are* present in leaves with consecutive indexes. Because the tree is sorted, there is no valid index for v, proving non-inclusion. The difficulty with sorted Merkle Trees lies in the complexity of updating them: efficient techniques make the tree depth non-constant which in turn makes circuit representations of tree operations much more complex.

A simpler approach is to use a (larger) sparse Merkle tree with the convention that a leaf with index i has label h(0) if value i is not present and label h(1) if it is.

2.4.3 Sparse Merkle Trees

Compared to a standard Merkle Tree, a sparse Merkle tree [DPP16; Öst16] is larger as it needs to account for every possible entry (rather than a maximum number of arbitrary entries), and it also needs to account for updates to arbitrary indexes (rather than only sequentially).

During the initial construction, the size of the tree can be addressed by caching default values: an empty tree has identical internal nodes across each level, so the effort to initialize a Sparse Merkle tree is linear to its depth.

Caching default values can also be used when constructing audit proofs by encoding them in a succinct way. However, this approach has limited benefits when used with circuit-based verification.

2.4.4 Updating Sparse Merkle Trees

Tree updates can still be performed efficiently by using a cache, but the size and structure of it are larger and more complex. Instead of needing to cache at most 1 non-default value per level, we will need to cache one value per branch. We say that an internal node is a branch if the labels of both its parents are non-default. The results of Östersjö [Öst16] show that caching the tree contents, default values and branch labels is sufficient to obtain good update performance. This makes sparse Merkle trees expensive to maintain storage-wise from the prover side, while still providing adequate verifier performance and acceptable proof size.

2.5 Digital Signatures

A digital signature enables a signer to certify that a message m has in fact been approved by her. Similar to public key encryption, a signature scheme uses a public-private keypair and three algorithms $\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}$ for key generation, signing and signature verification.

A secure digital signature must be *correct* (verification should succeed for properly signed messages) and *unforgeable*, i.e., an adversary must be unable to produce verifying signatures w.r.t. honest parties apart from copies of already-produced signatures. The allowance of replaying past signatures in the definition

of unforgeability is natural, but at the same time implies that replay attacks must be handled at a higher level: in usual blockchain applications replaying an existing signature would be considered a double-spending attack.

2.5.1 Schnorr Signatures

Schnorr signatures are based on the eponymous identification scheme: given a public key of the form $k = g^x$, the signer needs to prove knowledge of x (i.e., the discrete log of h) to the verifier. This is performed via a zero knowledge protocol so that the value of x is not leaked. Schnorr's protocol can be used as an interactive sigma protocol (Sect. 2.6.1), but in practice it is used non-interactively via the Fiat-Shamir transformation. The same transformation enables its use as a signature scheme (as opposed to an identification scheme) by including the message to be signed in the protocol statement.

Another common use of Schnorr's protocol is to demonstrate knowledge that a particular Pedersen commitment has a value of zero, i.e., it can be written as $c = h^r$ for a value r known to the prover.

2.5.2 Ring Signatures

Generalising standard digital signatures, group and ring signatures allow a member of a group to sign on behalf of the entire group, without disclosing their individual identity. In a group signature, membership to the group is controlled by a group manager who is responsible for, e.g., provisioning credentials or managing a membership roster. Ring signatures on the other hand operate without a fixed authority: any collection of public keys can be used as a Ring. This allows a user to freely choose a set of other user keys to "hide" amongst, enabling privacy with decentralization.

A simple way to obtain ring signatures from Schnorr signatures is to directly use the "OR" construction of CDS [CDS94] to prove that one knows the secret key of one public key out of a list of public keys, without revealing the witness. The "OR" construction runs a separate instance of the Schnorr protocol for each list element, but modifies the challenge: a single joint challenge c is produced with the additional requirement that the challenges c_i used in each instance must sum to the joint challenge c. Effectively, this gives the prover freedom to freely choose every challenge but one. Thus, the prover simulates every instance that does not correspond to their key, calculates the remainder challenge, and completes the instance corresponding to their key normally. The zero knowledge property of the underlying scheme implies that the simulated instances are indistinguishable from the real one.

While the above construction is straightforward, its efficiency is lacking: running a separate instance for each key requires 2 elements for each ring member. Improvements in the form of the scheme of Abe et al. [AOS02] and Borromean signatures [MP15] can reduce this cost to 1 element per member, but do not address the linear relation.

Subsequent protocols starting with the work of Groth and Kohlweiss [GK15] reduce the size cost to logarithmic in the ring size. In [GK15] and [Boo+15] this is done by committing to a digit representation of the index of the user's key, and proving knowledge of the logarithm of the indexed key, by providing a blinded opening of the commitments and error terms to account for the blinding. The communication cost is thus made linear to the number of digits of the group size, i.e., logarithmic in the group size.

Later work for RingCT 3.0 [Yue+20] achieves similar efficiency with a different formulation: they identify that for a given list of keys $Y = y_1 \dots y_n$, the key with index k is $y_k = \mathbf{Y}^{\mathbf{v_k}}$, where $\mathbf{v_k}$ is a vector of n bits where only position k is 1. Thus, their protocols relies on committing a vector, and proving the necessary properties via an efficient logarithmic argument.

2.5.3 Linkable Ring Signatures

When ring signatures are used to authorize payments, a natural question arises with regards to double spending. When producing single party signatures, it is trivial to determine if a key has been used more than once. In a ring signature setting however, this is not trivial. Linkable ring signatures provide a way for signatures of the same user across different keys to be identifiable either in general, or in a specific context. This can be accomplished by specifying that the signer must attach a unique function of their key to their signatures.

In the protocol of Groth and Kohlweiss this is achieved by altering the items of the list: instead of Schnorr keys of the form g^x, the list items are "coins" of the form $g^x h^r$, i.e., Pedersen commitments. Their key observation, is that the h^r term can be safely removed by revealing r and dividing $C = g^x h^r$ by h^r. Even better, this can be performed without revealing the index, by simply revealing the nonce r, and dividing each element of the list by h^r. The protocol can now proceed as above, with the caveat that the prover will have to reveal r again if they try to sign a second time. This is easily detectable, and is unavoidable unless the prover knows a non-trivial discrete log relation between bases g and h.

Other schemes [Sun+17; Yue+20] achieve linkability by means of a key image that is embedded in the signature computation.

2.6 Zero Knowledge Protocols

As a motivating example, consider Confidential Transactions [Maxb]. In a confidential transaction, the values being exchanged are hidden in the form of Pedersen commitments. Whereas it is simple to show that a set of values in the open balance by calculating the sum of inputs and outputs, this is made non-trivial when some of the values are hidden: First, one needs to prove that the sum of inputs minus the sum of outputs is in fact a commitment with value 0. This can be accomplished without zero knowledge by revealing the randomizer of the sum, or by using Schnorr signatures (e.g. in Mimblewimble [Jed16] and related constructions). Second, one needs to prove that none of the outputs

have negative values. Otherwise, Alice could buy a \$1000 laptop by spending a \$1 input and adding a −\$999 change output that she never intends to use again. The transaction would balance out as $1 = 1000 + (-999)$, but should in fact be rejected.

As arithmetic is performed modulo q, these would not be negative values per se, but by convention; for instance, a natural one would be that values over $\frac{q-1}{2}$ are defined to be negative. In practice, we would solve this issue via a range proof: show that for each output commitment c we know of v, r such that $c = \mathsf{Com}(v, r) \wedge v \in [0, V_{\max}]$, where $V_{\max}$ is the maximum allowable value in the application. Of course, we need to accomplish this in zero knowledge i.e., without revealing anything about v and r other than satisfying the above requirement.

Zero Knowledge Protocols were originally introduced by Micali, Goldwasser, and Rackoff [GMR89] and allow one party (the prover) to demonstrate that a statement is true without revealing any information other than the validity of said statement. More formally, for an agreed-upon relation $R(x, w)$ where R is an efficiently calculable function, the prover discloses a statement x and proves that there exists (or, in stronger variant that they know of) a witness w such that $R(x, w)$ is true.

The standard security properties of zero knowledge protocols are (a) completeness, (b) soundness, and (c) zero knowledge. Completeness demands that the protocol is successful if both parties are honest and the prover knows a correct witness w for which $R(x, w) = 1$. Soundness implies that provers are unable to prove false statements. In some settings we require a stronger property, knowledge soundness, which requires that if a prover is successful in proving statement x, then not only is x true, i.e., a correct witness w *exists*, but also that the prover effectively "knows" said witness. Returning to the CT example, for any Pedersen commitment c there *exist* witnesses such that c can be opened to any value[3]. However, calculating or "knowing" such a witness is non-trivial: either the prover created a commitment to known values (which is the intended result of the protocol) or they are in a position to derive discrete logarithms between g and h in the commitment scheme. Zero knowledge implies that the protocol can be simulated under certain conditions which in turn implies that transcripts of the protocol's execution cannot reveal the witness to an observer. In other words, the verifier does not learn any information apart from the existence of the witness (or the prover's knowledge thereof).

Finally, for our applications we require that protocols can operate non-interactively: otherwise the prover and verifier need to be online simultaneously in order to execute the protocol. Furthermore, unless one verifier trusts another, the protocol must be executed separately for each verifier, which is not feasible for our setting.

[3]But knowing of more than one opening is computationally infeasible in a proper setup.

2.6.1 Sigma protocols

A common class of zero knowledge protocols is the family of Sigma (and sigma-like) protocols. Their main characteristic is that all verifier messages are specified to be uniformly random with the only non-trivial verification step being in the final step.

This allows the protocols to be made non-interactive in the random oracle model [BR93] by using the Fiat-Shamir transformation [FS87]. This replaces the verifier's messages with hashes of the protocols transcript up to that point, effectively collapsing the entirety of the protocol into one round.

Canonically, sigma protocols have three rounds, zero knowledge against honest verifiers[4] (HVZK) and a variant of knowledge soundness termed special soundness.

Schnorr's protocol for demonstrating knowledge of a discrete logarithm (cf. Sect. 2.5.1) are one of the most common examples of sigma protocols. The zero-knowledge property is necessary to maintain security against adversaries who have access to past signatures. A more complex example is the membership protocol (and resulting ring signature) of Groth and Kohlweiss [GK15].

2.6.2 Bulletproofs

Bulletproofs [Bün+18] are a powerful proof system which can be used to perform proofs of arbitrary calculations expressed as an arithmetic circuit, but can also be highly efficient to perform range proofs of Pedersen commitments.

Bulletproofs have size logarithmic w.r.t to the witness and do not require a trusted setup. However, the verifier's computation is considerable as it scales linearly. This makes bulletproof performance adequate when used in native code, but potentially problematic if used inside a smart contract.

2.6.3 SNARKs

Succinct non-interactive arguments of knowledge add succinctness to the already introduced properties of non-interaction and knowledge soundness. Succinctness requires that (a) the size of the proof is at most logarithmic in respect to the size of the computation (modelled as a circuit or constraint system) (b) the speed of verification is logarithmic in respect to the size of the computation and at most linear in respect to the public statement[5].

To achieve such efficiency most SNARKS rely on structured reference strings (SRS), that is, carefully pre-constructed elements that are used when constructing and verifying the proof. Because they need to be structured (i.e. the various elements must be related to each other in a particular way) they need to be constructed rather that simply sampled at random. At the same time, the

[4] This is rendered moot via the Fiat-Shamir transformation as the verifier‘s messages are determined via the hash function.

[5] This is the motivation behind some applications structuring public data in the witness, and checking for consistency against a known hash.

randomness used in their construction is effectively a trapdoor that allows simulating proofs without knowing a witness. In this sense, the existence of the trapdoor is a double edged sword: it is used to prove zero knowledge (under the assumption that the simulator knows it) but in real-world use we must ensure that the adversary never learns it.

Groth16 [Gro16] is currently the most-used SNARK due to its efficiency with regards to its small proof size, and secondly, its limited verifier computation.

SRS Setup. In order to ensure that an adversary cannot learn the trapdoor, we must use multi-party computation protocols [BGM17; Koh+21] to divide the computation of the SRS amongst multiple parties in such a way that learning the trapdoor necessitated corrupting all involved parties. This implies that the SRS is secure as long as at least one participant was honest. Furthermore, any participant in the MPC ceremony would have reason to trust the SRS unless they knowingly exposed the randomness they uses or they have reason to believe they were compromised.

Updateable Setup. A SNARK has updateable [Mal+19] (or semi-trusted) setup if it is possible for a party to contribute entropy to the SRS even after it has been produced, producing an updated version (along with a proof of correctness for their contribution). Concrete examples of such systems include Sonic and PLONK.

Contrary to what the name implies, the main benefit of the property is that the initial (i.e., before production use) ceremony can be performed with less coordination requirements.

Of course, updating the SRS can also be used to allow a new party to establish trust in the proof system by being allowed to contribute to it. This contribution however is not retroactive: while the new party will have reason to trust proofs using the updated SRS for future proofs (because the proofs now also rely on the entropy they personally supplied), the trust assumptions for proofs in the past remain unchanged. Furthermore, it is hard to scale SRS updates so that they can be performed by any interested party: the updates as well as the accompanying proofs will need to be transmitted to all nodes in the system.

2.6.4 STARKs

STARKS (Scalable Transparent Arguments of Knowledge) [Ben+18] offer a trasparent alternative to SNARKS. While asymptotically efficient, and highly practical in terms of verification time, proof sizes are prohibitive with regards to direct blockchain inclusion, especially in the case of a smart contract which would involve additional overheads.

2.6.5 Recursive Proof Systems

The works of [Val08],[Bit+13; BGH19] present techniques that can be used to enable a proof system to model its own verification circuit. This in turn enables one to construct proofs with statements that include the validity of other proofs. In this manner a complex statement can be broken into a number of incremental sub-statements where each subsequent sub-statement consists of (1) a part of the original statement and (2) a claim that there exists a proof of the previous sub-statement. The result of this re-organization is that the complexity of the last statement is equal to that of the last part plus the complexity of the verification circuit. Thus, if the original statement is amenable to being subdivided into parts, the verification cost becomes independent of the complexity of the statement and is mainly bounded by the complexity of the verification circuit. In the domain of private transactions, many operations are small enough that recursive techniques offer no benefit. For this reason, we leave the discussion of recursive proofs as a potential optimization for a later stage.

3 On-Chain Solutions

3.1 Monero

Monero is a privacy focused blockchain that makes use of ring signatures. In the initial design, based on the Cryptonote protocol [Van13], privacy was established by using a linkable ring signature (Sec. 2.5.3) over a set of UTXOs of the same value for each input of a transaction. In this way, the actual inputs used to fund a transaction remain obscured, while the linkability property of the ring signature prevents double spending. This initial design has two main drawbacks: first, transaction amounts are public. This implies that privacy can only be established amongst transactions of the same value. Second, the efficiency of the signature scheme limits the size of the ring, and by extension the anonymity set of each input. The initial scheme was based on the traceable ring signature of Fujisaki adn Suzuki [FS07] which has linear size, this led to small ring sizes and a fragile notion of privacy: if a number of ring elements of a particular transaction can be attributed to specific users via de-anonymization techniques (see Sect. 5.3), the ring may collapse to a single element, providing a concrete link of input and output. Furthermore, this link can cascade into later transactions, eroding their privacy as well.

Later iterations of the protocol address both issues: the later RingCT [Noe15] construction uses signatures which are still linear in size with respect to the number of elements but allow for transaction values to be hidden. Furthermore, with time, the ring size was increased [Wu+22] to 10 elements to increase privacy and resilience to side channel attacks. However, this increase in the ring size can still be overcome, especially in transactions with multiple inputs: as the input and output amounts must be shown to balance, the protocol necessitates that the actual inputs to the transaction share the same index in their respective rings. Thus, if, for instance, the third element of the first input is established

to have been spent in a different transaction, the third element of each other input ring can also be disregarded by the adversary. Equivalently, if a single input is successfully linked, all other inputs will be linked as well. In the more recent RingCT 3.0 design [Yue+20], the indexes over different rings are allowed to differ and the signature size is made logarithmic with regards to the size of the ring using techniques derived from Bulletproofs, allowing for much larger ring sizes. The computational cost for the verifier also be improved by using multiexponentiation techniques to go from linear in the number of ring members to $O(\frac{n}{\log n})$.

3.2 Zerocoin

Zerocoin augments the design of bitcoin with a parallel ledger containing "coins" of a fixed denomination. Coins are minted, i.e., created by destroying an equivalent value in the original ledger. A coin is represented as cryptographic commitments to a serial number. When a coin is to be spent, its serial number is revealed and a zero knowledge proof is provided demonstrating that a commitment to the revealed value exists amongst the coin commitments in the parallel ledger. As the zero knowledge proof does not indicate *which* commitment is being spent, the link between the depositor and withdrawer of a coin is broken. At the same time, this precludes pruning the ledger of spent coins. Zerocoin addresses this by storing the commitments in a cryptographic accumulator, a construction that enables a set of values to be represented via a single object and that supports efficient proofs of membership. As accumulator proofs do not hide the value that is being proven, zero knowledge proofs need to be leveraged on top of them in order to facilitate the design.

3.3 Zerocash/Zcash

Zerocash [Sas+14], implemented as ZCash [Hop+] expands on Zerocoin by allowing for commitments to contain arbitrary quantities of funds as opposed to a fixed denomination, and also for transactions between committed values (i.e., joining and splitting existing commitments into new ones). The second change is necessary to maintain privacy when commitments are not fixed-value: otherwise, when users are allowed to mint a note of any value, the anonymity set for the corresponding withdrawal is limited to the set of mints of exactly the same value. By allowing notes to be spent in part, this issue is completely avoided. A further consequence of this is that transactions that create commitments must also prove that the value of such commitments is positive[6].

Due to the complexity of the statements being proven, ZCash opts to use a ZK-SNARK based proof system to reduce the overhead imposed by transmitting and verifying the accompanying proofs. ZCash also uses a Merkle tree instead of an accumulator to store the commitments. This reduces both the cost of

[6]It is not enough that the sums of the inputs equal the sums of outputs: one could simply create a negatively-valued output to force an underfunded transaction to balance

storing the accumulator itself (i.e., the tree root) as well as the computational cost of updating a membership witness when new values are added.

In addition to allowing join/split transactions Zcash also facilitates transaction discovery. This is performed by adding a short memo field to transactions so that the sender can add information retrievable by the sender's long term keys.

3.4 CoinJoin

CoinJoin, initially described by Maxwell [Maxa] notes that bitcoin transactions need not be generated by a single entity: instead, a number of users may present each other with a list of inputs and outputs. After agreeing on both lists, each user signs the combined transaction with regard to the input UTXOs they control. While this ensures privacy against outsiders, it is completely linkable to members of the group. A potential solution, described in the initial design, is to use a semi-trusted server to ensure coordination (so that the link can only be established by the server), and a further improvement is to use blind signatures to enable members to submit their outputs separately from their inputs. The blind signature mechanism requires that the submitted inputs and outputs match in value. This can be accomplished by either fixing the transacted denomination, or by using more complex cryptographic primitives as in WabiSabi [Fic+21]. Other variants such as Coinshuffle [RMK14] eschew the use of a centralized server at the cost of higher user to user communication.

3.5 MimbleWimble

MimbleWimble [Poe16] is to some degree a continuation of the CoinJoin concept with the addition of Confidential Transactions (i.e., values are represented as Pedersen commitments rather than in the open), and a change to the signing mechanism. Rather than signing individual inputs, the protocol leverages the homomorphic nature of Pedersen commitments: if the values of the inputs and outputs balance[7], the difference between the sum of input commitments and output commitments must be a commitment with value zero (or equivalently, the difference minus the remainder commitment must be the identity in the group). The transaction is then "signed" by proving knowledge of the randomness contained in the remainder. By relaxing the transaction definitions to allow for multiple signed remainders transactions can be merged by simply merging their inputs, outputs and remainders.

The end result is that block producers can operate as CoinJoin coordinators with no additional overhead. While this achieves some level of privacy, some implementations of MimbleWimble opt to add an additional level of privacy via a zero knowledge membership protocol [CG21; Jiv19].

[7] We also require that outputs are positive, necessitating range proofs.

3.6 Quisquis

Quisquis [Fau+19] is an anonymous cryptocurrency design in the account model. It relies heavily on the concept of updateable keys: this allows other users to effectively re-randomize the keys (and balance encryptions) of other users without changing the associated secret key. This forms the basis of the scheme: each transaction includes a set of $\mathsf{key}, \mathsf{balance}$) pairs, where key is of the form $g_i, h_i^{x_i}$ and $\mathsf{balance}$ is an ElGamal ciphertext under (key. Key updates occur by raising all elements of the ($\mathsf{key}, \mathsf{balance}$) pair to the same random exponent r. By including the base element g_i in the key, this ensures that the value stored in $\mathsf{balance}$ is unchanged. Anonymous transactions are facilitated by including a number of "dummy" ($\mathsf{key}, \mathsf{balance}$) pairs in addition to those of the sender and recipient. The sender then updates all keypairs, and transfers value v from their account to the recipients. Also, the sender constructs a zero knowledge proof that the updates and transfers were well formed, and that the initial balance in their account is at least v. The recipient (as well as the other involved parties) identify the transaction via their public key, derive their new public key via exhaustive search and update their public key and balance as needed.

A significant efficiency benefit of Quisquis is its use of the account model, limiting the state of its ledger to only the current keys of users and their balance. However, the short-lived nature of Quisquis addresses creates issues with concurrent transactions, as a key used by a prepared transaction might have been updated in the meantime. Futhermore, users need to spend effort in updating their key since the prescribed method of updating requires the secret key and thus cannot be easily outsourced.

4 Smart Contracts

4.1 Möbius

Möbius [MM18] is a minimal, trustless mixer with anonymous payment functionality. That is, it allows Alice (in conjunction with Bob) to perform a deposit that can be withdrawn by Bob, but not Alice. Möbius uses a fixed denomination for all deposits, as well as a preset, fixed ring size representing the number of deposits allowed before the ring is considered full. Withdrawals are not permitted before the ring has been filled. Once the ring is full, withdrawals are performed via a linkable link signature, allowing each deposit to be withdrawn only once. The scheme uses the signature scheme of Franklin and Zhang [FZ13], but other options may be leveraged.

The scheme does not rely on a trusted setup, but as a consequence, requires size and computation linear to the fixed size of the ring. For a ring of size n, the originally published version quotes a cost of $335,714n$ gas for each withdrawal transaction, but pre-dates EIP-1108 which made vast improvements to ECC operations. A more recent implementation available at `https://github.com/clearmatics/mobius` quotes 725k for $n = 4$ while still predating EIP-1108. The verification of Möbius involves approx $4n$ point-scalar multiplications, so

the benefits of EIP-1108 would be approximately $4n*(40.000-6.000)$ or, almost a 75% reduction in gas. While the asymptotics of Möbius remain unfavourable, for (very) small rings it can outcompete SNARK-based solutions.

Another point of interest on Möbius is that of liveliness and capacity: as it has a fixed capacity of ring members there needs to be a mandated solution when a ring has failed to fill past a certain date: the paper suggests offering refunds or (as an alternative) allowing a reduced ring size. At the same time, the capacity of a single ring is very limited, suggesting the use of multiple copies of the contract or separate ring-spaces within a single contract. Ideally, rings that have been fully withdrawn can be reused by allowing them to be re-initialised. A complication is possible when a filled ring has been withdrawn only partly: it might be beneficial to mandate that deposits must be withdrawn within a fixed period after the last deposit, as this will ensure that rings are emptied out in a timely manner. Note however, that it is not feasible to return unclaimed deposits after withdrawals have started; the contract does not know which deposits have been spent and which have not, therefore the deposits need to be claimed by the recipient.

4.2 TornadoCash

TornadoCash [Aleb] is a smart-contract based solution that offers the equivalent of fixed-denomination mixing. Deposited funds are represented as hash-based commitments, each containing a two-part nullifier, stored in an append-only Merkle Tree, similar to Zerocoin modulo differences in function choices. Odd sums are handled by splitting them into the fixed denominations supported. This is somewhat conductive to privacy, as the protocol implies a small number of large anonymity sets. However, as each denomination can be considered a separate instance of the protocol, transactions in one denomination do not increase the anonymity pools of others.

When funds are to be withdrawn, a zero-knowledge proof indicates the commitment that is being spent, along with a publicly-posted hash of one part of the nullifier to prevent double-spending. The zero-knowledge proof is bound to the withdrawing address, and since it does not reveal the full nullifier, front-running attacks are not possible.

The design of TornadoCash has been audited [KV] and implemented [Alea]. The main complexities lie in designing the Groth16 verification circuit as well as handling nullifier storage (see section 5.2). An additional concern involves concurrency: as deposits will change the Merkle tree holding the deposited commitments, the contract needs to maintain a buffer of previous roots so that a withdrawal request is not invalidated because a deposit was executed just before.

4.2.1 TornadoCash Nova

Nova represents a significant enhancement to the design of TornadoCash, allowing for significantly more complex functionality, i.e., it supports arbitrary

(confidential) denominations and internal transactions. Effectively, Nova is to standard TornadoCash what ZCash is to Zerocoin. While, Nova is not as well documented as standard TornadoCash, many of the design patterns can be traced to ZCash which is well documented.

4.3 Zether

Zether is a confidential payment system designed to be compatible with the Ethereum Virtual Machine (EVM). In effect, it implements a simple account-model ledger inside a smart contract with the key difference being that balances are stored encrypted, in a parallel to confidential transactions. However, since values are *encrypted* rather than *commited*, there is no additional requirement for recipients to be able to use funds. This is in contrast to commitment-based protocols where the randomness used must either be communicated to the recipient or derived in a system akin to key agreement (as in ZCash). This observation is a key pillar of the system's operation, enabling use of the account (rather than the UTXO) model. In the UTXO model, there is no practical way to inconvenience another user by sending them unusable funds: the term itself is paradoxical. If funds cannot be used by a user, they can be safely ignored. In a hypothetical commitment-based accounting model, adding a spurious commitment to one's balance might make the total unusable. While safeguards could be implemented by requiring deposits to be manually accepted or by requiring proofs of deposits being derived in a particular fashion as in Quisquis, Zether's approach offers an elegant solution to the problem. By using encryption to represent balance, all that is needed is to demonstrate that balances sent to users are not negative.

Zether uses the concept of "epochs" to effectively freeze the balance of parties for fixed periods of time. This introduces latency but at the same time avoids failed transactions due to front-running or race conditions. To achieve anonymity, Zether uses a modified membership proof: an anonymous transaction consists of two non-zero balances that sum to zero, representing the transaction input and output as well as a number of zero-valued encryptions to serve as an anonymity set. Each encryption is addressed to a different account. Additionally, the transaction includes a specifically constructed nonce. The protocol then proves that (1) only two encryptions are non-zero, (2) all encryptions sum to zero, (3) the account indexed by the negative-valued encryption holds balance greater than the absolute value of the encryption, and (4) the attached nonce is derived from the secret key of the indexed account. The cryptographic protocols are quite involved, with the authors quoting upwards of 1M gas (taking EIP-1108 into account) for transfers before adding anonymity overhead. A later implementation [Dia21] including anonymous transfers quotes 5.6M for an anonymity set of 4.

Zether also supports the concept of "locking" accounts to smart contracts so that for each account a smart contract address can be allowed to issue transactions exclusively. Account locking can be used to enable escrow or deposit functionality without directly revealing the amount at play. Account locking

is only partially compatible with anonymous transactions, but Zether also supports transparent (i.e. non-anonymous) ones as well.

4.4 Mixeth

Mixeth is a solution that relies on cryptographic shuffles [BG12; Nef01] to ensure privacy akin to a traditional mixnet, as described by Chaum [Cha81]. At a high level, the design consists of fixed-value deposits, with each deposit linked to a public key. To remove the link between depositors and keys, a number of shuffling (i.e. mixing) rounds is performed. In each round, the keys are partly re-randomized and their order is permuted. A key characteristic of Mixeth is that the cryptographic proof of the mixing is not checked by the contract in full: the contract requires a deposit from the shuffler and allows for fraud proofs by each depositor in the event that the key to their deposit has been dropped by the shuffler.

The benefit of this approach is that the more costly cryptographic checks are performed off-chain by the depositors, as opposed to the smart contract. This reduces gas costs for shuffles as well as deposits. The authors quote withdrawals at 113k gas, predating EIP-1108 [CWa]. The largest part of this cost is a custom ECDSA check involving two point-scalar multiplications. Currently, using EIP 1108 would reduce that cost significantly, to ca. 40k gas.

On the other hand, the protocol used requires that every depositor needs to check every shuffle: checking that a particular public key has not been dropped requires the corresponding private key. This places a high burden on users, as they need to be online to perform the corresponding checks in a timely manner. Additionally, relaxing the time limit for fraud proofs would slow down the protocol.

A further issue is the identity of the shufflers: Mixer security assumes that honest parties will always exercises their right to shuffle, but that creates high participation costs and exacerbates the problem of a receiver being able to fund the withdrawal transaction: they now need to fund the shuffle as well, or hope that the sender will perform it (this is a non-issue if one is mixing their own funds).

SNARK-based proofs can be leveraged to reduce participation requirements for proof checking, but ensuring that shufflers are using good-quality randomness is a tougher issue to handle. A design using SNARK-based shuffle proofs and a set of semi-trusted shufflers (in the sense that no number of malicious shufflers can reroute funds, and even 1 honest shuffler can establish privacy) can be produced, but would require significant divergence from the described protocol.

5 Challenges

5.1 Harmony-Specific Concerns

5.1.1 Staking & Privacy

The staking mechanism of Harmony relies on an auction mechanism that assigns verification slots to bidders. Holders of small amounts of stake can participate via delegating their stake to others. Absent an additional delegation mechanism, this precludes private funds from participating in the staking process. The impact of this restriction is not well understood, but as a conservative approximation it could be modeled by reducing the amount of stake held by honest parties by the amount of funds held in the pool of private funds.

Following this, there needs to be a balance of incentives with regards to keeping funds private: on the one hand maintaining a particularly large pool may impact the working of the stake auction. On the other, if the pool is too small, large transactions are made conspicuous by their size. In this regard, solutions such as Möbius are the most safe as withdrawing funds is mandatory and happens in a predictable timeline. Incentive structures such as "Anonymity Mining" as implemented by TornadoCash [] need careful consideration before deployment on Harmony: the original design targets Ethereum which, at the time of writing, is based on a proof of work mechanism. A different option is to provide avenues for contracts to participate in delegation, but additional research is needed in that front.

5.1.2 Sharding

As Harmony uses sharding, any on-chain solutions need to be adjusted to account for it. As such, this would either require all private transactions to be treated as cross-shard thus reducing the benefits of sharding, or to run a separate instance on each shard, reducing the anonymity provided to a per-shard basis. Solutions that are implemented as smart contracts are effectively "cross-shard", but require no design changes.

5.2 Nullifier Storage

A common requirement amongst most privacy solutions is that of storing the list of nullifiers used in previous transactions. Without a privacy requirement, one would simple delete used inputs or simply adjust account balances to handle transaction inputs. However, simply deleting inputs would violate privacy. Instead, nullifiers or key images must be stored to explicitly prevent inputs being re-utilized. Zether utilizes an account model, but the requirement for concurrency within an epoch implies that a short-term storage of nullifier is still necessary, but limited.

Möbius 4.1 is similar to Zether in this regard as an emptied-out ring can be reused. Unlike Zether, Möbius does not specify the notion of an epoch with

regard to withdrawals, but can be extended to provide a fixed time period for withdrawals to be made.

For native support, a sorted list is sufficient and simple to implement with predictable performance characteristics. Bloom filters may also be utilized to improve performance, and enable different caching strategies if in-memory storage becomes a concern.

For smart contacts, the storage cost under a naive approach may be considerable: at a minimum, storing a nullifier requires writing to a previously null address (to write the value itself) as well as updating the index of the last-written position. Additionally, checking that a nullifier has not been re-used is linear with regard to the number of previous withdrawals[8].

A more efficient but more complex approach [GWW] is to use sparse Merkle trees (Sect. 2.4.3). In this approach, checking for non-inclusion requires a Merkle tree inclusion proof, and adding a nullifier to the set requires a Merkle tree insertion (which overlaps with the inclusion proof). The checks for both operation can be done in the open as well as inside a circuit, as the value (in this case also the index) to be inserted is public knowledge. A hybrid approach involves maintaining a rolling buffer of "recent" nullifiers as contract storage and periodically updating a sparse Merkle tree in batches via a helper function that verifies a number of leaf insertions and at the same time clears the buffer.

5.3 Side-Channels and Heuristic Attacks on Privacy

A number of research efforts [Wu+22; TBP20; AZ19; Mei+13; Kap+18] identify techniques that can be used to link or even de-anonymize blockchain transactions transactions. The "account" model makes this even more clear as transactions by the same entity are linked explicitly (instead of the UTXO model, where the link, however unambiguous is implicit).

As a trivial example, consider a user who uses a service to take funds from their primary account to a secondary one. After a period of time, the user wishes to make a transaction that requires more funds than the ones available to either account. The user does not require the transaction to be private. A careless option would be to fund the transaction by transferring all funds from the secondary account to the primary one, effectively winding it down. It is clear however that this would link the two accounts, retroactively breaking privacy.

While some heuristics are application specific, others are applicable to a wide number of solutions. Careful choices in implementation and user interface can mitigate a number of them, but used education is also necessary to prevent trivial misuses: if a user believes that they are able to "anonymize" funds sourced from their primary (transparent) address and then return them to the same address, they will be unsuccessful.

Specific side-channels and heuristics include:

[8] Sorting the nullifier list does not help: typically, the number of writes to the list is equal to the number of checks.

- Unique gas price [Wu+22] (for smart contract based solutions): If a user manually enters a gas price, the exact value might leak information (e.g. if the fee is a round number when converted to a particular fiat currency), and also be used to link transactions (if the user habitually repeats patterns).
- Address Matching: This represents the trivial scenario where a user mixes funds with the same return address as the originating one.
- Leaks by value: In systems where arbitrary values are allowed, and values are not masked privacy is necessarily limited to a small anonymity set. If a particular value is actually unique in the system, then privacy is impossible. A potential solution is to enforce fixed denominations (as in TornadoCash 4.2) but that is not without complications; if the entirety of a sum of funds is withdrawn at the same time, then the total value is made apparent, and can be cross-checked against deposits.
- Incentive-based linking: [Wu+22] mention that TornadoCash offer rewards in the form of "anonymity points" to users who keep their deposits for a period of time before withdrawing. As the points are a function of time, this poses the following two potential problems.

  First, if the points only correspond to a single deposit, they reveal the time difference between deposit and withdrawal. If this time window can only match a limited number of potential pairs, this breaks (or severely impacts) privacy.

  Second, further leakage may occur if the address claiming the result is the same as the one making the deposit or withdrawal.

5.3.1 Gas Funding

In solutions that rely on smart contracts one must issue a withdrawal transaction in order to receive all or part of the funds initially deposited. However, this requires an amount of gas (or equivalently, funds) to be provided for the transaction to go forward. This creates a chicken and egg problem, where in order to establish funds in an account unrelated to the original one, one needs to already possess funds in such an account. Relayers, as described in TornadoCash for instance, can provide a partial solution by receiving withdrawal request out of band and issuing the transaction in exchange of a fee.

EIP-86 describes a system that allows transaction fees to be paid without requiring an address to have funds initially, effectively generalizing the notion of relayers. It is, however, not trivial to arrive at a solution that protects miners from abuse while at the same time ensuring liveness.

6 Evaluation & Recommendations

6.1 On-Chain vs Smart Contracts

A major decision point for developing a privacy solution is that of choosing between adding direct on-chain functionality, or indirectly providing it via smart contracts. An on-chain solution can be more performant by reducing overheads and can be customized to integrate with operations such as staking or validating. However, it would also require that development is in sync with the core protocol and increase the size of the code-base. Most relevant, however, is that the most established solutions use UTXO-based as opposed to account-based semantics. Zether could be used as a design guide to bridge the gap, by attaching encrypted balance values to accounts and introducing a form of ring-like transfers, where zero-valued transaction outputs would be sent to third-party accounts to establish an anonymity set. The design and implementation scope of such a solution would be considerable. Maintaining a separate UTXO-ledger would similarly add high complexity for not much benefit. Quisquis is the most fitting design due to its use of the account model but would require changes to the signature infrastructure or implementation of a parallel ledger. Furthermore, issues with concurrent transactions and client updates should be considered.

As Harmony shares the Ethereum EVM, smart-contract based solutions can be readily used. While the overhead for smart contract execution is significant, efficient SNARKS such as Groth16 or PLONK can easily fit within current block gas constraints. Furthermore, development of a smart contract can be decoupled from the main chain, or may be entirely community-led. Core development efforts can be used to augment smart contracts by providing additional functionality via precompiled contracts. EIP-1108 [CWa] produced a vast improvement on the practicality of ECC and pairing operations on smart contracts, and similar proposals [CWb] can lead to further improvements.

6.2 Smart Contract Proof Systems

For smart contracts, cryptographic operations are quite expensive, so significant care is required to select a proof system that achieves acceptable performance and proof size. Bulletproofs, sigma protocols and direct computation can be used for operations involving very limited data, e.g., ring signatures over small rings in the case of Möbius or single signatures in the case of mixeth. In Zether, where one needs to perform somewhat more complex operations on a number of elements, gas costs immediately become a concern, even taking EIP-1108 [CWa] into account (without it, costs are almost-prohibitive). Thus, such solutions are best paired with a minimalist approach, where the provided functionality and size of anonymity set are kept small.

SNARK-based systems on the other hand, are able to scale much better, at the cost of establishing a trusted structured reference string (SRS). As such, we are able to maximize the anonymity set at little cost. Additional functionality (e.g. confidential values and internal transactions) can also be handled, sidestep-

ping some, but not all of the issues with transaction funding (Sect. 5.3.1.)

6.3 Smart Contract Solutions

A solution akin to TornadoCash appears to be the most attractive in terms of anonymity pool size and operational efficiency. Furthermore, it can be augmented with additional functionality (as in Nova) equivalent to ZCash. The proof complexity of Zether is a significant issue, and re-engineering it to use a SNARK would remove the benefit of trasparency.

As an alternative, Möbius can provide a limited[9] level of privacy with a very small code footprint, no trust requirements, and minimal impact on the active stake distribution.

6.4 Additional Provisions

For smart contract based solutions, a number of improvements can be facilitated by changes to the EVM, in two main areas. First, by providing precompiled contracts for common yet complex cryptographic operations. This can be fruitful by reducing gas costs for the contract itself [CWb], or by allowing operations to be performed cheaply in the contract reducing load on the proof system. A caveat is that the gas costs of Harmony are significantly lower than those of Ethereum, in turn reducing the impact of cost reductions. Furthermore, implementing an EIP that has not been finalized can lead to incompatibilities if the proposal changes after being implemented.

A second area of interest lies in the ability to alter gas funding requirements for smart contracts. For example, EIP-86 (Sect. 5.3.1) enables validators to execute code (up to a bounded gas cost) on the expectation of payment once execution concludes successfully. While this solves the issue of funding private withdrawals without relayers, it also allows for a degree of denial of service attacks with a wide impact. As such, the initial recommendation is to keep the current funding mechanism.

Lastly, mechanisms that can facilitate selective disclosure or audits can be considered. A degree of such functionality can be achieved via the use of viewing keys, as in ZCash [Hop+], with limited overhead.

References

[Cha81] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[FS87] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Advances in Cryptology—Crypto'86*. Springer. 1987, pp. 186–194.

[9] Due to its restricted pool size.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The knowledge complexity of interactive proof systems". In: *SIAM Journal on computing* 18.1 (1989), pp. 186–208.

[BR93] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of partial knowledge and simplified design of witness hiding protocols". In: *Annual International Cryptology Conference*. Springer. 1994, pp. 174–187.

[Nef01] C Andrew Neff. "A verifiable secret shuffle and its application to e-voting". In: *Proceedings of the 8th ACM conference on Computer and Communications Security*. 2001, pp. 116–125.

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. "1-out-of-n signatures from a variety of keys". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2002, pp. 415–432.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. "Traceable Ring Signature". In: *Public Key Cryptography – PKC 2007*. 2007.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*. Vol. 4948. Lecture Notes in Computer Science. Springer, 2008, pp. 1–18. DOI: 10.1007/978-3-540-78524-8_1. URL: https://iacr.org/archive/tcc2008/49480001/49480001.pdf.

[BG12] Stephanie Bayer and Jens Groth. "Efficient zero-knowledge argument for correctness of a shuffle". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2012, pp. 263–280.

[Bit+13] Nir Bitansky et al. "Recursive Composition and Bootstrapping for SNARKS and Proof-Carrying Data". In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '13. Palo Alto, California, USA: Association for Computing Machinery, 2013, pp. 111–120. ISBN: 9781450320290. DOI: 10.1145/2488608.2488623. URL: https://doi.org/10.1145/2488608.2488623.

[FZ13] Matthew Franklin and Haibin Zhang. "Unique ring signatures: A practical construction". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2013, pp. 162–170.

[Mei+13] Sarah Meiklejohn et al. "A fistful of bitcoins: characterizing payments among men with no names". In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 127–140.

[Van13] Nicolas Van Saberhagen. "CryptoNote v 2.0". In: (2013).

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "Coinshuffle: Practical decentralized coin mixing for bitcoin". In: *European Symposium on Research in Computer Security*. Springer. 2014, pp. 345–364.

[Sas+14] Eli Ben Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: *2014 IEEE symposium on security and privacy*. IEEE. 2014, pp. 459–474.

[Boo+15] Jonathan Bootle et al. "Short accountable ring signatures based on DDH". In: *European Symposium on Research in Computer Security*. Springer. 2015, pp. 243–265.

[GK15] Jens Groth and Markulf Kohlweiss. "One-out-of-many proofs: Or how to leak a secret and spend a coin". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 253–280.

[MP15] Gregory Maxwell and Andrew Poelstra. "Borromean ring signatures". In: 8 (2015), p. 2019.

[Noe15] Shen Noether. *Ring Signature Confidential Transactions for Monero*. Cryptology ePrint Archive, Report 2015/1098. `https://ia.cr/2015/1098`. 2015.

[DPP16] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. "Efficient sparse merkle trees". In: *Nordic Conference on Secure IT Systems*. Springer. 2016, pp. 199–215.

[Gro16] Jens Groth. "On the size of pairing-based non-interactive arguments". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2016, pp. 305–326.

[Jed16] TE Jedusor. *Mimblewimble*. Defunct hidden service. Copy retrieved from `https://github.com/mimblewimble/docs/wiki/Mimblewimble-origin`. 2016.

[Öst16] Rasmus Östersjö. *Sparse Merkle trees: Definitions and space-time trade-offs with applications for balloon*. 2016.

[Poe16] Andrew Poelstra. "Mimblewimble". In: (2016).

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. "Scalable multi-party computation for zk-SNARK parameters in the random beacon model". In: *Cryptology ePrint Archive* (2017).

[Sun+17] Shi-Feng Sun et al. "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero". In: *European Symposium on Research in Computer Security*. Springer. 2017, pp. 456–474.

[Ben+18] Eli Ben-Sasson et al. "Scalable, transparent, and post-quantum secure computational integrity". In: *Cryptology ePrint Archive* (2018).

[Bün+18] Benedikt Bünz et al. "Bulletproofs: Short proofs for confidential transactions and more". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 315–334.

[Kap+18] George Kappos et al. "An empirical analysis of anonymity in zcash". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 463–477.

[MM18] Sarah Meiklejohn and Rebekah Mercer. "Möbius: Trustless tumbling for transaction privacy". In: (2018).

[AZ19] Nasser Alsalami and Bingsheng Zhang. "SoK: A Systematic Study of Anonymity in Cryptocurrencies". In: *2019 IEEE Conference on Dependable and Secure Computing (DSC)*. 2019, pp. 1–9. DOI: 10.1109/DSC47296.2019.8937681.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. https://ia.cr/2019/1021. 2019.

[Fau+19] Prastudy Fauzi et al. "Quisquis: A new design for anonymous cryptocurrencies". In: *International conference on the theory and application of cryptology and information security*. Springer. 2019, pp. 649–678.

[Jiv19] Aram Jivanyan. *Lelantus: A New Design for Anonymous and Confidential Cryptocurrencies*. Cryptology ePrint Archive, Report 2019/373. https://ia.cr/2019/373. 2019.

[Mal+19] Mary Maller et al. "Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2111–2128.

[TBP20] Florian Tramèr, Dan Boneh, and Kenneth G. Paterson. *Remote Side-Channel Attacks on Anonymous Transactions*. Cryptology ePrint Archive, Report 2020/220. https://ia.cr/2020/220. 2020.

[Yue+20] Tsz Hon Yuen et al. "Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 464–483.

[CG21] Pyrros Chaidos and Vladislav Gelfer. "Lelantus-CLA". In: *Cryptology ePrint Archive* (2021).

[Dia21] Benjamin E Diamond. "Many-out-of-Many Proofs and Applications to Anonymous Zether". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1800–1817.

[Fic+21] Ádám Ficsór et al. *WabiSabi: Centrally Coordinated CoinJoins with Variable Amounts*. Cryptology ePrint Archive, Report 2021/206. https://ia.cr/2021/206. 2021.

[Koh+21] Markulf Kohlweiss et al. "Snarky ceremonies". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2021, pp. 98–127.

[Wu+22] Mike Wu et al. "Tutela: An Open-Source Tool for Assessing User-Privacy on Ethereum and Tornado Cash". In: *CoRR* abs/2201.06811 (2022). arXiv: 2201.06811. URL: https://arxiv.org/abs/2201.06811.

[Alea] Roman Storm Alexey Pertsev Roman Semenov. *Tornado Cash Github*. URL: https://github.com/tornadocash/tornado-core.

[Aleb] Roman Storm Alexey Pertsev Roman Semenov. *Tornado Cash Privacy Solution*. URL: https://tornado.cash/audits/TornadoCash%5C_whitepaper%5C_v1.4.pdf (visited on 2019).

[CWa] Antonio Salazar Cardozo and Zachary Williamson. *EIP-1108: Reduce alt_bn128 precompile gas costs*. URL: https://eips.ethereum.org/EIPS/eip-1108 (visited on 2018).

[CWb] Antonio Salazar Cardozo and Zachary Williamson. *EIP-3068: Precompile for BN256 HashToCurve Algorithms*. URL: https://eips.ethereum.org/EIPS/eip-3068 (visited on 2020).

[GWW] Ariel Gabizon, Zac Williamson, and Tom Walton-Pocock. *Aztec Yellow Paper*. URL: https://hackmd.io/@aztec-network/ByzgNxBfd (visited on 2021).

[Hop+] Daira Hopwood et al. "Zcash protocol specification". In: ().

[KV] Dmitry Khovratovich and Mikhail Vladimirov. *Tornado Privacy Solution Cryptographic Review*. URL: https://tornado.cash/audits/TornadoCash_cryptographic_review_ABDK.pdf (visited on 2019).

[Maxa] Greg Maxwell. *CoinJoin: Bitcoin privacy for the real world*. https://bitcointalk.org/index.php?topic=279249.0.

[Maxb] Greg Maxwell. *Confidential transactions*. https://people.xiph.org/~greg/confidential_values.txt.

[] *Tornado Cash Anonymity Mining*. URL: https://docs.tornado.cash/tornado-cash-classic/anonymity-mining.